Access Control Policies Specification and Analysis for Multi-Institutional Collaborative Projects

Abhimanyu Chawla, Mahmoud Abdelgawad, and Indrakshi Ray
Department of Computer Science, Colorado State University
Fort Collins, CO, 80523, USA
{abhimanyu.chawla,m.abdelgawad,indrakshi.ray}@colostate.edu

***Abstract*—Most of the scientific research is collaborative in nature, often involving multiple institutions, possibly spanning multiple countries, with a requirement to follow various institutional and legal regulations. Such a collaboration is often dynamic in nature, with users and organizations joining or leaving, data and software resources created or destroyed at the various stages in the scientific workflow. Access policies are set-up manually, often in an ad-hoc manner. This creates two potential problems. First, there is often times over-provisioning of resources. Over-provisioning of resources, including failure to revoke access when the user has left the collaboration, can lead to security breaches. Second, an entity belonging to an external organization often has access to sensitive system resources of an institution. For example, access to a specific task in a workflow may necessitate root access to a server. In short, in research collaborations there is a potential for compromising the confidentiality and integrity of the data, the program, and the system hosting them. A dynamic access control model is needed, whose policies must be correctly specified and automatically enforced at all levels of abstraction to protect the research security of this scientific collaboration. We propose the use of NIST Next Generation Access Control (NGAC) model for protecting the confidentiality and integrity of the research and the associated infrastructure for individual stakeholders in a collaborative project. We describe how to map security requirements into NGAC policies that govern the workflow and the system operations. We define a set of desirable properties for the NGAC policies that protect the workflow and describe consistency checkers for verifying them. Workflows are defined as a set of high-level coordinated tasks; these tasks are executed by low-level system operations. We propose a refinement checker that verifies that a user executing an abstract task does indeed have the permissions to access the low-level system operations that are implementing the task. We demonstrate our approach on various usecases from scientific research workflows and show its effectiveness. Our approach ensures that access control requirements are correctly and consistently specified and enforced at the various levels of abstraction in a dynamic and collaborative environment.**

***Index Terms*—NIST Next Generation Access Control (NGAC), Policy Refinement, Policy Checking, Scientific Workflows, Research Computing Infrastructure (RCI).**

I. Introduction

Scientific research is becoming a collaborative effort involving multiple users, possibly associated with different organizations, each with its own interests in the scientific discovery process [1]. Often times the collaboration is dynamic in nature with users entering or leaving a project and with different types of research products being created at the various stages of the project. The various organizations may be in a competitive relationship and every stakeholders' resources must be carefully managed [2]. Fine-grained access control policies must be specified and automatically enforced for such a dynamic and collaborative project.

We propose the use of the NIST Next Generation Access Control (NGAC) [3] for securing such a dynamic and collaborative project. The constraints for the NGAC model come from the various regulations, including those driven by project needs, the organizational policies of the institutes, state or national domain level policies, if applicable, such as Health Insurance Portability and Accountability (HIPAA).

NGAC is based on attribute-based access control, where access to resources depends on the properties of the users, characteristics of the resources, and attributes of the environment. Moreover, NGAC allows for the access control policies to be changed while they are deployed. It also allows grouping policies, based on the policy provenance.

Scientific projects involve execution of various workflows [4,5]. Each workflow consists of a set of tasks that are coordinated by various control-flow dependencies. Each modification of the scientific workflow or change in the regulations (such as, HIPAA) that govern the access control rules necessitates updating the access control model. These changes must be immediately enforced for the security of the workflow and also for ensuring compliance. We refer to the inconsistencies that arise because of the evolution of policies or workflow as *compliance inconsistencies*. Consequently, we need to propose approaches to verify the absence of such inconsistencies.

Workflow tasks are defined at a high level of abstraction, such as *archiveData* and *submitReport*. These tasks may implicitly invoke one or many system operations, such as *delete*, *format*, and *mount*. If these system operations have access control rules that do not support the workflow tasks, then the abstract tasks cannot be executed safely and correctly. For instance, a task designed to store data could trigger an unauthorized *mount* or *restore* operation on protected storage. Without consistent mapping of authorization of tasks to authorization of system operations, workflows may breach enforcement boundaries and compromise system stability. Thus, it is important to ensure that the authorizations for workflow tasks are consistent with those given for system operations. We refer to the violation of these as *refinement inconsistencies*.

Our key contributions are as follows:

- We provide a formal definition of a scientific workflow and demonstrate how it can be mapped to a set of system operations.
- We describe how to derive the security requirements of the project and map them to NGAC policies that govern the workflow and system operations.
- We present a verification approach that detects compliance inconsistencies when policies or workflow changes.
- Although a user has permission to execute a workflow task, she may not have the permission to perform the low-level system operations that are needed to execute this task. We propose an approach to check for such refinement inconsistencies.
- We demonstrate our approaches on various usecases.

The rest of the paper is organized as follows. Section II shows how we derive security requirements, formulate the NGAC model and the scientific workflow, and discuss the access control for the workflow. Section III shows the approach for verifying compliance consistency of the NGAC policies of the workflows and that of the NGAC policies for system operations. We also show how to verify refinement consistency of authorization policies of the workflows and that of the system operations comprising the workflow. Section IV provides various usecases to illustrate the application of these algorithms for detecting inconsistencies. Section V discusses the computational time complexity of our approach. Section VI reviews the literature relevant to this work, while Section VII summarizes our findings and outlines future research.

II. Access Policies, NGAC, and Workflow

A. Deriving Security Requirements

The security requirements of Research Computing Infrastructure (RCI) for a specific project are derived from various regulations, including the organizational policies of the institutes, funding agencies, and state or national domain-level policies, along with those driven by project needs. These policies are expressed in natural language. Table I illustrates a selection of specific RCI policies extracted from institutional regulatory sources. These 6 policies enforce secure data access, encryption and decryption requirements, and infrastructure control.

Policies P1 and P2 govern who can read, write, and upload sensitive data based on institutional affiliation and role, including Principal Investigator (PI), researcher, and student. Policies P3 and P4 enforce cryptographic standards (e.g., AES-256) for encrypting and decrypting sensitive data, which must be performed by PIs. Policy P5 requires that only PIs and researchers are permitted to access and use High-Performance Computing (HPC) resources. Policy P6 defines administrative privileges for resource provision and executes infrastructure administration-level system operations, such as formatting and partitioning. These policies are translated into the basic elements and relations of the NGAC model [3].

