Formal Modeling and Analysis of Scientific Workflows Using Hierarchical State Machines

Ping Yang
Computer Science Department
Binghamton University
Binghamton, NY, 13902, USA
pyang@cs.binghamton.edu

Zijiang Yang
Computer Science Department
Western Michigan University
Kalamazoo, MI, 49008, USA
zijiang.yang@wmich.edu

Shiyong Lu
Department of Computer Science
Wayne State University
Detroit, MI, 48202, USA
shiyong@wayne.edu

Abstract

Scientific workflows have recently emerged as a new paradigm for representing and managing complex distributed scientific computations and data analysis, and have enabled and accelerated many scientific discoveries. Many scientific workflows are distributed and collaborative as they result from some collaborative research projects that involve a number of geographically distributed organizations. In these workflows, information flow control becomes a key security problem. In this paper, we propose to model a scientific workflow using a hierarchical state machine and present techniques for verifying and controlling information propagation in scientific workflow environments based on hierarchical state machines. To the best of our knowledge, this is the first effort for information flow analysis in the area of scientific workflows.

1. Introduction

Today, many scientific discoveries are achieved and accelerated by scientific workflows, which have recently emerged as a new paradigm for representing and managing complex distributed scientific computations and data analysis [31]. A scientific workflow is a formal specification of a scientific process, which represents, streamlines, and automates the steps from dataset selection and integration, computation and analysis, to final data product presentation and visualization. A Scientific Workflow Management System (SWMS) supports the specification, execution, re-run, and monitoring of scientific processes. The scientific workflow paradigm not only enables a scientist to focus on the science itself rather than underlying computation resource and data management, but also facilitates the exploratory process and result reproducibility [22].

While business workflows are typically control-flow oriented, scientific workflow tend to be dataflow-oriented. More specifically, in a business workflow model, the design of a workflow focuses on how execution control flows from one task to another (sequential, parallel, conditional, loop, or event-condition-action triggers), forming various *control flows*; in a scientific workflow model, the design of a workflow focuses on how the input data are streamlined into various data analysis steps using data channels to produce various intermediate data products and final workflow data products, forming various *data flows*.

As more and more scientific research projects become collaborative in nature, scientific workflows can also be collaborative and involve a number of geographically distributed organizations. In these workflows, information can flow from one workflow task to another and across different organizations in the forms of dataflows and file, database, and Web access. As a result, information flow control [32] becomes a key security problem in such scientific workflow environments. Information flow control is different from traditional access control: while access control mechanisms prevent information from being accessed by unauthorized users, they do not prevent an authorized user to pass on the accessed information to another unauthorized user. Thus, the lacking of an information flow control mechanism might result in scientific workflows in which information is leaked to unauthorized workflow tasks, users, or hosts. Formal modeling and verifying that a scientific workflow complies with a given information flow policy is an important and challenging problem. We motivate our research by the following example.

A motivating example. Let us consider a human epilepsy collaborative research study that is conducted by a collaboration among Children's Hospital of Michigan (H_1), Henry Ford Hospital in Detroit (H_2), a number of epilepsy researchers R from the Department of Neurology, and a number of computer scientists C from the Department of Computer Science at Wayne State University.

The goal of the project is to identify fiber tract patterns and their associations with abnormal cortical regions. Figure 1 shows various principals, datasets, and programs involved. In the figure, an oval represents a principal within the system; the system has the following principals: H_1, H_2, p, R, W, and C. Arrows in the diagrams represent the directions of information flows between principals; a square box represents a piece of data that is flowing. Double ovals represent trusted software programs.

Each principal O defines its own information policy that specifies a set of principals $access(O)$ that can access data of O. In our example, two pieces of data are needed for each patient p: the patient p's PET data that can be retrieved from the Children's Hospital, and his/her fMRI data that can be retrieved from the Henry Ford Hospital. To control the propagation of these pieces of data, each hospital limits the data only to the patient and the hospital itself. This is represented by $\{p, H_1\}$ and $\{p, H_2\}$, respectively. The program "Retrieve Data" is a trusted program which is run by the hospital to retrieve the data and can be read by both p and researchers R. A researcher from R can invoke a scientific workflow W (to be described later) to identify each patient p's fiber tract pattern and abnormal cortical regions. The execution of the workflow also needs the interaction from a computer scientist which specifies appropriate parameters with assistance from some knowledge base. The knowledge base is secured by label $\{C\}$ and the interaction information between the workflow and computer scientists are protected by $\{C, W\}$ to ensure that these information are only readable to C and W. Finally, the result of the study is only readable to the patient and R. This is achieved by label $\{R, p\}$.

