THESIS

INTEGRATION OF TASK-ATTRIBUTE BASED ACCESS CONTROL MODEL FOR MOBILE WORKFLOW AUTHORIZATION AND MANAGEMENT

Submitted by

Rejina Basnet

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Spring 2019

Master's Committee:

Advisor: Dr. Indrakshi Ray
Co-Advisor: Dr. Ramadan Abdunabi

Dr. Indrajit Ray
Dr. Leo R. Vijayasarathy

ABSTRACT

INTEGRATION OF TASK-ATTRIBUTE BASED ACCESS CONTROL MODEL FOR MOBILE WORKFLOW AUTHORIZATION AND MANAGEMENT

Workflow is the automation of process logistics for managing simple everyday to complex multi-user tasks. By defining a workflow with proper constraints, an organization can improve its efficiency, responsiveness, profitability, and security. In addition, mobile technology and cloud computing has enabled wireless data transmission, receipt and allows the workflows to be executed at any time and from any place. At the same time, security concerns arise because unauthorized users may get access to sensitive data or services from lost or stolen nomadic devices. Additionally, some tasks and information associated are location and time sensitive in nature. These security and usability challenges demand the employment of access control in a mobile workflow system to express fine-grained authorization rules for actors to perform tasks on-site and at certain time intervals. For example, if an individual is assigned a task to survey certain location, it is crucial that the individual is present in the very location while entering the data and all the data entered remotely is safe and secure.

In this work, we formally defined an authorization model for mobile workflows. The authorization model was based on the NIST(Next Generation Access Control) where user attributes, resources attributes, and environment attributes decide who has access to what resources. In our model, we introduced the concept of spatio temporal zone attribute that captures the time and location as to when and where tasks could be executed. The model also captured the relationships between the various components and identified how they were dependent on time and location. It captured separation of duty constraints that prevented an authorized user from executing conflicting tasks and dependency of task constraints which imposed further restrictions on who could execute the tasks. The model was dynamic and allowed the access control configuration to change through

obligations. The model had various constraints that may conflict with each other or introduce inconsistencies. Towards this end, we simulated the model using Timed Colored Petri Nets (TCPN) and ran queries to ensure the integrity of the model. The access control information was stored in the Neo4j graph database. We demonstrated the feasibility and usefulness of this method through performance analysis. Overall, we tended to explore and verify the necessity of access control for security as well as management of workflows. This work resulted in the development of secure, accountable, transparent, efficient, and usable workflows that could be deployed by enterprises.

ACKNOWLEDGEMENTS

First of all, I would like to express my sincere gratitude to my advisors, Dr. Indrakshi Ray and Dr. Ramadan Abdunabi, for their constant and long-enduring support. I have learned a great deal from them through their guidance and supervision. Their immense knowledge in the field has been of great help during this thesis work.

I want to extend my special thanks to my thesis committee members, Dr. Indrajit Ray and Dr. Leo R. Vijayasarathy, for providing me with their insightful comments and encouragement throughout my research work. Furthermore, I would also like to express my sincere thanks to all members of my research lab for their collaboration and continual assistance.

All my friends at CSU have been an indispensable part of my academic life as well as my personal life. Especially, Wendy Stevenson, Administrative Assistant for the Department of Computer Science, who has supported me with her wise opinions and assistance during different events throughout the process. I heartily thank them for all their ever-lasting love and care.

Last but not least, I would like to thank my family and friends from Nepal, for having faith in me and supporting my higher studies despite all the odds. I would not have become who I am today without their love and encouragement.

DEDICATION

To my parents

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Work Flow Management is a fast-evolving technology which is increasingly being exploited by businesses in a variety of industries. Its primary characteristic is the automation of processes involving combinations of human and machine-based activities, particularly those involving interaction with IT applications and tools [1]. A workflow consists of a set of partially ordered tasks designed to achieve a goal. Tasks represent the logical unit of work that is executed in a workflow via one or more authorized users. A participant (a human being or machine agent) assigned to execute a task requires access to system resources. Workflow provides the required resources on required time to proper participants through a set of authorization rules that govern the relationship between users, tasks, and resources. A workflow management system defines, manages and executes "workflow" through the execution of software whose order of execution is driven by a computer representation of the workflow logic [1].

The growth of mobile technology has benefited numerous workflow domains including e-commerce, electronic government, health care, and power control systems. For example, mobile health care applications [2] can detect and alert medical professionals of a patient's fall anytime and anywhere. Also, iMedik [3] is a mobile telemedicine application accessible by handheld devices that are integrated with Global Positioning System (GPS).

Through proper specification and execution of a workflow, business processes can improve their productivity and accountability. The addition of mobile technologies makes it more flexible and reachable. Therefore, the mobile workflow along with a proper management system aids the development of the organization.

1.1 Underlying Problem

To ensure proper execution of the workflow task, the dependencies between the task needs to be addressed. The task dependencies can be static as well as dynamic. Dynamic dependencies

might occur due to certain chances after the workflow has been executed. Also, there can be constraints related to a specific task itself. Considering mobile workflow, some task might need to be performed on a certain location and time. The mobility also requires the availability of the information; therefore stored in the ubiquitous cloud. This creates security and usability challenges. The valuable information is stored as well as risk for breaches. Intruders might get access to the system through stolen or lost nomadic devices causing controversial updates: infesting deadlocks, obstructing time-sensitive task, jeopardizing the schedule, and so on.

We believed that the authorization rules could be utilized to control the execution of tasks while ensuring task execution by legitimate actors and, therefore, the data security, efficiency, and productivity in a workflow would be preserved. To prevent fraudulent actions, rules must be expressed for the control flow, assignment, execution, and separation of the duties of tasks. Previously Role-Based Access control (RBAC) model [4] was used to formally define and implement the authorization rules of a workflow. It controls access based on the roles that users have within the system and on rules stating what access was allowed for what users in given roles. The concept of roles is useful in reducing administrative work and maximizing operational efficiency. Furthermore, researchers have advocated RBAC with spatial and temporal constraints. To the best of our knowledge, the most known and detailed spatio-temporal RBAC extensions are in [5–9]. However, the RBAC model is not appropriate for complex workflows applications involving inter industries (have different role structure) communication.

Existing works have authorized access based on the present environmental conditions related to a user but did not provide mechanisms for persistent spatio-temporal control enforcement after resources have been accessed. Neither did they consider the impact of spatio-temporal constraints on user-role assignment or permission-role assignment [4]. In situations where organizations have a huge set of roles, RBAC could encounter the role explosion problem which could be a major drawback of the RBAC model [10].