Policy ID	Policy Description
P1	PI can read, write, download, and upload sensitive data
P2	Researchers, students, and contributors from Univ-A, Univ-B, and Inst-Health may read but not write sensitive data
P3	Encryption of sensitive data must be done by PI and performed using a standard encryption algorithm
P4	Decryption of sensitive data requires PI
P5	Access to secure HPC nodes requires PI or researcher. HPC nodes include storage nodes, e.g., ServerD, and GPU computation nodes, e.g., GPU_x1
P6	Only admin can provision HPC resources, and execute infrastructure system operations

TABLE I: Security Requirements

B. Formulating NGAC Model from RCI Policies

NGAC is a fine-grained, dynamic and context-aware access control model standardized by NIST [3]. In the NGAC model, the entities are users, user attributes, resources, resource attributes, and policy classes. Users correspond to active entities requesting access. Resources are objects that need protection. User attributes (resource attributes) describe properties of users (resources) that are important for access control decisions. Policy classes are useful for grouping related policies. In NGAC, the relations are classified as assignments, associations, prohibitions, and obligations. Assignments map a user (resource) to the attributes she possesses. Assignments also describe attribute hierarchy. Association (prohibition) is a mapping between user attributes to resource attributes and are labeled with operations; a user assigned to those attributes is allowed (denied) to perform the operation on the resources possessing the corresponding resource attributes. Obligations are event-based conditions that are executed pre or post-access.

Each attribute has a set of possible values extracted from the security requirements, as shown in Table I. The access rights are also obtained from the security requirements (a portion of which is shown in Table I. Consider policy P2 in Table I. From this policy statement, we can extract the user and resource attributes. Here, the user attributes are `institution` $\in$ {`Univ-A, Univ-B, Inst-Health`} and `role` $\in$ {`researchers, students, contributors`}, and the resource attributes are `resource type = data` and `resource label = sensitive`. The allowed access right is `read` indicated by association and denied access right is `write` denoted by prohibition. This grants users with role `researchers, students, or contributors` in `Univ-A`, `Univ-B`, or `Inst-Health` the privilege to `read` data that is `sensitive`. More details on extracting NGAC attributes and relations from policies written in natural language are found in [6]. The complete sets of user attributes, resource attributes, and access rights extracted from the six policies in Table I are described below.

User Attributes. Let **U**, **UA** be the set of users and their attributes, respectively. **U** = $\{u_1, \ldots, u_n\}$, where n is the

number of users (for instance, John, Alice and Bob). Each user attribute is of the type enumerated.

UA = $\{institute, role\}$, where

- *institute* $\in$ {*Univ-A, Univ-B, Inst-Health*},
- *role* $\in$ {*admin, PI, contributor, researcher, student*},

Resource Attributes. Let **R** be the set of resources (data, software, and hardware) that need protection, and let **RA** be the set of their attributes. **R** = $\{r_1, \ldots, r_m\}$, where m is the number of resources (e.g., "PatientData" and "ServerD"). The attributes are of the enumerated type.

RA = {*own, label, encrypted, type, id*}, where

- *own* $\in$ {*Univ-A, Univ-B, Inst-Health*},
- *label* $\in$ {*public, sensitive*},
- *encrypted* $\in$ {*yes, no*},
- *type* $\in$ {*data, storage, computation*},
- *id* $\in$ {*PatientData, ServerD, GPU_x1*}.

Associations. NGAC association relation (allow) is defined as **Assoc** $\subseteq UA \times 2^{\mathsf{Ops}} \times RA$. The set of associations derived from the 6 policies is described as **Assoc** = {

((role=PI), (read, write, download, upload), ((type=data), (label=sensitive))),
((((institute=Univ-A), (institute=Univ-B), (institute=Inst-Health)), ((role=researcher), (role=student), (role=contributor))), (read), ((type=data), (label=sensitive))),
((role=PI), (encrypt), ((type=data), (label=sensitive))),
((role=PI), (decrypt), ((type=data), (label=sensitive), (encrypted=yes))),
(((role=PI), (role=researcher)), (read, write, execute), (((type=storage), (type=computation)), ((id=ServerD), (id=GPU_x1)))),
((role=admin), (provision, execute), ((type=storage), (type=computation))) }

Prohibitions. NGAC prohibition relation (deny) is defined as **Prohib** $\subseteq UA \times 2^{\mathsf{Ops}} \times RA$. The set of prohibitions extracted from the 6 policies is described as **Prohib** = {

((((institute=Univ-A), (institute=Univ-B), (institute=Inst-Health)), ((role=researcher), (role=student), (role=contributor))), (write), ((type=data), (label=sensitive))),
(((role=admin), (role=researcher), (role=student), (role=contributor)), (encrypt), ((type=data), (label=sensitive)),
(((role=admin), (role=researcher), (role=student), (role=contributor)), (decrypt), ((type=data), (label=sensitive), (encrypted=yes)),
(((role=student), (role=contributor)), (read, write, execute), (((type=storage), (type=computation)), ((id=ServerD), (id=GPU_x1)))),
(((role=PI), (role=researcher), (role=student), (role=contributor)), (provision, execute), ((type=storage), (type=computation))) }

These complete sets of user attributes, resource attributes, associations, and prohibitions are used to govern access control for workflows.

C. Workflow Model

A scientific project involves executing various workflows, which we formally defined below. A workflow consists of a set of tasks connected through operators [7].

Definition (Workflow). *A workflow is defined as:*

$$W = (t | W_1 \otimes W_2 | W_1 \& W_2 | W_1 \# W_2 | if\{C\}\ W_1\ else\ W_2 | while\{C\}\{W_1\})$$

where it can contain sequence, parallel, exclusive, conditional, and iteration operators. In the above definition:

- t *is an atomic task.*
- $\otimes$ *denotes the sequence operator.* $W_1 \otimes W_2$ *specifies* W_2 *is executed after* W_1 *completes.*
- $\#$ *denotes the exclusive choice operator.* $W_1 \# W_2$ *specifies that either* W_1 *executes or* W_2 *executes but not both.*
- $\&$ *denotes the and operator.* $W_1 \& W_2$ *specifies that both* W_1 *and* W_2 *must finish executing before the next task can start.*
- $if\{C\}W_1\ else\ W_2$ *denotes the conditioning operator.* C *is a Boolean-valued expression. Either* W_1 *or* W_2 *execute based on the result of evaluating* C *but not both.*
- $while\{C\}\{W_1\}$ *denotes iteration operator. If* C *evaluates to true,* W_1 *executes repeatedly until expression* C *evaluates to false.*

Workflows can be executed by system administrators and project participants. Examples of workflows executed by administrators include *provisionHardware*, *secureDelete*, *addUser*, *deleteUser*, *addPrivilege*, *deletePrivilege*, and *secureMigration*. Examples of workflows executed by project users are *createDataset*, *uploadDataset*, *downloadDataset*, *anonymizeDataset*, and *runSoftware*. Each workflow takes one or more parameters, and the tasks of the workflow may be executed by one or more users. Access control rules derived from the security requirements as shown in Table I determine the attributes needed to perform each task in each workflow.