A sample human epilepsy scientific workflow is shown in Figure 2. In this example, the two *Load* actors enable an e-scientist to choose two files, one for a PET data file and the other for a fMRI data file, via the two input parameters, *Source1* and *Source2*, respectively. The two *Stage* actors upload these two files to two remote hosts, *CS_HOST1* and *CS_HOST2*, respectively. Parameters *Target1* and *Target2* are used to specify the target file names. Two instances of a coclustering subworkflow previously proposed by us [28] will be executed at *CS_HOST1* and *CS_HOST2*, respectively, to conduct independent analysis with different input parameter values for *mp*, which specifies the mutation rate of the algorithm. Finally, the outputs of the two subworkflows are fed to the inputs of the last actor *Visualization* at host *CS_HOST3*, which visualizes a 3-D brain model with identified fiber tract patterns. In this workflow, if the *Visualization* actor is changed to an actor that is developed by H_1, then the information flow policy is violated since patient p's PET is not supposed to be disclosed to H_2 and his/her fMRI data is not supposed to be disclosed to H_1. A manual analysis of such a violation is sophisticated and challenging for large-scale and hierarchical scientific workflows.

In this paper, we propose a model *hierarchical state machine for scientific workflow* (HSMSW), which is based on *hierarchical state machines* (HSM) [5, 4, 3], to formally model and verify scientific workflows, and control information propagation in scientific workflows. A hierarchical state machine is an extension to a finite state machine, in which a state can be an ordinary state or a superstate, which is a state machine itself. Such a nesting structure makes

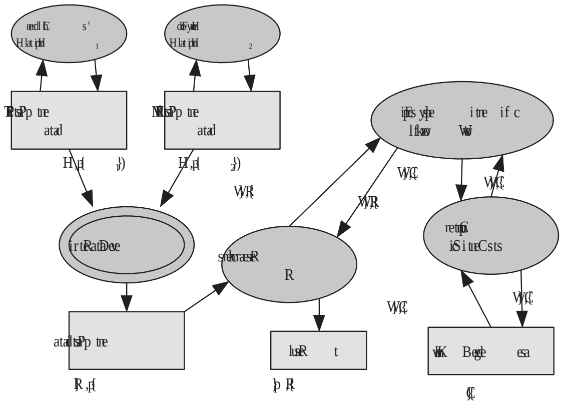


Figure 1. A medical scenario for human epilepsy research.

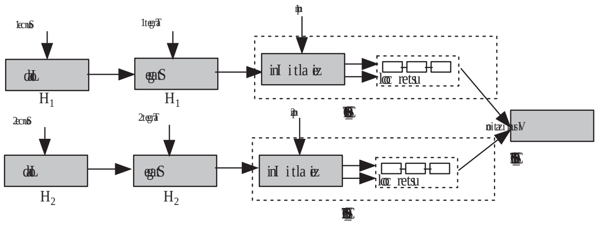


Figure 2. A human epilepsy scientific workflow.

hierarchical state machines a natural formalism for modeling scientific workflows where actors may be composed of multiple sub-actors. Furthermore, compared to a finite state machine, a hierarchical state machine provides a more modular and succinct system representation and allows us to model large systems. For example, if a component is used more than once, we only need to specify it once and reuse it in different contexts. HSMSW extends hierarchical state machines with a notion of *connection channel* which is used to model control and data flows in scientific workflows. HSMSW also extends hierarchical state machines with modular features such as variable scoping.

2. Hierarchical State Machines

A hierarchical state machine (HSM) [5] K is a tuple $\langle K_1, \ldots, K_n \rangle$ of modules, in which each module K_i has the following components:

1. A finite set N_i of *states*.

2. A finite set B_i of *sub-modules*, representing superstates. The sets N_i and B_i are pairwise disjoint.

3. A subset I_i of N_i, called *entry states*.

4. A subset O_i of N_i, called *exit states*.

5. An indexing function $Y_i : B_i \mapsto \{i+1 \ldots n\}$, which maps each sub-module b of K_i ($b \in B_i$) to j with $j > i$. If $Y_i(b) = j$, then b is a reference to the definition of module K_j, each pair (b, u) with $u \in I_j$ is called *a call* of K_i, and each pair (b, v) with $v \in O_j$ is called *a return* of K_i.

6. A transition edge relation E_i consisting of pairs (u, v), in which the source u is either a state or a return of K_i and the sink v is either a state or a call of K_i.