1.2 Our Approach

To avoid RBAC's "role explosion" and to provide higher agility, Attribute-Based Access Control (ABAC) model was introduced [11]. In this model, access could be determined based on various attributes presented by a subject. The ABAC controls access based on three different attribute types: user attributes, attributes associated with the application or system to be accessed, and current environmental conditions. Also, ABAC model provides dynamic, fine-grained and context-aware access control to resources allowing access control policies that include specific attributes from many different information systems to be defined to resolve authorization and achieve efficient regulatory compliance, allowing enterprises flexibility in their implementations based on their existing infrastructures [12].

In this work, we tended to merge the essential aspects from both RBAC and ABAC to extend the earlier work on workflows along several dimensions. Firstly, we provided a multi-modular architecture that represented and executed ordered tasks to fulfill the business needs. It improved the adaptability and helped in risk management. Not only did it enforced the process logistics but enforces the access control, control flow, and the constraints. Information required was shared between the modules among which the authoritative decisions were made by an access control module based on our proposed dynamic, spatio-temporal access control model introduced as Task-Attribute Based Access Control (T-ABAC).

In our model, we introduced the concept of spatio-temporal zone attribute that captured the time and location as to when and where tasks could be executed. Abstracting location and time into a single STZone also reduced the number of entities in the model, making it easier to understand, manage, and verify. The ability of the model to represent the real-life relationship between the entities help to address all the major components required for the workflows. It explicitly modeled the resources and, therefore, handled the workflow situation where access to the object may be contingent on its location as mentioned in [13].

The access-control components were entered in a formal verification tool for the automaton checkers to detect any contradiction of policy rules or incompleteness. Researchers have proposed

automated analysis approaches for existing formal method notations and tools. Examples included UML/OCL [14, 15], Alloy [16, 17], and Colored Petri Nets [18, 19]. Due to the presence of hard real-time properties, we simulated the model using Time Colored Petri Nets(TCPN).

Finally, we introduced a platform-independent architecture model for designing mobile workflow applications enforcing an access control policy from an abstract perspective to the context of the real-life application. For the analysis, the policies were represented as the authorization graph using the policy machine constructs and enforced with Neo4j (an open source graph database system) and extended by the inclusion of a graph algorithm to traverse the authorization graph. We analyzed the efficacy of the algorithm in terms of the time required to answer the queries. In addition, we also analyzed the correctness of the authorization graph or policy representation. Hence, the analysis helped us explore the usability of our implementation in real-time systems and helped discover the factors to increase efficiency and performance.

1.3 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 presents a few previous works and a brief comparison with our work. Chapter 3 presents the real-world mobile workflow which served as problem reference that was being solved through this work. In Chapter 4 we show an architecture to create and enforce a well-managed workflow system. It is followed by Chapter 5 where we describe our location and time model and further continue with Task-Attribute Based Access Control Model in Chapter 6. We introduce all the components, their relationships, and mathematical derivation of access in this chapter. Next, we talk about the model checking, its challenges and our approaches to overcome it in section 7. Finally, we describe the enforcement architecture and the performance of the enforced system in Chapter 8. We conclude with Chapter 9 by briefing our work and findings. We also point towards the extension of our work that we are looking forward to.

Chapter 2

Literature Review

The contribution of this work was to provide a complete framework that dealt not only with the access control but also the management and processing of a workflow. There was no prior work addressing all the problems, therefore, in this section, we consider the similar problems like information systems (including workflows) and their management, access control, and integration of access control into real-world systems.

2.1 Information System and Management

Current work of Christopher Yang et al. [20] on health care informatics envisioned that the integration of different disciplines to healthcare would advance the services and well-being of our society would accelerate. They further stated that the research activities in this sector of interdisciplinary was broadly classified into three major tracks, (i) system, (ii) analytics, and (iii) human factors. In sum, Christopher Yang et al. indicated challenges in architecture, management, and security which held for another information system as well. Shortly after, Suruchi Deodhar et al. [21] implemented integrated, high-performance computing oriented epidemic modeling environment named DISimS (Distributed Interactive Simulation System) which focused on improving user productivity and ease of use, supporting interactive modeling and pervasive web-based access to the models. The work showed the importance of integration of computing and information system to solve real-life scenarios. It also highlighted the perks of the ubiquitous availability of the information. In contrast to these works, we focused on management and security of the whole process such that we could enjoy all these services and yet preserve the information from breaches.

In a single workflow, there could be multiple security scenarios to be considered, each with multiple protection choices and implementation locations to address confidentiality, integrity, and availability concerns. Erik Rolland et al. [22] tried to solve the flow risk reduction and control placement decision problem and showed important connections between security investment

decisions and information risk management outcomes using integer programming methods and heuristics. Doug Vogel et al. [23] included business process integration as business crosses organizational boundaries which raised the issue of protecting organizations' competitive knowledge and private information while enabling Business to Business (B2B) collaboration. This research presented a dataflow perspective using workflow management and mathematical techniques to address the data exchange problem in independent multi-stakeholder business process integration in dynamic circumstances. In contrast to our work, researchers did not mention access control constraint. In contrast to these works, we modeled the control flow of the workflow into our access control model itself.

Basit Shafiq et al. [24] proposed that tasks in a workflow need to be performed in a certain order and often would be subject to real-time constraints and dependencies. Real-time constraints may be in the form of strict deadlines that must be met by the system to operate safely. Violations of these deadlines may cause malfunctioning of the system and, in some cases, may imply the loss of human life and health. A key requirement for real-time workflow systems has been to provide the right data to the right person at the right time. This requirement would motivate dynamic adaptations of workflows. However, in the context of mobile workflow domain, deadlocks may take place which occurred in case of no participants were available to perform a task while some tasks were waiting for the completion of the tasks in first place. This problem could arise due to an error in the business rules specification. Our proposed mobile workflow model along with the task-attribute based access control model governs user access to tasks, data, and services such that it complied with control flow and access policies. That is, tasks, objects, and applications were associated with attributes that defined where and when a user could perform the access.

2.2 Access Control Mechanism

Role Based Access Control models have been a primary preference to represent the policy needs due to its efficiency in administrative management. However, the basic role-based concept was not sufficient to handle all the real-life scenarios. The RBAC model got additional features

and, therefore, contained many open-ended unresolved problems [4]. Additionally, while role-based access control was popular as a generalized approach to access control, Bertino et. al. [25] pointed out that the roles might not always be available. Bertino et. al. [25] extended the basic RBAC to include the temporal factor and called it TRBAC: A temporal Role based access control. In 2000 G.J Ahn and team, brought to attention that the web-based workflow systems, due to their heterogeneous computing environment, required a strong security mechanism but relied on basic authentication and network security. Therefore, in their work [26] they simplified the existing basic RBAC model to build an architecture that could incorporate access control into the web-based workflow system. The RBAC was incorporated to ensure that each task could only be executed by users belonging to a specific role. However, implementing the simplified RBAC would not be sufficient. In our model, we included the constraints like separation of duty, the dependency of activities, and so on to handle all the requirements of a workflow system. Former research work of James Shook et al. [27] considered the problem of insiders having access to more data than necessary to perform their job function, exacerbating the impact of leaking sensitive information. A solution to this problem was to enable the instantiation of multiple security policies within a single access control system which was best described by our Task-Attribute based access control model. Our efforts employed access control in mobile workflow systems to expresses fine-grained authorization rules for actors to perform tasks on-site and at certain time intervals.