Let us consider the workflow, W_1 *uploadData*, that PI executes to encrypt and upload a sensitive dataset *patientData* to *ServerD*, which is passed as parameters to *uploadData*. The informal description of the workflow is as follows:

1) If *patientData* is sensitive
   a) Automatically determine encryption algorithm based on dataset attributes (label)
   b) Generate symmetric key
   c) Encrypt dataset with key
2) Upload PatientData

Let PI from *Inst-Health* who is uploading the *PatientData* to storage *ServerD*. The workflow is described as follows:

$W_1 = if\{t_{11}\}(t_{12} \otimes t_{13} \otimes t_{14}) \otimes t_{15}$, such that

$t_{11} = check(PatientData.label == sensitive)$

$t_{12} = determineEncType(PatientData)$

$t_{13} = generateSymmetricKey(Key1)$

$t_{14} = encrypt(PatientData, Key1)$

$t_{15} = write(PatientData, ServerD)$

Let *John* is the PI of *Inst-Health* perform the workflow W_1, where *User* = *John*, *John.role* = *PI*, *John.institute* = *Inst-Health*. *Resource* = {*PatientData, ServerD*}. *PatientData.label* = *sensitive*, *PatientData.encrypted* = *no*, *PatientData.type* = *data*, and *ServerD.type* = *storage*.

In this example, we will have *refinement inconsistency* if *John* does not have access to perform a write operation on *ServerD*. In such a case, he will not be able to execute the workflow. *Compliance inconsistency* can occur if workflows or requirements change. In this example, *John*, because he is a PI, can upload sensitive data. Suppose that the RCI security requirements change such that only PI from *Univ-A* can upload sensitive data. Now, if this change is not reflected in the user attribute to execute the task, then the above workflow will no longer be compliant with the regulations and it will exhibit *compliance inconsistency*.

III. Consistency Resolution Approach

As access control policies (derived from rules and regulations) and workflows (derived from scientific collaboration) evolve independently, it is important to maintain compliance consistency between them. Figure 1 presents a flowchart illustrating the verification processes of compliance and refinement consistency. It begins with using security requirements to define the access control policies for the system operations and also formally defining the workflow model from the workflow requirements. The access control policies for the system operations are represented as an NGAC model. Access control policies derived from security requirements also dictate the NGAC model for the workflow. Each of these models, namely, NGAC Workflow Authorization Model and NGAC System Operation Authorization Model, are analyzed in isolation for consistency checks. Once the consistencies are verified, then we compare the two models and check for refinement consistencies. If any consistency check fails, the administrator is notified. Note that, this is a one-time effort. This analysis needs to be redone only when the workflow or security requirements change.

A. Workflow Authorization Matrix

The workflow authorization matrix D is a binary matrix. Each row represents a task, and each column corresponds to an attribute in the unified attribute space. Table II visualizes the workflow authorization matrix. For each cell, a value of **"1"** indicates that the attribute is required for workflow authorization, while **"0"** denotes not required. We define the workflow authorization matrix D with dimensions $n \times (i+j)$ as follows:

$$D \in \{0,1\}^{n \times (i+j)}$$

where:

- Each row D_k (for task t_k) is a binary vector of length $i+j$
- The first i entries in each row correspond to user attributes
- The next j entries correspond to resource attributes

Workflow Tasks	User Attributes				Resource Attributes			
	ua_1	ua_2	$\cdots$	ua_i	ra_1	ra_2	$\cdots$	ra_j
t_1								
t_2								
$\vdots$								
t_n								

TABLE II: Workflow Authorization Matrix

Let $D_{k,m}$ be the entry in the k-th row and m-th column of the workflow authorization D. Then:

$$D_{k,m} = \begin{cases} 1 & \text{if task } t_k \text{ requires attribute } a_m \\ 0 & \text{otherwise} \end{cases}$$

where $a_m \in \{ua_1, \ldots, ua_i, ra_1, \ldots, ra_j\}$. We define a mapping function $\mathcal{M}$ that translates elements of workflows, *tasks*, and access control policies, *attributes*, into the workflow authorization matrix D. Let $T = \{t_1, \ldots, t_n\}$ be the set of tasks in a workflow, and $UA = \{ua_1, \ldots, ua_i\}$ and $RA = \{ra_1, \ldots, ra_j\}$ be the user and resource attributes, respectively, from the access control policies. The workflow authorization mapping function is defined as:

$$\mathcal{M} : T \times (UA \cup RA) \to \{0,1\}, \quad D_{k,m} = \mathcal{M}(t_k, a_m)$$

This abstraction allows static verification to identify *compliance inconsistencies* when a workflow evolves and when a policy is updated. The verification process, illustrated in Algorithm 1, ensures that any change in tasks or access control policies does not violate the authorization semantic between these tasks and attributes.

Algorithm 1: Workflow Authorization Verifier

Input: workflow authorization matrix D, updated task list T_{new}, user attributes UA, resource attributes RA
Output: `True` if consistent; `False` otherwise

```
A ← UA ∪ RA;
for t ← 1 to |T_new| do
    for a ← 1 to |A| do
        if D[t][a] = 1 and A[a] ∉ UA ∪ RA then
            // compliance inconsistency found
            NotifyAdmin("inconsistent: task t → missing attribute A[a]") ;
            return False;
        end
    end
end
return True ;                    // workflow is consistent
```

B. Consistency Properties

We further define a set of consistency properties to ensure semantic alignment of the NGAC attributes and workflow tasks. These properties are maintained by the workflow authorization matrix D. The following consistency properties must always hold:

Property 1. *"Every task must map to a valid combination of existing attributes"*. This property guarantees that all task-to-attribute mappings in the workflow authorization