7. A labelling function $L_i^g : E_i \mapsto G$ that maps each edge of the K_i to a condition guard.

8. A labelling function $L_i^a : E_i \mapsto A$ that maps each edge of the K_i to an action.

A hierarchical state machine can be "flattened" to a finite state machine by recursively substituting each module with the associated sub-module references. Since the references of the same sub-module can reside in different modules, each module can appear in a number of different contexts. It has been shown in [5] that flattening may cause exponential blow-up, especially when there are many references pointing to the same module. A module is called a *top-level module* if it does not have parent modules. The transitions are edges connecting states and modules with one another. Each transition is of the form $s_1 \overset{G/A}{\to} s_2$ such that, given a source state s_1, if guard G (condition) holds, then a set of variable assignments A will be performed, leading to the target state s_2.

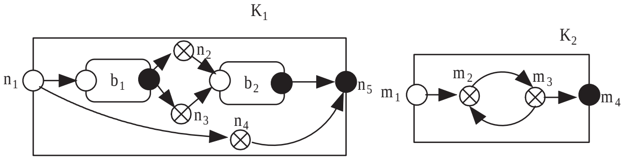


Figure 3. A hierarchical state machine

Figure 3 illustrates an example of a hierarchical state machine $K = \langle K_1, K_2 \rangle$. We use squares to denote modules and round-corner rectangles to denote module references. We use $\circ$, $\bullet$, and $\otimes$ to denote entry states, exit states, and internal states (states that are neither entry nor exit states), respectively. The guards and actions are omitted in the figure. By distinguishing between modules and module references, we may control the degree of sharing of modules. There are two modules in the graph, K_1 and K_2. K_1 is a top-level module. It contains five states $N_1 = \{n_1, n_2, n_3, n_4, n_5\}$ and two sub-modules $B_1 = \{b_1, b_2\}$, both of which are references to the definition of K_2. $Y_1(b_1) = Y_1(b_2) = 2$. The entry states of K_1 are $I_1 = \{n_1\}$ and exit states are $O_1 = \{n_5\}$. (b_1, m_1) and (b_2, m_1) are the calls of K_1. (b_1, m_4) and (b_2, m_4) are the returns of K_1. Each edge in K_1 is a pair (u, v) where u is either a state of K_1 or a return of K_1 and v is either a state of K_1 or a call of K_1. K_2 is a module that contains a finite state machine with four states and four transitions. We assume that all references form an acyclic graph.

3 Formal Modeling of Scientific Workflows Using Hierarchical State Machines

In this section, we present Hierarchical State Machine for Scientific Workflows (HSMSW), a hierarchical state machine extended specifically for scientific workflows. The HSMSW model is different from the traditional hierarchical state machine model given in Section 2. First, HSMSW resembles modules using connection channels. A connection channel connects an exit state of one module with an entry state of another module. Each connection channel is labelled with a guard, which specifies the condition under which the data can be passed through this channel. Connection channels are different from transitions in hierarchical state machines: If a module K_1 has two incoming connection channels from modules K_2 and K_3, respectively, then K_1 cannot be executed until both K_2 and K_3 have finished their execution. In contrast, transitions represent "or" relation: if a state n_1 has two incoming transitions from n_2 and n_3, then n_1 can be reached if n_2 or n_3 is reached and the condition in the corresponding transition holds. Second, actions in the transitions are not restricted to a set of assignments, instead, actions may also be file, database, and Web access. Finally, HSMSW extends the hierarchical state machine with variable scoping.

It is straightforward to model scientific workflows using HSMSWs. Each atomic actor (i.e., actor that does not have sub-actors) is modeled using a module containing a finite state machine modeling the internal structure of the actor. Each occurrence of an atomic actor is modeled as a reference to the corresponding module of the actor. The abstract level of the finite state machine depends on the properties we want to verify. One of such finite state machines can be constructed as follows: states represent the control locations associated with a set of variables; actions are sequences of sequential statements; and guards are conditions in conditional or loop statements. The input and output ports in a scientific workflow are modeled using entry and exit states in HSMSW, respectively. Data and control flows between two actors A_1 and A_2 are modeled as connection channels between the two HSMSWs modeling A_1 and A_2. A composite actor (i.e., actor that contains sub-actors) is modeled as a module with all subactors modeled as modules in HSMSW.

We now illustrate how to model the scientific workflow shown in Figure 2 using HSMSW. Each atomic actor, *Load*, *Stage*, *Initialize*, and *Visualization*, is modeled using a module containing a finite state machine that models the internal structure of the actor. Each occurrence of these actors is modeled as a reference to the corresponding module. The composite actors located at *CS_HOST1* and *CS_HOST2* are modeled using modules, each of which contains references referencing to modules modeling *Initialize* and *Cocluster*. A data channel between two actors, e.g. between actors *Load* and *Stage*, is modeled using a connection channel.