In the RBAC model, access rights were associated with role and users were assigned to an appropriate role. In the real-world, a role could be defined as a job function that described authority and responsibility. Arindam Roy et al. [28] used the RBAC model for a commercial information system and focused on the problem of employee assignment. The objective was to identify an employee to role assignment such that it permitted the maximum flexibility in assigning a task by using 0-1 integer linear programming. To determine the optimum assignment, it assumed that the set of roles and employees in an organization was completely known. Therefore, the model was not suitable if a workflow application cross-cuts several organizations having different role structure. Our model was based on Next Generation Access Control(NGAC) and followed the

policy machine constructs [29] and, therefore, could handle the inter-organizational access control policies.

Companies today manage business information with computer systems and need a security mechanism to effectively protect important information. Moreover, they need to minimize interruptions from security mechanisms that cause delays in the execution of business activities. Task-Role based access control model (TRABC) is an integration of role-based access control and activity-based access control model [30]. Sejong Oh and Seog Park [31] proposed a methodology for access control to enable support for task level access control, partial inheritance, active and passive access control to address the enterprise business better. At the same time, they struggled to provide fine-grained access control and did not consider the effect of location and time which was very crucial. To address this problem, our model was based on attribute based access control and, therefore, could get as fine-grained as required. We also modeled location and time as one of the several attributes and considered the effects to make an access decision.

There are Workflow management systems (WFMS) which allow the execution of tasks using mobile devices like Personal Digital Assistants(PDAs) with the ability of wireless data transmission. However, the employment of workflow systems, as well as mobile technologies, could come along with special security challenges. Michael Decker, in his paper [13] has indicated the necessity of location aware and workflow aware access control model to solve the problem. In our work, we defined such access control model and provide an architecture to integrate with the workflow.

The work by Guoping Zhang and Jing Liu [32] was similar to the access control model part of our work. In their work, they had made an attempt to integrate physical infrastructure and network infrastructure of "Internet of thing" into unified infrastructure and implement workflow-oriented attribute based access control. The model granted permission to a subject based on attributes and the current task. They also had similar concepts of freezing the subject's permissions once a task was completed successfully. Furthermore, they purposed use of eXtensible Access Control Markup Language (XACML) to represent the policies and Security Assertion Markup Language(SAML) for access decisions. However, they did not provide an enforcement technique. In contrast, we pro-

vided all the features with the addition of other features like separation of duty. We also provided a model to represent location and time attribute as a single component which aided the management of attributes. We preserved the enabling and disabling feature from RBAC to work with attribute and made it more flexible and dynamic. Additionally, we chose policy machine constructs to represent the policy information which allowed graphical visualization of policies and made it more meaningful. We also provided performance analysis as a proof of concept for applicability of our access control model.

Another research model of James BD Joshi et al. [33] (Generalized Temporal Role-based Access Control) provided a temporal framework for specifying an extensive set of temporal constraints and used a language-based framework. Generalized Temporal Role-based Access Control (GTRBAC) model allowed various types of temporal constraints such as temporal constraints on role enabling/disabling, temporal constraints on user-role and role-permission assignments/deassignments, role activation-time constraints, etc. However, this research work did not include location constraint to grant access. In the mobile workflow application domain, workflow management systems were influenced by location and time information, making them very sensitive. We tended to solve the problem in our work by modeling location and time as a special purpose environment attribute, use it as a constraint in authorization and provide spatio-temporal based access control.

To sum up, in our work we attempted to provide a complete picture of a solution to a workflow management problem. The integration of access control model to the management system alone addressed a critical issue which had not been done in previous works. Additionally, the access control model was devised considering most of the constraint required for proper functioning of a workflow within or in between organization. We also investigated the verification using the formal analysis tools to ensure the reliability of the model as described. Furthermore, we added performance analysis to provide a feasible way to enforce the model into the system. Therefore, as opposed to previous work, our work covered more ground and presented effective ways.

Chapter 3

Motivation Example:Dengue Decision Support System (DDSS)

The workflow being used in this work is a representation of real-world Dengue Decision Support System(DDSS). It was initially deployed in Mexico and aimed to improve prevention, surveillance and control of the dengue vector-borne disease. It aided end-users in the prediction and quick response to outbreaks of the dengue disease. The workflow consisted of several types of tasks with certain execution order adhering to predefined control flow. Also, authorization rules did not allow a direct assignment of tasks to users, the activity/task types that could only be performed by a collaborating user depending on the attributes that the user owned, such as a job title attribute.

The DDSS has helped state-level public health officers respond to local outbreaks of dengue. The response consisted of vector control and vector surveillance, namely spraying for mosquitoes (control) and investigating locations where they might be breeding and living (surveillance) in areas where the level of confirmed dengue cases had increased above a prescribed threshold. Public health officials were organized in jurisdictions, based on population, and multiple jurisdictions were included in a single state. When the threshold was reached, officials at both levels responded. The jurisdiction officer enabled vector control and surveillance teams that were local to the jurisdiction, with instructions regarding the specific control and surveillance protocols to follow and the locations where they were to be performed. The state officer released materials for control to the team, and the local team then performed the controls and surveillance ordered. The jurisdiction and state vector control officials were often located in different buildings, although the vector control team was co-located with the jurisdiction officer. All control materials were located in warehouses elsewhere, and for coordination reasons, were controlled by the state officer. Information about specific cases of dengue was retained in what was called an epidemiological study. This data included information about the patient, the location where the patient lived (the premise),

the case, and control and surveillance actions performed at the premise. The patient and case data were considered private information and were only available to epidemiologists at the jurisdiction and state levels.

The workflow dealt with mobile teams. After completing the job, a mobile worker entered a report (e.g., what had to be done during the course of control or surveillance mosquito vector, this should be done at the target location). The workflow database was remotely accessed by mobile works to process dengue data via smartphones while they were working in the surveillance areas. Also collected were samples that needed to be taken to a state lab where the experiment was performed by a lab technician. All this information was updated in the workflow data center on regular basis. Laboratory technicians collected laboratory test data including the type of samples, method(s) used to test the samples, framework for interpreting test results, and interpreted results. The confirmation of work done would be in the form of updates to the database that had to be done from the specified location. The success of the activity was determined by analyzing the database records. In other words, updating the database with all the required information would be the end state of the workflow process.