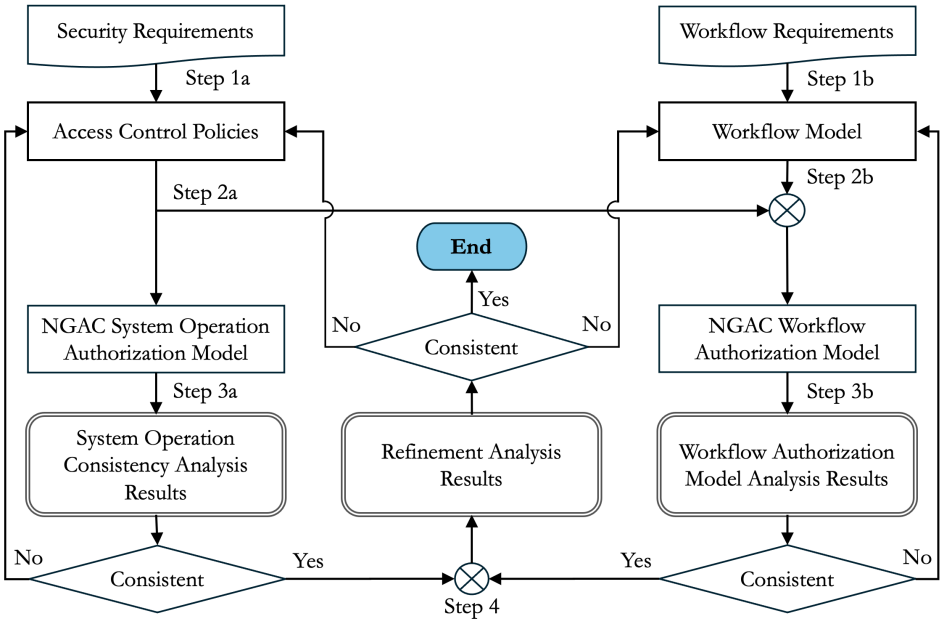

Fig. 1. Flowchart of Compliance and Refinement Checks for NGAC Model, Workflows, and System Operations

matrix remain valid under the updated NGAC model. Let $A = UA \cup RA$, with $|A| = m$, each entry $D_{k,\ell} = 1$ implies that $a_\ell \in \mathrm{A}$, i.e., the attribute must exist in the current model:

$$\forall t_k \in T,\ \forall a_\ell \in \mathrm{A},\ D_{k,\ell} = 1 \Rightarrow a_\ell \in \mathrm{A}$$

If a task references an obsolete attribute (i.e., $a_\ell \notin \mathrm{A_{new}}$), the matrix becomes inconsistent, the update is rejected, and the administrator is notified with *compliance inconsistency*.

Property 2. *"New attribute must be used by at least one task, or should be flagged as unused"*. A policy update introduces a new attribute $a_\ell \in \mathrm{A_{new}} \setminus \mathrm{A_{old}}$, where $\mathrm{A_{new}}$ is the new attribute set and $\mathrm{A_{old}}$ is the old attribute set. This new attribute must appear in at least one task row in the workflow authorization matrix:

$$\forall a_\ell \in \mathrm{A_{new}} \setminus \mathrm{A_{old}},\ \exists\, t_k \in T \text{ such that } D_{k,\ell} = 1$$

If no such task exists, $\forall\, t_k \in T,\ D_{k,\ell} = 0$, then the attribute a_ℓ is flagged as *unused*, and the administrator is notified with *compliance inconsistency*.

Property 3. *"No dangling tasks (i.e., every task must have at least one required attribute)"*. For a task, if all attributes are removed due to an administrative update, this task becomes dangling. Thus, for every task $t_k \in T$ the following must hold:

$$\sum_{\ell=1}^{m} D_{k,\ell} > 0$$

If it equals zero, then t_k is considered a dangling task and is flagged and notified as *compliance inconsistency*.

Algorithm 2 verifies these properties and ensures that the validity of the attributes, the usage of the attributes, and the compliance of tasks are enforced across the NGAC model and workflows.

Algorithm 2: Consistency Properties Verifier

Input: Updated workflow task list T_{new}; updated attribute sets UA, RA; system operations Op; dependency matrix D; conflict matrix C
Output: `True` if properties hold, `False` otherwise

$A \leftarrow UA \cup RA$; // unified attribute set
foreach $t_k \in T_{new}$ **do**
  foreach $a_\ell \in A$ **do**
    if $D[k][\ell] = 1$ ***and*** $a_\ell \notin A$ **then**
      // compliance inconsistency found
      NotifyAdmin(*"Task t_k references undefined attribute a_ℓ"*) ;
      return `False` ; // property 1 violation
    end
  end
end
foreach $a_\ell \in A_{new} \setminus A_{old}$ **do**
  if $\forall t_k \in T_{new},\ D[k][\ell] = 0$ **then**
    // compliance inconsistency found
    NotifyAdmin(*"Attribute a_ℓ is unused in tasks"*) ;
    return `False` ; // property 2 violation
  end
end
foreach $t_k \in T_{new}$ **do**
  if $\sum_{\ell=1}^{|A|} D[k][\ell] = 0$ **then**
    // compliance inconsistency found
    NotifyAdmin(*"Task t_k is dangling (no attribute dependency)"*) ;
    return `False` ; // property 3 violation
  end
end
return `True` ; // all properties satisfied

C. System Operation Authorization Matrix

System operations refer to the executable actions carried out by infrastructure services. Examples of system operations include `mount`, `unmount`, `format`, `partition`, `backup`, `restore`, `allocate`, `reboot`, `shutdown`, `write`, `read`, `delete`, `execute`, `rename`, `copy`, `encrypt`, and `decrypt`. Every workflow task may either implicitly or explicitly invoke one or many system operations. However, the authorization to invoke system operations must be restricted to specific roles. For instance, regular users, such as researchers, are permitted to access `ServerD` and `delete` a file, but they do not have permission to perform a restricted operation on `ServerD`, such as `format` and `partition`. The user attributes set UA captures these role restrictions. Let $a_k \in \{ua_1, \ldots, ua_i, ra_1, \ldots, ra_j\}$ be the user and resource attributes and let $Op = \{op_1, op_2, \ldots, op_m\}$ be the set of system operations. The binary system operation authorization matrix C defines the relationship between system operations and the user attributes required to perform them.

$$C \in \{0,1\}^{m \times (i+j)}$$

where the first i columns represent user attributes from UA, the next j columns represent resource attributes from RA, and the m rows correspond to the system operations. Each entry $C_{k,\ell}$ is defined as:

$$C_{k,\ell} = \begin{cases} 1 & \text{if operation } op_k \text{ requires attribute } a_\ell, \\ 0 & \text{otherwise.} \end{cases}$$

The system operation authorization mapping function is defined as:

$$\Phi : Op \times (UA \cup RA) \to \{0,1\}, \quad C_{k,\ell} = \Phi(a_\ell, op_k)$$

Algorithm 3: System operation Authorization Verifier

Input: system operation matrix C, operations Op, user attributes UA, resource attributes RA
Output: `True` if consistent; `False` otherwise

```
A ← UA ∪ RA;
for op ← 1 to |Op| do
    for a ← 1 to |A| do
        if C[op][a] = 1 and A[a] ∉ UA ∪ RA then
            // refinement inconsistency found
            NotifyAdmin("inconsistent: task op → missing attribute A[a]") ;
            return False ; // attribute is undefined
        end
    end
end
return True ;  // system operations are consistent
```

Each system operation should also be authorized. If an attribute is requested for performing system operation, but is not an existing attribute of an user or resource, this check fails. Algorithm 3 ensures that any changes to user and resource attributes will prevent the execution of unauthorized system operation. This algorithm identifies *compliance inconsistencies* by scanning the system operation authorization matrix.