The execution of an HSMSW starts when data arrives at all entry states of the top-level modules. The execution terminates if all processed data have been transferred back to the exit states or the execution gets stuck as all outgoing transitions are disabled.

4. Verification and Information Flow Control of Scientific Workflows

In this section, we present techniques for using HSMSW to formally verify properties of scientific workflows and to control information propagation in scientific workflow environments. Our framework can also be used to collect file access provenance for scientific workflows. Here, we assume that the permission for executing each actor is obtained from appropriate access control.

4.1. Verification of Scientific Workflows

Alur and Yang [4] proposed a symbolic model checking algorithm for hierarchical reactive machines. A hierarchical reactive machine is a hierarchical state machine with modular features, such as variable scoping and exceptions. The algorithm is directly applied to hierarchical reactive machines with more efficiency than first flattening the hierarchical reactive machine into a finite state machine and then performing model checking on the transformed finite state machine. Similar to hierarchical state machines, hierarchical reactive machines use transitions instead of connection channels to connect two modules. Thus, the algorithm cannot be directly applied to verify HSMSW.

In this section, we extend the algorithm in [4] to deal with HSMSW. As in [4], the guards in transitions are encoded symbolically using binary decision diagrams (BDDs) [12]. The state space of a finite state machine is partitioned into a set of state regions, each of which represents a set of states. Each state region is represented by a single BDD. The algorithm then computes a set of reachable states. If a top module K_i gets control for the first time, it computes the set of reachable states from its entry states until it gets stuck at a state. A state c is called a *stuck state* if none of the guards leaving c are satisfied and we say that the control *gets stuck* at c. An entry state p is called a *stuck entry state* if p waits inputs from other modules. When a stuck entry state p is encountered, the model checker backtracks to traverse other paths until p does not get stuck or no new states can be reached.

After every top module has finished the first computation, a stuck set S is constructed which consists of stuck states at each top module. The algorithm then constructs a *current onion ring* for each top module K_i based on S. The current onion ring maps the states where the control got stuck during the last computation at K_i to newly reached state sets obtained from image computations (i.e., single-step reachability computation) at top modules other than K_i. By applying the image computation to the current onion ring of K_i, the control may continue from those stuck states (for example, the value of global variables at a stuck state of a module may be changed by another module). The algorithm terminates if all the onion rings for top modules are empty, which means that no new states can be reached.

4.2. Information Flow Control of Scientific Workflows

In this section, we consider hosts as principals and leave other finer-grained information flow control for future work. In our model, objects are system resources such as files, databases, or tuples of a database. Each object has an object ID. Object O in host h is specified as $h : O$. Each host h has a *host information flow policy*, $access(h)$, which specifies the set of principals that can access the objects in h. However, this policy can be overruled by an *object information flow policy*, $access(O)$, which specifies the set of principals that can access O. Therefore, for each object O in host h, the set of principals that can access O is $access(h)$ if $access(O)$ is not specified, and $access(h) \cap object(O)$ otherwise. We say that *information flows from object O_1 to object O_2* if information stored in O_1 is transferred to O_2 through a sequence of operations, including assignment statements, file reading/writing statements, I/O operations, parameter passing, and file transfers.

Our Information flow control technique consists of two stages: 1) prior to execution, static information flow analysis is performed to ensure that no information will be leaked to unauthorized hosts. During the construction of a scientific workflow, a HSMSW is constructed which can be used to detect violations that are independent of input and output parameter values of the scientific workflow. After the user provides input and output parameter values to the scientific workflow and before the scientific workflow executes, the input and output parameter values are passed to the HSMSW, which allows us to detect violations that depend on input and output parameter values. 2) If no violation is detected at Stage 1), the workflow is executed. During the execution, we keep track of how information propagates and collect a set of object IDs $source(O)$ from which information flows to object O. At the end of execution, $source(O)$ will be written to the provenance store of O. $source(O)$ is used in both stages to control information propagation because a scientific workflow may be executed multiple times with different input datasets and parameter values or multiple workflow runs may access the same object. For example, suppose that during the execution of workflow W_1, information of O_1 is transferred to O_2. As a result, $source(O_2) = \{O_1\}$. Next, workflow W_2 is executed, which reads from O_2 and writes its contents to O_3. To keep track of information propagation, $source(O_2)$ is read as well so that we know O_2 contains the information originated from O_1. After W_2 finishes execution, $source(O_3) = \{O_1, O_2\}$, indicating that object O_3 contains the information that originated from both O_1 and O_2.