With the legacy DDSS, vector control and surveillance teams went house-to-house and collected data in paper-based forms, and then data were entered into the system. However, entering data from paper sheets had many disadvantages. Paper-based data collection was time-consuming, error-prone, degraded the system performance (i.e., late responses), and these forms might easily get lost or damaged. Furthermore, much of the collected information was never entered into the system and, henceforth, the analysis precision was lost. Mobile workflows could solve the paper-based forms problem; it expedited emergency responses and tracking the disease evolution. With the help of cell phones, data were instantaneously collected and transmitted to the workflow data center in a reliable manner. In case network access was not available, data were stored locally in cell phones and transmitted later. However, mobile workflows has had their own security and usability challenges to which we discussed the solution in the following chapters.

Chapter 4

System Architecture

The workflow management system needs to ensure the correct representation, instantiation, and execution of the business processes while considering the authorization rules and control flow. In this chapter, we introduce an architecture that helped us to obtain the constructs and the objectives of a mobile workflow. Through the architecture, we were able to generate a fully functional workflow assigned to appropriate users and processed following all the constraints. Also, we required instant management of dynamic situations where the configuration to predefined workflow changed.

The system architecture was composite from collaborative modules responsible for the definition, execution, and management of business processes as well as the implementation of access control, control flows, and constraints. The architecture is shown in Figure 4.1 is composed of three major components: Workflow Formulation Module (WFM), WorkFlow Management Module (WMM), and Access Control Module (ACM).

4.1 Workflow Formulation Module

The Workflow Formulation Module (WFM) formulates the essential components of a workflow, using its authoring and analyzer tools. The authoring tool would allow to state required workflow tasks and define the constraints between those tasks. The analyzer would re-evaluate the specification and allow improvements. Therefore, the WFM would helped us create a functional representation of the desired goal.

Through the WFM module, an application user could form a graphical representation of well-ordered activities of a business process along with business rules for collaborative users to process and to invoke necessary documents and services required to perform those activities. Each task/activity would be represented by a node in the workflow graph as shown in Figure 4.1. Directed arcs in Figure 4.1 represent the flow of control (and information) and have two properties,

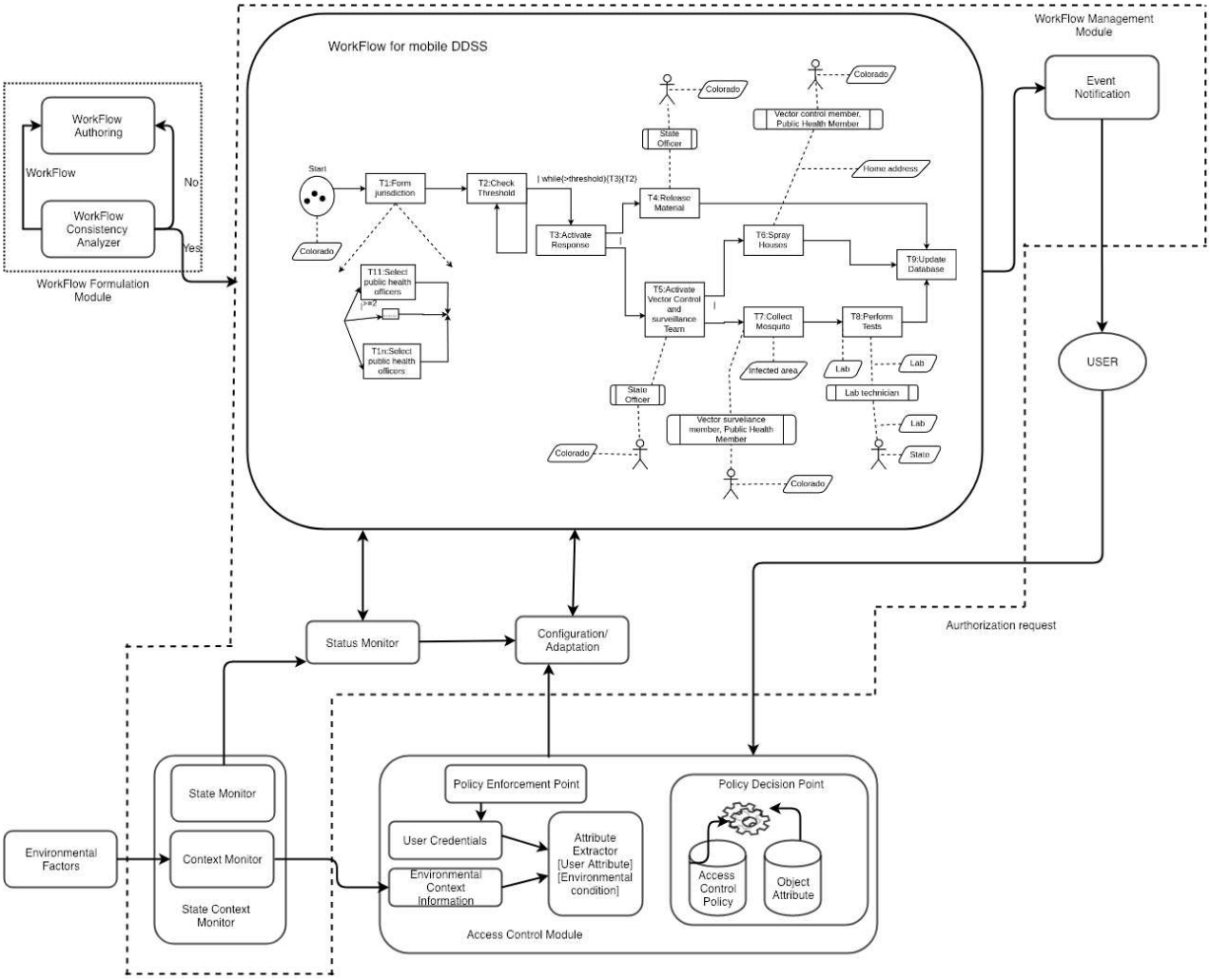


Figure 4.1: System Architecture

arc “previous node" and “subsequent node". A workflow node may have one or more input and output points. An input point would be connected with a previous node by an arc while output point would be connected with a subsequent node via another arc.

Module WFM defined three types of nodes in a workflow model denoted by a source node, sink node, and task node. The source node would be the start node of a business process, which might have multiple source nodes. A source node would have one output point and no input point. When a workflow reached the sink node, the business process would be ended. A business process would have one or more sink nodes, which would represent the end of the business process. A sink node would be one or more input points and no output point; it would represent the process termination. The task node would depict a task that forms one logical step in the business process; it would have a particular “task name”. The task node would have one or more input and output points.