D. Translational Logic

We have previously mapped NGAC attributes to workflow tasks and, separately, mapped NGAC attributes to system operations. We now introduce a translational logic that collectively maps NGAC attributes to workflow tasks and system operations. In other words, this translational logic enforces access control by using user attributes and resource attributes assigned to workflow tasks, determining which system operations can be executed. For example, in the `UploadData` workflow, shown in section II-C, task t_{15} = `write(PatientData, ServerD)` semantically maps to the system operation op_1 = `write` $\in$ `Op`. To enforce access control on this task, user attributes and resource attributes are verified as:

`(user.role = PI)` $\wedge$ `(PatientData.label = sensitive)` $\wedge$ `(PatientData.own = Inst-Health)` $\wedge$ `(PatientData.type = storage)` $\wedge$ `(PatientData.id = ServerD)` $\Rightarrow$ `allow(write, PatientData, ServerD).`

This translational logic ensures that task t_{15} is authorized to invoke the `write` operation, along with the access control conditions derived from the NGAC attributes, which authorize this system operation. If any of these conditions are violated, e.g., the user is not a PI, a *refinement inconsistency* is flagged and the task is declined.

To ensure translational logic consistency, we reuse the matrix D for workflow authorization and the matrix C for system operation authorization. We define another matrix, a refinement matrix R, that determines what system operations each task is allowed to invoke. The refinement matrix is defined as $R \in \{0,1\}^{n \times p}$, where n is the number of workflow tasks, $t_i \in T$, and p is the number of system operations, $\text{op}_x^\ell \in Op$. The refinement mapping function is:

$$\Theta : T \times \text{Op} \to \{0,1\}, \quad R[i,\ell] = \Theta(t_i, \text{op}_x^\ell)$$

where:

$$\Theta(t_i, \text{op}_x^\ell) = \begin{cases} 1 & \text{if } t_i \text{ is allowed to invoke operation } \text{op}_x^\ell \\ 0 & \text{otherwise} \end{cases}$$

These three matrices, D, C, and R, are collectively used to detect *refinement inconsistencies* between NGAC attributes, workflow tasks, and system operations. For instance, for a task t_i, if $R[t_i, \text{op}_x^\ell] = 1$, t_i is allowed to trigger operation op_x^ℓ, and $C[\text{op}_x^\ell, a_k] = 1$, op_x^ℓ requires attribute a_k, but if $D[t_i, a_k] = 0$, the task t_i does not require attribute a_k, then a *refinement inconsistency* is flagged.

Algorithm 4 iterates through the matrices D, C, and R to enforce the translational logic consistency. It ensures that, for every system operation invoked by a workflow task, the task declares the required attributes necessary to authorize that operation.

IV. Approach Demonstration: RCI Usecases

This section presents usecases from a real-world RCI environment to validate the effectiveness of matrix-based consistency verification processes. These usecases are simplified

Algorithm 4: Translational Logic Verifier

Input: attribute set A, workflow task set T, operation set Op;
Workflow authorization matrix $D \in \{0,1\}^{n \times m}$;
Operations authorization matrix $C \in \{0,1\}^{p \times m}$;
Refinement matrix $R \in \{0,1\}^{n \times p}$
Output: `True` if all attribute-task-operation mappings are satisfied; `False` if inconsistency found

```
foreach t_i ∈ T do
    foreach op_j ∈ Op do
        if R[t_i, op_j] = 1 then
            foreach a_k ∈ A do
                if C[op_j, a_k] = 1 and D[t_i, a_k] = 0 then
                    // refinement inconsistency found
                    print "conflict: task t_i triggers op_j requiring a_k, but lacks this attribute" ;
                    return False
                end
            end
        end
    end
end
return True   // translational logic is consistent
```

to reflect typical access control, workflow, and operational inconsistency often encountered in practice. Each usecase illustrates how *compliance inconsistency* or *refinement inconsistency* arises, how it is caught, and how it is resolved. The first usecase addresses a *compliance inconsistency* induced by a workflow that violates existing policy constraints due to a missing security requirement. The second usecase demonstrates a *refinement inconsistency*, where a user is assigned to a task that attempts to invoke a restricted system operation. The third usecase illustrates a more complex scenario of *refinement inconsistency*, where multiple tasks within a workflow are assigned to different users with distinct roles to perform different system operations. This requires translational logic requirements to be held in order for this workflow to be valid.

A. Compliance Inconsistency Usecase: Authorize workflows

In a collaborative scientific environment, the workflow W_2 $ML-Secure$ is designed to support the development of Machine Learning (ML) models using sensitive healthcare data. $W_2 = t_{21} \otimes t_{22} \otimes t_{23} \otimes t_{24} \otimes t_{25}$ where
$t_{21} = download(PatientData, ServerD)$,
$t_{22} = decrypt(PatientData, Key1)$,
$t_{23} = assignLabel(PatientData)$,
$t_{24} = trainModel(PatientData)$,
$t_{25} = evaluateModel(Model1)$.
The user attributes required to perform workflow W_2 are *user.institute = Univ-A*, *user.role = student*, *user.role = researcher*, and *user.role = PI*, and the resource attributes needed are *PatientData.own = Inst-Health*, *PatientData.label = sensitive*, *PatientData.enc = yes*, *PatientData.type = storage*, and *PatientData.id = ServerD*. The dataset $PatientData$ is encrypted and uploaded to *ServerD* by *Inst-Health*, as detailed in section II-C. Subsequently, a research team from *Univ-A* seeks to develop an ML model by downloading, decrypting, and labeling this dataset. They then train and evaluate the ML model using the labeled dataset. The workflow W_2 includes 5 tasks, assuming that a researcher can download, decrypt, and label the $PatientData$ dataset. The model training can be assigned to a student and then the researcher can evaluate the model. The workflow authorization matrix D of size 5×21 (5 tasks $\times$ 8 user attributes + 13 resource attributes) is initialized. Because of the page size limits, Table III presents only a subset of 9 attributes sufficient to demonstrate this usecase.

According to the enforcement policies P1 and P2, only PIs are allowed to download and decrypt sensitive data. The ML model evaluation also requires PI due to the sensitivity of the evaluation results, which are classified as writing sensitive data. The workflow authorization verifier, Algorithms 1, validates each task-to-attribute mapping to ensure compliance with access control policies. For tasks t_{21}, t_{22}, and t_{25}, the matrix entry $D[t_k, \texttt{role=PI}] = 1$, shown in Table III in bold, indicates that these tasks require the `role` attribute set as PI. However, the workflow $W2$ violates this security requirement because the `role` attribute for tasks t_{21}, t_{22}, and t_{25} was incorrectly assigned as `researcher`. Thus, the verifier identifies this *compliance inconsistency*, notifies the administrator and returns `False`. Consequently, adding this workflow to the database is declined.