For clarity of presentation, we use an abstract syntax of our original XML-based language to illustrate how to construct a HSMSW and to control information flow based on HSMSW. Let S denote the set of statements, C the set of conditions, and let x, y, f, and fp range over variables. The language in its abstract form is given below:

$$\begin{aligned} S \quad ::= \quad & x = y \mid fp = fopen(f) \mid x = fread(fp) \\ & \mid fwrite(x, fp) \mid fclose(fp) \mid S_1; S_2 \\ & \mid \text{if } C \text{ then } S_1 \text{ else } S_2 \end{aligned}$$

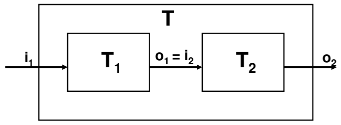
Algorithm 1 Algorithm for constructing finite state machine from atomic actor

```
procedure $CreateFSM(S)$
Create an entry state $T$
$AddMultiStmts(S, T)$

procedure $AddMultiStmts(S_1; S_2, State)$
if $S_1$ is "if $C$ then $S_3$ else $S_4$" then
  $AddMultiStmts(S_3; S_2, State)$
  $AddMultiStmts(S_4; S_2, State)$
  Replace $State \xrightarrow{\alpha} T_1$ with $State \xrightarrow{C/\alpha} T_1$
else
  $Target = AddSingleStmt(S_1, State)$
  $AddMultiStmts(S_2, Target)$
end if

procedure $AddSingleStmt(S, State)$
if $S$ is "$x = y$" then
  $Target = State\overline{\cup}\{x = y\}$
else
  if $S$ is "$fp = fopen(f)$" then
    $Target = State\overline{\cup}\{fp = id(f)\}$
  else
    if $S$ is "$x = fread(fp)$" then
      $Target = State\overline{\cup}\{x = (source(*fp) \cup \{*fp\})\}$
    else
      if $S$ is "$fwrite(x, fp)$" then
        $Target = State\overline{\cup}\{*fp = *fp \cup x\}$
      else
        $Target = State$
      end if
    end if
  end if
end if
if $Target \neq State$ then
  Add $State \overset{true/S}{\longrightarrow} Target$
end if
return $Target$
```

The program consists of a set of statements separated by ";". "$fopen$", "$fread$", and "$fwrite$" represent opening an object, reading from an object, and appending data at the end of an object, respectively. The translation from a scientific workflow to a HSMSW has been given in Section 3. Each atomic actor is translated into a finite state machine using Algorithm 1. The value of each variable v in such a finite state machine is either the ID of an object (when evaluating "$fp = fopen(f)$"), or a set of IDs of objects whose content may be transferred to v. This is different from the finite state machine generated for verification where each variable contains its real value, e.g. content read from an object. The top-level function $CreateFSM(S)$, when given a program S as input, create a state T and calls $AddMultiStmts(S, T)$ to construct a



T_1: input - i_1, output - o_1	T_2: input - i_2, output - o_2
$fp_1 = fopen(i_1);$	$fp_1 = fopen(i_2);$
$x = fread(fp_1);$	$x = fread(fp_1);$
$y = x;$	$y = x;$
$fclose(fp_1);$	$fclose(fp_1)$
$fp_2 = fopen(h_2 : f_2);$	$fp_2 = fopen(h_3 : f_3);$
$fwrite(y, fp_2);$	$fwrite(y, fp_2);$
$fclose(fp_2);$	$o_2 = y;$
$o_1 :=$ "$h_2 : f_2$";	$fclose(fp_2)$

Figure 4. Description of a scientific workflow that consists of an actor T.

finite state machine for S with entry state T. The function $AddSingleStmt(S, State)$ generates transitions for each single statement S with entry state $State$. After assignment statement "$x = y$" is processed, $\{x = y\}$ is added to the state, which means that the value of y is assigned to x. When "$fp = open(f)$" is processed, fp is assigned the ID of the object f. $\overline{\cup}$ is *overriding union* which is defined as follows: $T_1 \overline{\cup} T_2 = T_2 \cup \{t | t \in T_1 \wedge var(t) \notin vars(T_2)\}$ where $var(t)$ is the variable of t and $vars(T_2)$ is the set of variables of T_2. After "$x = fread(fp)$" is processed, $source(*fp) \cup \{*fp\}$ is assigned to x, where $*fp$ is the value of fp, i.e., the object ID stored in fp. "$fwrite$" is similarly handled. "$fopen$" does not change the state. To construct the state machine from statement "if C then S_3 else S_4; S_2" with entry state $State$, we construct state machines from S_3 and S_4 with entry state $State$ and then add condition C to transitions of the state machines constructed whose source state is $State$.