The WFM module would allow one to formulate the control flow, which defines the dependencies between task nodes in a workflow graph. Task dependencies would depict the execution order of tasks in a workflow. The execution order of tasks could be sequential, mutually exclusive split, and split, joins, or conditional repeats. Two tasks could be sequential if one task was bound to be done before or after the other. For example, task of spraying the houses (T6) would need to be done before any changes to the database were made (T9). Therefore, T6 and T9 would be sequential. If the tasks could be performed independently of each other and either of them were sufficient to reach the goal, they were ordered as mutually exclusive split. In our example, this kind of split was not shown. However, if the Manager of the system decided that spraying houses and mosquito sample were not required from all the areas but only some predefined areas than the split between the task T6 (spray houses) and T7 (collect mosquito) would be a mutually exclusive split for those areas where only one of the actions was required, because execution of one path would be sufficient for the completion of workflow. The tasks that could be performed independently but both were required to reach the goal were ordered as the and split. The task T4 (Release Material) and T5 (Activate VC and VS team) underwent the and split. Joins contributed in ordering the tasks

where a task would depend on the completion of two or more tasks. Similarly, conditional repeats were the cases where the same task would need to be performed repeatedly until some conditions were satisfied. For example, the subtask T11 to T1n (selecting a public health officer to form jurisdiction) was the repetition until all the required teams were formed with the required number of members. Also, the task T2 (check threshold) was repeated until the threshold was exceeded.

Finally, the authoring tool would allow the specification of the authorization rules.It should support the specification of spatio-temporal constraints that may be required for various purposes like accessing workflow resources or execution of business process tasks. With WFM, spatio temporal constraints could be associated with different components of a workflow like the collaborating users, the assigned tasks, the access to sensitive objects. The WFM would make sure that the tasks were being operated by authorized users and in accordance to spatio-temporal conditions, through the help of access control model. For example, the task of updating the condition of the lab should only be done from the lab or should only be done by someone who knows how to analyze the lab. Access control model would provide the information to maintain authorized access to resources and follow the constraints like separation of duty (SoD). It would do nothing to implement the rules but simply states the requirements of the system. Due to the presence of all these elements,the template produced by incorporating all the feasible task, their dependencies, and the authorization specifics might not be the final workflow. The consistency analyzer, would, thereafter, verify that the template was consistent and correct. The analyzer unit would be able to point out any kind of deadlocks, irregularity, and ensure the completeness upon instantiation. For example, the analyzer would be able to tell if there existed an infinite repeats in the template due to the condition specified or splits and the joins together obstruct the achievement of workflow goal.

The WFM could improve the initial template by switching between two of its component: authoring unit and the analyzer and returns a workflow that fulfilled the business requirement. Figure 4.1 shows a workflow with tasks, control flow, and spatio-temporal constraints in it. A workflow model is formally defined as below.

- A workFlow (W), $W = [id, T, C, A]$, has one start node and one sink node, is a 4-tuple of elements:
  - id is the unique identification for the workflow
  - T is the set of tasks
  - C is the control flow or the dependencies between the tasks. The orders of C belong to {sequential, mutually-exclusive-split, and-split, join, triggered sequence, conditional repeats }. Therefore C is defined as subset of $(T \times \ order \times T)$. The control flow is defined between two tasks, whereas, there could be a relationship between more than two tasks in a workflow. Therefore, there might exist more orders related to the same task. For example, if task (A) splits into B and C, and the split is an and-split, we would represent such a scenario by defining multiple orders. Here, we would say, B comes after A, C comes after A, and B and C are and-split.
  - A is the set of authorization rules. Set A also includes other constraints required for the workflow like spatio-temporal. Mobile workflow authorization rules and constraints in the context of spatio-temporal attributes are discussed in section 4.2.1.

4.1.1 Example Workflow model

Following is the workflow model of mobile workflow based on our motivation example in 3

- Mobile Workflow for DDSS $(mWDDSS) = \{W1, T, C, A\}$, where,
  - $W1 = id\ of\ the\ workflow$
  - $T = \{FormJurisdiction(T1), SelectPublicHealthOfficers(T1_s),$ $CheckThreshold(T2), ActivateResponse(T3), ReleaseMaterial(T4),$ $ActivateVectorControlAnd\ SurveillanceTeam(T5), SprayHouses\ (T6),$ $CollectMosquito(T7), PerformTests(T8), UpdateDatabase(T9)\}$
  - $C = \{CR(CR(T1_s, n >= 2), N), (T1, S, T2), CR(T2, infection > threshold),$ $(T2, S, T3), (T2, ES, T3)(T3, S, T4), (T3, S, T5), (T4, AS, T5), (T5, S, T6),$

$(T5, S, T7), (T6, AS, T7), (T4, S, T9)$,$(T6, S, T9), (T7, S, T8), (T8, S, T9)\}$,
$where, S = Sequential, CR = ConditionalRepeat, EX = MutuallyExclusive$
$Split, AS = AndSplit$

- A ={*(Manager needs to handle T1, T2, and T3 from the head office), (State officer releases the material from the warehouse), (Jurisdiction officer activates the Vector Control and Surveillance Team), (Vector Control Team sprays houses at the home address), (Vector surveillance Team collect the mosquito sample from the infected area), (Lab technician perform the test inside the lab),(Everyone is allowed to update the database from their designated location),(single person cannot activate both vector control and surveillance team attribute within the same instance)* }

According to the mobile workflow model, the workflow model consisted of 9 tasks, where, T1 was the collection of subtasks (the task that was repeated for completion of another task) represented by $T1_s$. For, completion of T1, $T1_s$ was repeated until a team of two or more mobile participants was formed from all the available public health officers. T2 continuously monitors the threshold and, therefore, was repeated until the infection was known to be greater than the threshold. Here, T2 underwent mutually exclusive split to T2 and T3. T2 either continued or expanded to T3. The split after T3 could be done in parallel and every task would independently update the information to the database. Each task had its spatio-temporal restriction and required attributes as described in the set A.

4.2 Workflow Management Module

The WorkFlow Management Module (WMM) was responsible for the implementation and management of the workflow. To execute the workflow, WMM allowed administrators to create an instance of the workflow model by assigning authorized users to tasks. A workflow could have different instances. Different instances could be executed by different sets of users, subject to their availability and as permitted by their authorization policies. Different instances may execute different sets of tasks. A workflow instance could be formalized as follows.

- WorkFlow Instance(WI) = $\{id, Wid, TI, UTA\}$ where,
  - id is the unique identification for the workflow instance
  - Wid is the id of the workflow which was instantiated to create this instance
  - TI is the instance of the task. TI includes a subset of task in Wid . i.e. $TI \subset T, where\, T \in W \wedge W.id = Wid$
  - UTA is the user to task activation. The task activation is done considering the access control model and requires a proper user attribute set assigned to the user before hand.