Usecase Resolution: To ensure compliance with security requirements, the workflow $W2$ is modified and restricted with roles. The workflow authorization matrix correctly maps tasks t_{21}, t_{22}, and t_{25} to the `role` attribute which in this case is `role = PI`. As a result, the workflow authorization verifier no longer flags any *compliance inconsistencies* for the W_2 $ML-Secure$ workflow because all workflow tasks are compliant with the security requirements.

B. Refinement Inconsistency Usecase: Authorize System Operations

To facilitate efficient training for the ML model, the workflow W_3 $ML-Training$, is designed to allow the research team of *Univ-A* to train and evaluate the ML model on an HPC, for instance, a powerful computational node with GPU referred to as GPU_x1. This workflow includes provisioning the HPC node, uploading the dataset, and then training and evaluating the ML model. $W_3 = t_{31} \otimes t_{32} \otimes t_{33} \otimes t_{34}$ where
$t_{31} = provision(GPU\_x1)$,
$t_{32} = upload(PatientData, GPU\_x1)$,
$t_{33} = trainModel(PatientData, GPU\_x1)$,
$t_{34} = evaluateModel(Model1, GPU\_x1)$.
Task t_{31}, provisioning an HPC, requires system operations such as `allocate` HPC node and `mount` a file system. Once task t_{31} is completed, the team proceeds to upload the dataset, task t_{32}, to the HPC node, which involves system operation `write`. Following this, they train the ML model on the HPC, task t_{33}, which requires system operation `read`, and evaluate the trained ML model, task t_{34}, that involves system operation `write`. The workflow authorization matrix has the following `role` attribute for the tasks $t_{31}, t_{32}, t_{33}, t_{34}$ as `PI, PI, researcher, PI` respectively. The user attributes required

Task	User Attributes				Resource Attributes				
	inst=Univ-A	role=student	role=researcher	role=PI	own=Inst-Health	label=sensitive	enc=yes	type=storage	id=ServerD
t_{21}	1	0	0	**1**	1	1	1	1	1
t_{22}	1	0	0	**1**	1	1	1	0	0
t_{23}	1	0	1	0	1	1	0	0	0
t_{24}	1	1	0	0	1	1	0	0	0
t_{25}	1	0	0	**1**	1	1	0	0	0

TABLE III: W_2 $ML-Secure$ Workflow Authorization Matrix

System Operation	User Attributes				Resource Attributes		
	inst=Univ-A	role=researcher	role=admin	role=PI	label=sensitive	type=computation	id=GPU_x1
op_1 $(allocate)$	1	0	**1**	0	0	1	1
op_2 $(mount)$	1	0	**1**	0	0	1	1
op_3 $(write)$	1	0	0	**1**	1	1	1
op_4 $(read)$	1	1	0	0	1	1	1
op_5 $(write)$	1	0	0	**1**	1	1	1

TABLE IV: W_3 $ML-Training$ System Operation Authorization Matrix

for W_3 are *user.institute = Univ-A*, *user.role = researcher*, *user.role = admin*, and *user.role = PI*, and the resource attributes needed are *PatientData.label = sensitive*, *PatientData.type = computation*, and *PatientData.id = GPU_x1*. The system operation authorization matrix C is initialized with a size of 5×21 (5 operations $\times$ 8 user attributes $+$ 13 resource attributes). Again, because of the page size limits, Table IV includes only a subset of 7 attributes that is adequate to demonstrate this usecase.

Policy P6 requires that only a user with an administrator role is allowed to provision an HPC. The system operation authorization verifier, Algorithm 3, checks each mapping of the operation to the attribute. For operations op_1 and op_2, the matrix entry $C[op_x, \texttt{role=admin}] = 1$ indicates that these operations require that the attribute `role` be set as `admin`. However, the workflow W_3 violates this security requirement, as the task t_{31} incorrectly sets the `role` attribute to `PI`, which is inconsistent with the system operation constraints. Thus, the verifier identifies this *refinement inconsistency*, notifies the administrator, and returns `False`.

Usecase Resolution: To ensure the consistency of system operations with security requirements, the workflow authorization matrix is adjusted. Now, the system operation authorization matrix correctly maps system operations to the `role` attribute, and the user with `role = admin` satisfies the security requirements to perform system operations op_1 and op_2. We change the workflow authorization matrix for tasks t_{31}, t_{32}, t_{33}, t_{34} as `role` attribute `admin, PI, researcher, PI` respectively. Hence, the system operation authorization verifier no longer flags any *refinement inconsistencies* for W_3 workflow.

C. Translational logic Inconsistency Usecase: Deployment of the Trained ML Model

After training the ML model on the $PatientData$ dataset, the *Univ-A* research team aims to deploy the model on a newly provisioned HPC storage (named `MLhealthcare`). To facilitate this, the deployment workflow $W4$ $ML-Deployment$ is designed. This workflow includes downloading the trained model `trainedModel` from GPU_x1, provisioning new storage `MLhealthcare`, uploading the model to `MLhealthcare`, connecting a client interface to `MLhealthcare`, and running `trainedModel`.
$W_4 = t_{41} \otimes t_{42} \otimes t_{43} \otimes t_{44} \otimes t_{45}$ where
$t_{41} = download(PI, Model1, GPU\_x1)$,
$t_{42} = provision(admin, MLhealthcare)$,
$t_{43} = upload(PI, Model1, MLhealthcare)$,
$t_{44} = connect(researcher, interface, MLhealthcare)$,
$t_{45} = runModel(researcher, Model1)$.
where the workflow authorization matrix t_{41}, t_{42}, t_{43}, t_{44}, t_{45} have `role` attribute `PI, admin, PI, researcher, researcher` respectively in the authorization matrix. As mentioned, a *refinement inconsistency* occurs when task t_k is assigned attributes that authorize it, which are verified through matrix D, and the required attributes for an operation op_x are valid through matrix C. If the refinement matrix $R[t_k, op_x] = 1$ does not semantically map task t_k to operation op_x, then op_x should not be invoked. The refinement matrix R is initialized, as shown in Table V, with a size of 5×6 (5 tasks $\times$ 6 operations).