Let $host(h : o) = h$ and $value(x)$ be the value of x. We say that a workflow run R *conforms to the information flow policy* if and only if there does not exist a transition $s \xrightarrow{c/fwrite(x,fp)} s'$ such that $host(fp) \notin access(o_i)$ for some $o_i \in value(x)$. We say a scientific workflow W is *safe* if and only if there does not exist a workflow run R of W such that R violates the information flow policy.

Consider an example workflow given in Figure 4.2. The workflow consists of an actor T, which is composed of two sub-actors T_1 and T_2. T_1 and T_2 execute in sequential order and the output of T_1 acts as the input to T_2. T_1 reads from input i_1 and appends the contents read to file $h_2 : f_2$. T_2 reads from i_2 and writes to $h_3 : f_3$. The HSMSW is constructed as follows: When i_1 is read by T_1, $source(i_1)$ is read and $x = source(i_1) \cup \{i_1\}$. After $y := x$ is evaluated, $y = source(i_1) \cup \{i_1\}$. When the contents of y is written to $h_2 : f_2$ in T_1, $h_2 : f_2 = y = source(i_1) \cup \{i_1\}$, indicating that information flows from i_1 and objects in $source(i_1)$ to $h_2 : f_2$. Next, $h_2 : f_2$ is passed to i_2 and T_2 is evaluated. At the end $h_3 : f_3 = source(h_2 : f_2) \cup \{h_2 : f_2\}$.

Suppose that a user wants to execute the workflow with input $h_1 : f_1$, then i_1 is replaced with $h_1 : f_1$. Assume that initially $source(h_1 : f_1) = source(h_2 : f_2) = source(h_3 : f_3) = \emptyset$. Also, assume that $access(h_1 : f_1) = \{h_2, h_3\}$ and $access(h_2 : f_2) = \{h_3\}$. After $h_1 : f_1$ is read and x is assigned to y, $x = y = \{h_1 : f_1\}$. When $fwrite(y, h_2 : f_2)$ is evaluated, because $h_2 \in access(h_1 : f_1)$, the information flow policy is not violated and hence the contents of y can be written to $h_2 : f_2$. As a result, $h_2 : f_2 = \{h_1 : f_1\}$. After processing T_2, $h_3 : f_3 = \{h_1 : f_1, h_2 : f_2\}$. Since no information flow violations are detected, the workflow can be executed. During the execution, we keep track of how information propagates and update $source(O)$. Now, suppose that we change $access(h_1 : f_1)$ to $\{h_2\}$. Then T violates this policy because there exists a transition $\xrightarrow{true/fwrite(h_2:f_2,h_3:f_3)}$ such that $h_3 \notin access(h_1 : f_1)$ where $h_1 : f_1 \in value(h_2 : f_2)$.

The second stage of information flow control technique gathers a set of original sources $source(O)$ for each object O, which carries extra information needed for scientific workflow provenance metadata other than those captured by dataflows and input and output parameters. In this way, we can capture the derivation history of a data product, including the original data sources, intermediate data products, and the workflow steps that were applied to produce the data product. The scientific workflow provenance metadata can then be stored and queried using a scientific workflow provenance store for reproducibility and recreativity support [13].

5. Related Work

Much work has been done for the modeling and analysis of workflows using formal methods, including Petri nets [1, 2], Workflow nets [15, 36, 34], UML [21, 20], and logics [29, 18]. Among the various properties of workflows, the modeling and analysis of security properties of workflows are particularly important, including access control [9, 39], delegation [37, 6], and separation of duty [26, 10].

Recently, there has been an increasing interest in the applications of workflow technologies in the scientific domain, resulting in *scientific workflows*. In contrast to their business counterparts, which are task-centric and control-flow oriented, scientific workflows are typically data-centric and dataflow-oriented, and thus pose different challenges [31]. Although several scientific workflow management systems [30, 23, 16, 33, 41, 42, 14] have been developed, few of them provide any form of support for verification and analysis, particularly for the modeling and analysis of secure information flow and provenance metadata access, which are very important in the environments of scientific workflows.

The area of information flow analysis has received considerable attention. The lattice model of information flow was first proposed in [7] and [19]. Recently, a number of information flow control techniques have been developed for decentralized systems or web services (e.g. [32, 24, 35, 27, 40]). However, none of these work is done in the context of scientific workflows or uses hierarchical state machines to perform information flow analysis. Furthermore, in addition to support information flow analysis, our framework also allows us to verify many other properties such as deadlock-freedom.