Since only authorized users could be assigned to any task, WMM communicated with the Access Control Module (ACM) to make access decisions. The access decision was based on the workflow instance, spatio-temporal constraints, and access constraints. For example, we might have two instances of the mWDDSS workflow model for two different states. The user assigned to the task would be authorized only if they belonged to the same state. Similarly, all other location constraints like the infected area or the warehouse would be based on the state. After the instance had all its tasks assigned to authorized users, the user was notified about their tasks via the event notification unit within WMM.

During the runtime, there might occur changes which might require attention for it to continue. For example, A vector control team might not get enough supplies from a particular warehouse or might not reach the destination at the exact time due to some unforeseen disturbances. In such cases, the WMM should be able to reconfigure the assignment and allow the workflow to execute by performing suitable changes in the configuration. Any changes to the execution plan during the runtime would be monitored by the status monitor. It would maintain the record for execution status and notify the configuration and adaptation unit with the changes that triggered the process.

Similarly, the State Context Monitor would continuously monitor the changes in environmental factors. The changes that would affect the execution of the workflow would again trigger the configuration and adaptation unit. For example, when the lab technician left the lab the address would change and thus would the access right. Therefore, the adaptation unit might have to communicate

with ACM and might have to assign another lab technician to the workflow task. At the same time, the previous lab technician should not have the access right to any of the information within the lab. It would be the responsibility of the configuration and adaptation unit to determine the most optimal solution to complete the workflow goal whenever there was a requirement of changes in the former workflow. The changes would be done in either of task instances or the activation and the ids would remain the same.

4.2.1 Example Workflow Instantiation:

Here we assigned the task to the authorized users by careful consideration of their attributes including spatio-temporal conditions. The workflow instance is,

- $WorkflowInstanceforDDSS(WIDDSS) = \{WI1, W1, TI, UUATA\}, where,$
  - $WI1 = id\,of\,the\,instance$
  - $W1 = id\,of\,the\,workflow$
  - $TI = \{T1, T2, T3, T4, T5, T6, T7, T8, T9\}$, here the task instance consists of all the task from the workflow. The task would have been instantiated with just two tasks initially, T1, T2, since T2 and T3 have mutually exclusive split. And the configuration and adaptation unit would make appropriate changes when the infection rate is more than the threshold. Here we have considered having infection rate more than the threshold.
  - $UUATA = \{(T1, Alice), (T2, Alice), (T3, Alice), (T4, Bob), (T5, Dave),$
    $(T6, Shan), (T6, Tim), (T6, Shelly), (T7, Phil), (T7, Lara), (T8, Evan),$
    $(T9, Bob)(T9, Evan), (T9, Shan), (T9, Tim), (T9, Shelly), (T9, Phil),$
    $(T9, Lara)\}$

Access Control Module was being reached to address the constraints during the activation of a task. All the rules specified on workflow definition was addressed during workflow instantiation. In the mWDDSS workflow, for example, Alice was the Vector Manager who formed the jurisdiction from public health officers and dispatched the state officer if the infection was severe. The state

officer (Bob) would release the material and Dave (who is the jurisdiction officer) would assign a task to each vector control and surveillance team member. The control team consisted of three members (for this instantiation–Shelly, Tim, and Shan) who were responsible for spraying the houses. After the process was concluded, they would update the database. The surveillance team was composed of Phil and Lara who would collect mosquito samples. The sample was tested by Evan who updated the database with the results.

Throughout the process, there existed many constraints. A few of which were: all the attributes were activated within the state, Manager must be in the head office within the state during the daytime, the state officer was mobile, would be able to activate the release from his office but update the database only from the warehouse where he would inspect the material being used at daytime. Similarly, the vector control team and surveillance team were mobile and would be able to activate their attributes and update the database at an infected area and during the assigned control and surveillance work period only. These dynamic constraints were looked into during the activation of the task or during the runtime. There also existed the separation of duties constraints between the control and surveillance tasks. Since, both the teams were formed by public health officers, a single person might belong to multiple teams in different instances. Therefore, T6 and T7 could not be assigned to the same user within an instance of the mWDDSS workflow, which was followed in this instance by the assignment of a different user. All these decisions were made by intervening the Access Control Module. The WMM would only communicate with ACM to make the appropriate decision during each step of the workflow execution.

4.3 Access Control Module

The Access Control Module(ACM) was responsible for making authorization decisions either during the assignment of the workflow tasks, workflow model, or during the execution of the tasks, workflow instances. Authorizations were triggered by the changes in the active user attributes and the mobile data objects required to execute tasks. User attributes represented a user of the workflow attempting to execute tasks such as current location and time, user health, user ID, job title,

department, etc. Since mWDDSS workflow was accessed by nomadic devices from diverse geographical locations and during different time intervals, spatiality and temporality were important security measures that must be considered. Healthcare information on dengue patients should be protected from unauthorized access. Public health professionals should be allowed to access their patients' records only in the same areas where the dengue case occurred and during the course of dengue. In mWDDSS workflow, a spatio-temporal evidence that VC and VS teams performing their tasks in the right place and time was also mandatory for the credibility of data and disease control.

With mWDDSS workflow, the spatio-temporal constraints could be used at different levels for different scenarios. Constraints on task assignments expressed that a task could be activated only on that location and at certain time intervals. Constraints on user expressed that a workflow collaborative user was granted access to protected resources and services only from a certain location and time. Constraint on user task assignment expressed that the task could be performed by the user having some attributes only on a particular location and during the course of dengue. Spatial-temporal constraints were also applied to workflow instances which was the case of having the same instances on different states.

Therefore, ACM was responsible for the proper execution of the workflow. It played a very important role from the very beginning (i.e., instantiation of the workflow). The decision made by the access control module allowed the WMM to select and assign collaborating users to particular tasks. The use of ACM was also prominent for the dynamicity (mobility) of workflows. Without ACM, it would not have been possible to handle the situations where location and time constraint changed for a workflow user. It made workflow operational in very critical situations. For example, a mobile user having harmful chemicals should not be allowed to use it in all the locations and the ACM ensured that, by revoking the access authorization from the users whenever s/he is not within the allowable location-time zone. At the same time, the workflow continued by activation of some other user for the same task.

ACM was composited from three components to achieve its functionalities. The ACM consisted of the Policy Enforcement Point (PEP) which was the point of contact with any other module in the system. It received requests from other modules and delivered the access decision. However, it did not have any capability to process the request itself, therefore, it parsed the input data to acquire user credentials along with any other useful information. This information was transferred to the Attribute Extractor Point (AEP) which determined the user environment and requested object attributes. To determine the environment attribute, AEP received information about the environmental context which was provided by the state context monitor. All this information was further provided to Policy Decision Point (PDP). PDP was where all the evaluation of the access request took place. PDP took in the information and evaluated it for the decision-making process of whether or not the user should be given the queried access. The PDP would have all the required information for the job (i.e., the attribute based policy, the attributes of the users and objects). The input to the PDP would have the user and object environment attributes, location-time information. The AEP and PDP collaboration would determine the access decision and forwards it to the PEP. The access control module, as a whole, was a black box for other modules, which took the input query and returned the appropriate access decision. The specification of the attribute-based authorization policy and the processing of the queries (access requests) by the module is described in Chapter 6.