Task	System Operations					
	downloadModel	provision		uploadModel	connect	runModel
	p_1	p_2	p_3	p_4	p_5	p_6
	$write$	$allocate, mount$		$write$	$mount$	$read$
t_{41}	1	0	0	0	0	0
t_{42}	0	1	1	0	0	0
t_{43}	0	0	0	1	0	0
t_{44}	0	0	0	0	**1**	0
t_{45}	0	0	0	0	0	1

TABLE V: Refinement Matrix for W_4 Workflow

In this usecase, the matrices D and C are valid, but the matrix R is not valid. The translational logic verifier, Algorithm 4, identifies a *refinement inconsistency* in the refinement

matrix R, that the `connect` operation requires mounting the client interface to the `MLhealthcare` node and the `mount` operation requires an administrative role. The translational logic for t_{44} is as follows:

`(user.inst = Univ-A) ∧ (user.role = admin) ∧ (MLhealthcare.own = Univ-A) ∧ (MLhealthcare.type = storage) ∧ (MLhealthcare.id = MLhealthcare) ⇒ allow(mount, interface, MLhealthcare).`

Thus, the verifier flags the workflow W_4 with an *refinement inconsistency*, notifies the administrator and returns `False`.

Usecase Resolution: To ensure the transitional logic consistency, the W_4 workflow authorization matrix is modified. Now the tasks t_{41}, t_{42}, t_{43}, t_{44}, and t_{45} have `role` attribute set to `PI, admin, PI, admin, researcher` respectively. The user with `role = admin` is authorized to perform task t_{44} that performs the `connect` operation, which involves the system operation `mount`. As a result, the three matrices, D, C and R, correctly map attributes, workflow tasks, and system operations. Consequently, the transitional logic consistency verifier no longer flags any *refinement inconsistencies* for the W_4 workflow.

In these usecases, matrix-based abstractions effectively identify and resolve *compliance* and *refinement* inconsistencies between security requirements, workflows, and system operations. The verification processes detect violations of unauthorized tasks, unauthorized operations, or mismatched translational logic. This allows policy and workflow administrators to make minimal adjustments for semantic policy-compliant workflows within dynamic RCI environments.

V. Discussion

Matrix-based abstractions have proven to be effective and efficient tools for reasoning about the consistency of security requirements, workflows, and system operations within the RCI environment. These abstractions, workflow authorization matrix and system operation authorization matrix, provide a formal yet lightweight mechanism as they use binary matrices and mapping functions. These matrices provide early detection of *compliance* and *refinement* inconsistencies that may occur between security requirements, workflows, and system operations. A set of algorithms, workflow authorization verifier, system operation authorization verifier, and translational logic verifier, use these matrices to collectively ensure consistency of security requirements, workflows, and system operations. The matrix-abstraction approach has further advantages. It identifies the obsolete attributes, those attributes that do not depend on tasks. It also identifies the dangling tasks, those tasks without attributes. The consistency properties verifier captures these obsolete attributes and dangling tasks, allowing the administrator to be notified of their removal. These verification processes execute in tandem, with one triggering the others, to ensure consistency between security requirements, workflow tasks, and system operations is always maintained.

The workflow authorization consistency verifier, Algorithm 1, runs in $\mathcal{O}(n \cdot m)$ time, where n is the number of workflow tasks and m is the total number of user and resource attributes. This linear-time complexity makes the algorithm practical to apply to a real-time RCI environment. The consistency properties verifier, Algorithm 2, verifies three properties separately. For property 1, the algorithm iterates over all n tasks and all m attributes. For property 2, it iterates over r new attributes and checks all n tasks. For property 3, it iterates over all n tasks and the sum of rows of attributes. Thus, Algorithm 2 runs in $\mathcal{O}(n + m + p)$ time. The system operation authorization verifier, Algorithm 3, runs in $\mathcal{O}(n \cdot m')$ time, where m' is the number of system operations. Since the number of system operations is finite and typically much smaller than the workflow tasks space, this algorithm is also computationally efficient. The translational logic verifier, Algorithm 4, has nested iteration over *task-operation* matrix. Thus, it runs in $\mathcal{O}(n \cdot p)$ time, where n is the number of tasks and p is the number of system operations.

Although matrix-based abstractions are limited to static access control policies, the underlying NGAC framework supports dynamic policy management through "policy obligations". We thus plan to explore the integration of NGAC policy obligations into matrix representations, where the NGAC model dynamically changes based on spatial and temporal constraints, along with multi-level sensitivity constraints. This requires enhancement to our model by including multi-valued attributes, obligation spatial and temporal guards, and constraint columns to the D, C, and R matrices. This integration will allow us to extend consistency verification beyond binary mappings to include conditional constraints as well as spatial and temporal properties.

Overall, abstraction matrices and consistency verification processes provide an efficient policy-compliant enforcement mechanism for collaborative workflows in RCI. Our ongoing research aims to validate the practical effectiveness, scalability, and integration overhead of these verification processes in a real-world RCI environment.

Our approach can also be extended to include more policy constraints, such as Separation of Duty (SoD) and policy binding constraints. To implement these extensions, we need to add an additional column in the dependency matrices to explicitly represent user assignments and define a new property to detect conflicts that arise from these constraints. Such enhancements would allow the framework to capture a broader range of policy semantics and to identify violations beyond attribute and system operation-based inconsistencies. Another potential extension is to incorporate multiple levels of abstraction in the system operations. While the current framework considers two levels, real-world systems may involve multi-level execution in which a high-level operation invokes intermediate methods, which in turn call lower-level methods, finally leading to system operations. Authorization checks should be enforced consistently at each level of this execution hierarchy to ensure end-to-end policy compliance. These extensions are considered promising directions for future work.

VI. Related Work

Access control in collaborative and distributed environments has been widely studied [5,8].

Uddin *et al.* [9] propose the AW-TRBAC model that incorporates authorization workflows and task-role-based access control. AW-TRBAC model supports dynamic segregation of duties (SoD), task-instance level restrictions, and real-time decision making using XACML extensions. It incorporates custom functions and data stores to support fine-grained access control policies. The authors have implemented and demonstrated the applicability of AW-TRBAC to a real-world financial institution usecase. They focus on achieving scalability and responsiveness without compromising performance.

Esterhuyse *et al.* [10] present real-time, privacy-preserving enforcement for distributed scientific workflows, particularly in medical research settings. The authors introduce a mechanism called "checkers" that evaluates access and usage policies (read, write, and compute) specified in eFLINT, which utilizes private and cooperative modes for selective disclosure, balancing privacy and efficiency. Demonstrations on medical data workflows highlight the value of embedding formal policy reasoning into collaborative workflow execution environments.

Liu *et al.* [11] address the limitations of traditional RBAC, ABAC, and task-based models in dynamic workflow environments. They introduce a Task-Attribute-Based Access Control (T-ABAC) model. This model establishes fine-grained access control through dynamic mappings between user attributes, task attributes, and task privileges. The authors separate the management of accounts, attributes, and privileges, to support strict verification and enforce static and dynamic Separation of Duties (SoD). They formalized the model using relational matrices that express task-attribute, task-status, and status-privilege relationships. This formalization provides real-time authorization and privilege revocation based on the workflow context. The authors demonstrated that this framework improves precision, reduces administrative overhead, and adapts more effectively to the dynamic nature of collaborative workflow systems.