Access control mechanism has also been widely used for preventing data from being leaked to unauthorized users/hosts. A number of access control mechanisms have been proposed for distributed systems, and workflows (e.g. [25, 38, 8]). However, access control mechanism is different from information flow analysis as it controls who can access the data, but does not control how data are propagated after they are obtains.

Recently, the Kepler system extends its actor-oriented modeling framework with frames and templates by borrowing ideas from hierarchical state machines [11]. This approach seamlessly integrates control-flows into a dataflow-based design paradigm without sacrificing the benefits of dataflows. Although Kepler provides a *concrete* hybrid model for designing and executing scientific workflows with both dataflows and control-flow features, verification and information flow analysis are not part of the framework. In contrast, we aim at developing an *abstract* model for scientific workflows based on hierarchical state machines, providing a foundation for formal modeling and analysis of scientific workflows, including information flow analysis.

6 Conclusion and Future Work

In this paper, we proposed to use hierarchical state machines to formally model and verify properties and control information propagation in scientific workflows. We plan to extend our work to deal with *implicit* information flow, support scientific workflows having Web services as their components and secure access control of scientific workflow provenance incorporating the notion of abstraction views [17]. Another direction for future work is the development of algorithms for incremental construction of hierarchical state machines and incremental verification of scientific workflows when minor structural changes are made to workflows during execution.

References

[1] W. Aalst and A. Hofstede. Verification of workflow task structures: A petri-net-baset approach. *Inf. Syst.*, 25(1):43–69, 2000.

[2] N. R. Adam, V. Atluri, and W. kuang Huang. Modeling and analysis of workflows using petri nets. *J. Intell. Inf. Syst.*, 10(2):131–158, 1998.

[3] R. Alur. Formal analysis of hierarchical state machines. *Verification: Theory and Practice*, pages 42–66, 2004.

[4] R. Alur, M. McDougall, and Z. Yang. Exploiting behavioral hierarchy for efficient model checking. In *Conference on Computer Aided Verification*, pages 338–342, 2002.

[5] R. Alur and M. Yannakakis. Model checking of hierarchical state machines. In *Foundations of Software Engineering*, pages 175–188, 1998.

[6] V. Atluri and J. Warner. Supporting conditional delegation in secure workflow management systems. In *Symposium on Access control Models and Technologies*, pages 49–58, 2005.

[7] D. Bell and L. LaPadula. Secure computer system: Unified exposition and multics interpretation. Technical Report ESD-TR-75-306, MITRE Corp., 1975.

[8] R. Bhatti, A. Ghafoor, E. Bertino, and J. Joshi. X-gtrbac: An xml-based policy specification framework and architecture for enterprise-wide access control. *ACM Transactions on Information and System Security*, 8(2):187 – 227, 2005.

[9] R. A. Botha and J. H. P. Eloff. A framework for access control in workflow systems. *Inf. Manag. Comput. Security*, 9(3):126–133, 2001.

[10] R. A. Botha and J. H. P. Eloff. Separation of duties for access control enforcement in workflow environments. *IBM Systems Journal*, 40(3):666–682, 2001.

[11] S. Bowers, B. Ludäscher, A. Ngu, and T. Critchlow. Enabling scientific workflow reuse through structured composition of dataflow and control-flow. In *IEEE Workshop on Workflow and Data Flow for Scientific Applications*, 2006.

[12] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, Aug. 1986.

[13] A. Chebotko, X. Fei, C. Lin, S. Lu, and F. Fotouhi. Storing and querying scientific workflow provenance metadata using an RDBMS. In *International Workshop on Scientific Workflows and Business Workflow Standards in e-Science*, 2007.

[14] A. Chebotko, C. Lin, X. Fei, Z. Lai, S. Lu, J. Hua, and F. Fotouhi. VIEW: a visual scientific workflow management system. In *IEEE International Workshop on Scientific Workflows*, pages 207–208, 2007.

[15] Y. Choi, X. Zhao, and K. Han. Hierarchical reachability analysis for workflow-nets. In *International Conference on Computer Supported Cooperative Work in Design*, pages 556–561, 2006.

[16] D. Churches, G. Gombas, A. Harrison, J. Maassen, C. Robinson, M. Shields, I. Taylor, and I. Wang. Programming scientific and distributed workflow with Triana services: Research articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1021–1037, 2006.

[17] S. Cohen, S. C. Boulakia, and S. B. Davidson. Towards a model of provenance and user views in scientific workflows. In *Data Integration in the Life Sciences*, pages 264–279, 2006.