Chapter 5

Location and Time Models

In mobile workflow systems, each entity and relation was constrained by location and time information; we now introduce how spatio-temporal information was modeled in our workflow system.

5.1 Location Model

The capability to determine the position of a user nomadic device (UND) has enabled application developers and wireless network operators to provide location and time aware services. The Global Positioning System (GPS), has been a celestial-based navigation system that uses at least 24 satellites for determining a position, getting from one location to another, monitoring object or personal movement, creating maps of the world, and getting precise time measurements [34]. Nowadays, GPS sensors are almost available on all modern handheld computing devices. Some examples of GPS applications include safety application that alerts drivers for exceeding the speed limit in school zones during the day, 8:00 am - 4:00 pm [35]. iFall is another GPS application in the area of mobile-healthcare that detects and alerts medical professionals of a patient's fall [36].

Kupper [37] provided an overview of many available techniques for the determination of the mobile user's position (locating a user nomadic device). Two methods are currently available for location determination, namely self-locating and remote-locating. The self-locating method permits the client mobile to present the location information to the workflow server using "Global Positioning System" (GPS), while the remote-locating method determines the device location by the network via the CellID-approach of mobile telephony like GSM or UMTS. It was worth mentioning that GPS and CellID approaches lack positioning inside buildings. Indoor localization technologies that could be used which included Active Badge and Bat, WIPS and CRICKET, or WiFi-based systems (for both self- or remote-locating) [38, 39]. We did not consider the trustwor-

thiness of the location determination system in our mobile workflow model, so we recommended using mixed positioning methods.

Self-locating methods do not guarantee the enforcement of location sensitive systems under the assumption a user or malicious attacker of a client mobile could invent arbitrary location information, such as “GPS Spoofing", to the workflow server allotting access to location and time-sensitive resources. This kind of access control attack would be hard to prevent and the countermeasures would not be practical for authenticated positioning systems [40], spoofing countermeasure could add substantial overhead and degrade the client mobile performance. The GPS Spoofing attack could cause GPS receivers to provide false information about position and time by broadcasting counterfeit signals similar to original GPS signal or by recording original GPS signal captured somewhere else in some other time and then retransmitting the signal [41].

The remote-locating method was more suitable for our workflow model as it was almost impossible to tamper with the positioning information. Examples of CellID based remote-locating in commercial services were guiding systems, friend finder, shopping assistance, presence services, community and communication services, information services giving the mobile user information about their surroundings, etc. Governments also put requirements on the network operators to be able to determine the position of an emergency caller. For instance, USA government mandates (FCC E-911) network operators to determine the position of a certain percentage of all emergency calls, with high accuracy and within a pre-specified maximum time, especially when the celestial positioning method (GPS) was not available or the UND was not capable of celestial positioning methods.

However, cell identity positioning was not sufficiently accurate compared to GPS. Therefore, we recommended mixed positioning methods, which combined the identity positioning method that made use of CellID and celestial positioning method employing GPS to define every positioning unit cell. That meant workflow server could verify the position information reported by the client mobile GPS to find out if it actually lied in the cell positioning obtained by the CellID method.

The mobile workflow presented in this paper implemented the following location representation model. There were two types of locations: *physical* (i.e., provided by GPS or CellID systems)and *logical*. All users and objects were associated with physical locations that corresponded to the physical world. A physical location was formally defined by a set of points in a three-dimensional geometric space. A *physical location* $ploc_i$ was a non-empty set of points $\{p_i, p_j, \ldots, p_n\}$ where a point p_k was represented by three coordinates. The granularity of each coordinate was dependent upon the application.

Physical locations were grouped into symbolic representations that would be used by applications. We refered to these symbolic representations as logical locations. A logical location was viewed as a non-empty area (e.g., a polygon, circle) within the reference space points. Logical locations could be categorized, e.g., “city", “state", “country" or “building". Examples of logical locations of category “city" and “state" were *Fort Collins* and *Colorado*, respectively. A *logical location* was an abstract notion for one or more physical locations. Mapping function m was defined to convert a physical location to a corresponding logical one.

Definition 1. [Mapping Function m**]** *m is a function that converts a physical location into a logical one. Formally, $m : P \longrightarrow L$, where P is the set of all possible physical locations and L is the set of all logical locations.*

All location instances of a given category may not cover the whole reference space points. Therefore, two locations from different classes may overlap in space points or one location may contain another one (i.e., “city" Denver is in “state" Colorado). We defined the *containment* $\subseteq$ and *equality* $=$ on physical locations. A physical location $ploc_j$ was said to be *contained in* another physical location $ploc_k$ if $ploc_j \subseteq ploc_k$. Two physical locations $ploc_r$ and $ploc_s$ were equal if $ploc_r \subseteq ploc_s$ and $ploc_s \subseteq ploc_r$.

We defined a logical location called *anywhere* that contained all other locations. Each application could describe logical locations at different granularity levels. For example, some authorization rules may be applicable to the entire state, whereas, others were only applicable to people in the class “city". Let us denote the logical locations that were of interest to the application by

the set **L**. Let the physical locations corresponding to these logical locations be denoted by **P**. The size of the smallest location in **P** corresponded to the *minimal location granularity* of the application. For example, in the organization Software Development Corporation, we may have **L** = {MainBuilding, TestingOffice, DirectorOffice, DevelopmentOffice}. The *MainBuilding* houses the three offices on separate floors of the building. In this case, the minimal location granularity was one floor.

5.2 Time Model

GPS and CeLLID report to the system the current client mobile location and time. Our workflow model used two kinds of temporal information. The first was known as time instant and the other was a time interval. A *time instant* was one discrete point on the timeline. A *time interval* was a set of consecutive time instants which could be represented in the form of $d = [t_s - t_e]$, where t_s, t_e represent time instants and t_s precedes t_e on the time line if $t_s \neq t_e$. We use the notation $t_i \in d$ to mean that t_i is a time instant in the time interval d. The exact granularity of a time instant was application dependent. Suppose the granularity of time instant in an application was one minute. In this case, time interval [3:00 a.m. - 4:00 a.m.] consisted of the set of time instants $\{3:00a.m., 3:01a.m., 3:02a.m., \ldots, 3:59a.m., 4:00a.m.\}$.