Access control for collaborative scientific workflows needs investigation for both computational and experimental sciences, as it is essential for protecting intellectual property and managing the flow of sensitive data [12,13].

Yong Zhao *et al.* [14] propose a cloud-enabled framework for scalable scientific workflow management. The authors address dynamic provisioning, fault tolerance, and heterogeneity across OpenNebula, Eucalyptus, and Amazon EC2. Experiments with real workflows (MODIS, Montage) demonstrate feasibility of running large-scale scientific applications in the cloud. The authors highlighted that such integration improves responsiveness, scalability, and cost effectiveness, which are essential for collaborative scientific research.

Greene *et al.* [15] present the NIST Open Access to Research (OAR) data infrastructure. It is a scalable, standards-compliant, and FAIR-aligned platform comprising the NERDm portal and a public repository that allows interoperable access across domains such as chemistry, materials science, and metrology. The framework demonstrates how federal research institutions implement open science mandates through sustainable and interoperable architectures. It improves the discoverability, quality, and usability of NIST datasets while complying with government open data policies.

NIST Next Generation Access Control (NGAC) [3] is a rich, flexible, and dynamic framework for specifying attribute-based access policies, making it a strong candidate for securing scientific workflows and the infrastructure on which they are executed. We demonstrate how NGAC policies that protect the workflow and system operations can be analyzed. To the best of our knowledge, we are unaware of any other work that tries to ensure that the access control of high-level workflow tasks maps to the access control of the system operations executing those tasks. This is crucial to ensure that the research infrastructure is not compromised due to some malicious task in a scentific workflow.

VII. Conclusion

In a scientific collaborative environment, where research security is critically important, access control plays a crucial role. Towards this end, we propose the use of the NIST NGAC for securing a scientific workflow involving dynamic set of users belonging to multiple institutions, possibly from different countries. The workflow tasks are implemented by system level operations. Consequently, there is a possibility of compromise through a malicious task that may possibly embed a malware on a secure resource. To address this, we propose approaches that not only verify the correctness of workflow access control policies and system level operations in isolation (through consistency checks) but also ensures that the mapping of workflow tasks to the system operations are also done in a secure manner (through refinement checks). We also presented some use cases to illustrate our approach.

We are currently implementing our approach within a real-world research computing infrastructure. Future work will provide experimental evaluation using multi-institutional collaborative projects. Future work also involves a more detailed model for the workflow that takes into account user-level constraints, such as binding, separation, and cardinality constraints, and also incorporates obligation policies through which access control configuration automatically changes. Another future work involves how detailed access control logs generated by various applications and systems can be collated and the access pattern determined to refine the access control policies, both at the workflow and at the system operation level. Once we have a more refined model, we plan to implement our approach for real-world use cases.

Acknowledgement

This work was supported in part by funding from NIST under Grant Number 60NANB23D152 NSF under Award Numbers CNS 2335687, AMI, and State of Colorado SB 18-086 funding.

REFERENCES

[1] R. Rybnicek and R. Königsgruber, "What makes industry–university collaboration succeed? a systematic review of the literature," *Journal of business economics*, vol. 89, no. 2, pp. 221–250, 2019.

[2] W. Penuel, R. Riedy, M. Barber, D. Peurach, W. LeBouef, and T. Clark, "Principles of collaborative education research with stakeholders: Toward requirements for a new research and development infrastructure," *Review of Educational Research*, vol. 90, no. 5, pp. 627–674, 2020.

[3] A. N. S. Institute, "Information Technology - Next Generation Access Control (NGAC)," ANSI, New York, NY, Information Technology, 04 2020, accessed: April 2025.

[4] V. Atluri and W. Huang, "An authorization model for workflows," in *Proceedings of the 4th European Symposium on Research in Computer Security (ESORICS)*. Springer, 1996, pp. 44–64.

[5] M. Alam, M. Shahid, and S. Mustajab, "Security challenges for workflow allocation model in cloud computing environment: a comprehensive survey, framework, taxonomy, open issues, and future directions," *The Journal of Supercomputing*, vol. 80, no. 8, pp. 11 491–11 555, 2024.

[6] M. Abdelgawad, I. Ray, S. Alqurashi, V. Venkatesha, and H. Shirazi, "Synthesizing and analyzing attribute-based access control model generated from natural language policy statements," in *Proceedings of the 28th ACM Symposium on Access Control Models and Technologies*, 2023, pp. 91–98.

[7] M. Abdelgawad, I. Ray, and T. Vasquez, "Workflow resilience for mission-critical systems," in *Proceedings of 25th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*. Springer, 2023, p. 498–512.

[8] F. Paci, A. Squicciarini, and N. Zannone, "Survey on access control for community-centered collaborative systems," *ACM Computing Surveys (CSUR)*, vol. 51, no. 1, pp. 1–38, 2018.

[9] M. Uddin, S. Islam, and A. Al-Nemrat, "A dynamic access control model using authorising workflow and task-role-based access control," *IEEE Access*, vol. 7, pp. 166 676–166 689, 2019.

[10] C. Esterhuyse, T. Müller, T. Van Binsbergen, and A. Belloum, "Exploring the enforcement of private, dynamic policies on medical workflow execution," in *Proceedings of the IEEE 18th International Conference on e-Science (e-Science)*. IEEE, 2022, pp. 481–486.

[11] Y. Liu, K. Xu, and J. Song, "A task-attribute-based workflow access control model," in *Proceedings of the IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing (GreenCom-iThings-CPSCom)*. IEEE, 2013, pp. 1330–1334.

[12] E. Deelman, T. Peterka, I. Altintas, C. Carothers, K. Van Dam, K. Moreland, M. Parashar, L. Ramakrishnan, M. Taufer, and J. Vetter, "The future of scientific workflows," *The International Journal of High Performance Computing Applications*, vol. 32, no. 1, pp. 159–175, 2018.

[13] M. Atkinson, S. Gesing, J. Montagnat, and I. Taylor, "Scientific workflows: Past, present and future," *Future Generation Computer Systems*, vol. 75, pp. 216–227, 2017.

[14] Y. Zhao, Y. Li, I. Raicu, S. Lu, W. Tian, and H. Liu, "Enabling scalable scientific workflow management in the cloud," *Future Generation Computer Systems*, vol. 46, pp. 3–16, 2015.

[15] G. Greene, R. Plante, and R. Hanisch, "Building open access to research (OAR) data infrastructure at NIST," *Data science journal*, vol. 18, pp. 10–5334, 2019.