[18] H. Davulcu, M. Kifer, C. Ramakrishnan, and I. Ramakrishnan. Logic based modeling and analysis of workflows. In *ACM symposium on principles of database systems*, pages 25–33, 1998.

[19] D. E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, 1976.

[20] M. Dumas and A. H. M. ter Hofstede. Uml activity diagrams as a workflow specification language. In *UML*, pages 76–90, 2001.

[21] R. Eshuis and R. Wieringa. Verification support for workflow design with uml activity graphs. In *International Conference on Software Engineering*, pages 166–176, 2002.

[22] J. Freire, C. T. Silva, S. P. Callahan, E. Santos, C. E. Scheidegger, and H. T. Vo. Managing rapidly-evolving scientific workflows. In *International Provenance and Annotation Workshop*, pages 10–18, Chicago, Illinois, U.S.A., May 2006.

[23] Y. Gil, E. Deelman, J. Blythe, C. Kesselman, and H. Tangmunarunkit. Artificial intelligence and Grids: Workflow planning and beyond. *IEEE Intelligent Systems*, 19(1):26–33, 2004.

[24] D. Hutter and M. Volkamer. Information flow control to secure dynamic web service composition. In *International Conference on Security in Pervasive Computing*, 2006.

[25] M. H. Kang, J. S. Park, and J. N. Froscher. Access control mechanisms for inter-organizational workflow. In *Symposium on Access control Models and Technologies*, pages 66–74, 2001.

[26] K. Knorr and H. Stormer. Modeling and analyzing separation of duties in workflow environments. In *SEC*, pages 199–212, 2001.

[27] P. Li and S. Zdancewic. Practical information-flow control in web-based information systems. In *Computer Security Foundation Workshop*, pages 2–15, 2005.

[28] C. Lin, S. Lu, X. Liang, J. Hua, and O. Muzik. Cocluster analysis of thalamo-cortical fiber tracts extracted from diffusion tensor MRI. *International Journal of Data Mining and Bioinformatics (IJDMB)*, 2008. to appear.

[29] S. Lu, A. J. Bernstein, and P. M. Lewis. Automatic workflow verification and generation. *Theor. Comput. Sci.*, 353(1-3):71–92, 2006.

[30] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the Kepler system: Research articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1039–1065, 2006.

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

[32] A. C. Myers and B. Liskov. A decentralized model for information flow control. In *16th ACM Symposium on Operating Systems Principles*, pages 129–142, 1997.

[33] T. Oinn, M. Greenwood, M. Addis, M. N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. R. Pocock, M. S. ger, R. Stevens, A. Wipat, and C. Wroe. Taverna: lessons in creating a workflow environment for the life sciences: Research articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1067–1100, 2006.

[34] Y. Pan, Y. Tang, H. Ma, and N. Tang. Workflow analysis based on fuzzy temporal workflow nets. In *CSCWD (Selected papers)*, pages 545–553, 2005.

[35] N. Ravi, M. Gruteser, and L. Iftode. Information flow control for location-based services. In *3rd Annual International Conference on Mobile and Ubiquitous Systems: Networks and Services*, 2006.

[36] F. L. Tiplea and D. C. Marinescu. Structural soundness of workflow nets is decidable. *Inf. Process. Lett.*, 96(2):54–58, 2005.

[37] J. Wainer, A. Kumar, and P. Barthelmess. Dw-rbac: A formal security model of delegation and revocation in workflow systems. *Inf. Syst.*, 32(3):365–384, 2007.

[38] 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. In *International Symposium on High-Performance Distributed Computing*, 2003.

[39] S. Wu, A. Sheth, J. Miller, and Z. Luo. Authorization and access control of application data in workflow systems. *J. Intell. Inf. Syst.*, 18(1):71–94, 2002.

[40] U. Yildiz and C. Godart. Information flow control with decentralized service compositions. In *IEEE International Conference on Web Services*, pages 9–17, 2007.

[41] Y. Zhao, M. Hategan, B. Clifford, I. T. Foster, G. V. Laszewski, V. Nefedova, I. Raicu, T. Stef-Praun, and M. Wilde. Swift: Fast, reliable, loosely coupled parallel computation. In *IEEE International Workshop on Scientific Workflows*, pages 199–206, 2007.

[42] Z. Zhao, A. Belloum, C. de Laat, P. W. Adriaans, and B. Hertzberger. Distributed execution of aggregated multi domain workflows using an agent framework. In *IEEE International Workshop on Scientific Workflows*, pages 183–190, 2007.