Two time intervals may overlap as well in some instances depending on the system time granularity. Now we defined the containment $\subseteq$ and equality $=$ on time intervals. A time interval d_j is said to be *contained in* another time interval d_k if $d_j \subseteq d_k$. Two time intervals d_s and d_r were said to be equal if $d_r \subseteq d_s$ and $d_s \subseteq d_r$. We defined a time interval called *always* that included all other time intervals. The set of all time intervals of interest to the application was defined by **I**. The *minimal time granularity* of an application referred to the size of the smallest time interval used by the application. For example, in the Software Development Corporation, we may have the following intervals that were of interest: **I** = $\{i_1, i_2, i_3, i_4\}$, where $i_1 = [8a.m. - 5p.m.]$, $i_2 = [8a.m. - 12p.m.]$, $i_3 = [12p.m. - 1p.m.]$, and $i_4 = [1p.m. - 5p.m.]$. The minimal time granularity pertaining to this application was one hour.

Chapter 6

Task-Attribute Based Workflow Authorization Model

Access control ensures that enterprises could deploy security policies that met their most stringent business needs to protect their sensitive data. Thereby, our objective was to introduce a dynamic access control model capable of expressing a wide set of spatio-temporal policies that ensured users had the appropriate access to the correct systems, resources, and workflow applications. Our proposed spatio-temporal access control model, denoted by Task-Attribute Based Access control (TABAC), embodied the essential aspects of the “next generation" authorization model known as Attribute Based Access Control Model (ABAC) to provide task-based authorizations model capable of expressing dynamic security policies. Users perform tasks on workflow based on their attributes as well as the current time and location information. Thus, spatio-temporal constraints determined when workflow tasks could be enabled or disabled. Although a user acquired all the required attributes to perform a workflow task, the user could only operate on that task only if the user attributes had been enabled by location-time constraints, which made the model inherently dynamic in nature.

Attribute based access control defined a logical relationship between requesting entity “subjects" (i.e., Individuals or Autonomous Entities) and sensitive information asset “objects" (i.e., Protected Resources) in terms of the attributes associated with each of them, the attributes could be about anything and anyone. The model defined three attribute categories: (i) user attributes – characteristics of the user that are important from the access control point of view. These included user identity, job title, workflow task (activity), age, gender, clearance, department, company, competencies, etc. (ii) object attributes – these describe characteristics of the protected resources. Examples included the classification, sensitivity, location, type (bank account, medical record, reimbursement form, vacation document), mode, etc. (iii) environment attributes described

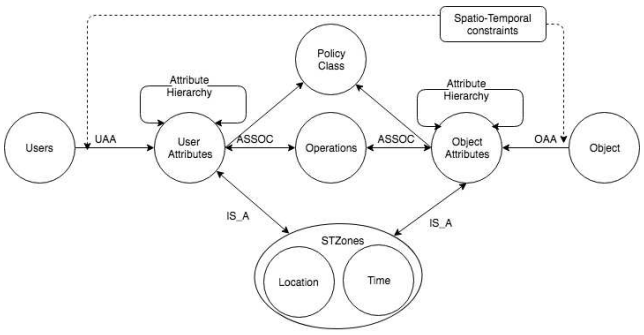


Figure 6.1: Task-Attribute Based Access Control Model

environmental conditions of access request scenario such as time and location. An access control policy brought together those attributes to express access was allowed or denied. For example, an access control policy may mandate a user with job title “department manager" attribute to approve a reimbursement document if the document was in the same department as the user and during the daytime, 8:00 am - 5:00 pm. The latter attribute may be represented as part of the user and object attributes as well.

We now discuss the formalism of integrating environmental conditions into ABAC for expressing access control policies to protect mobile Information Systems. This model allowed one to express policies where access to sensitive resources depended on the associated attributes with subjects and objects as well as the current location of the subject and time of access. We defined our model in terms of a standardized set of relations and functions of policy entities capable of expressing and enforcing a wide range of security policies. The model defined key relations of assignments, that were a many-to-many mapping of policy entities in a meaningful authorization structure, associations specified the user authorizations on objects, prohibitions defined the inverse of association, and obligations defined a set of automatic administrative actions taken in response to events. Model key functions assisted in examining the prerequisite constraints on access rights of users to objects and policy enforcement. Figure 6.1 shows entities, relations, and functions interpreting a system policy in our access control model.

6.1 Entities

The basic policy elements to express in the model included authorized users (U), objects (O), user attributes (UA), object attributes (OA), operations (OP) and policy classes (PC).

6.1.1 Users

Users were the entities that would access to sensitive resources in accordance with policy and require guidance for the efficient and safe use of the resources. A user represented an authenticated individual or an autonomous agent who directly interacted with a system. Users in our model could be mobile or static, and each user had a unique identification in the system.

- Users (U) : A finite set of authorized users which we denote as; $\{u_1, u_2, u_3, ...\}$.

For mobile users, our assumption was that the users would have some mechanism to give away their location, and the user location changed with time. The spatio-temporal zone provided the user's current location and time. For example, if a user was responsible for releasing some materials from a warehouse located in some city and the head office was located in some other city, we had to ensure that the user had traveled from the head office to the warehouse location before allowing him to actually release the material from the warehouse. The current spatio-temporal zone of the user was returned by function $currentzone$. This function was formally defined as follows:

- $currentzone : U \rightarrow STZones$

6.1.2 Objects

Objects were the system's sensitive resources that required protection from unauthorized access. The object may be the file systems, database systems (cells in fine grained representation) or devices like sensors, actuators. As users, each object had a unique system-wide identifier and it could be mobile. Additionally, access to the object may be restricted to a certain location and time. Such factors would be represented as their attributes.

- Objects (O): A finite set of protected objects denoted as; $\{o_1, o_2, o_3, ...\}$.

6.1.3 User Attributes

Access control policy decisions about permitting or denying certain types of access were regulated by attributes of users and objects of a system. User attributes (UA) allowed a logical categorization of users. The important user attributes for the model were their job title as well as the spatio-temporal zone if the user was mobile. For example, a user could be a manager to an organization located at Fort Collins but would not remain a manager in his home at Loveland. The spatio-temporal zone attribute as one of the user attributes would be describing the particular environmental conditions for a user to achieve a particular attribute adequately satisfying policy. Another important attribute of a user was the task assigned to them with respect to the workflow instance.

- User Attributes (UA): A finite set of user attributes denoted by; $\{ua_1, ua_2, ua_3, ...\}$.

6.1.4 Object Attributes

User and object attributes played a similar role in policy. Like user attributes (UA), object attributes (OA) objects had object attributes in order to govern access to objects, represent allowed and denied access modes to specific users. Objects may also be mobile as the user, thereby spatio-temporal attributes could be a property for each object of a system as well. Spatio-temporal attributes designated the locations and time when and where an object could be accessed. Here, again, we had to locate devices that tracked the location of an object. For example, a file which was a part of information related to organization "A" would have an attribute "A". It might also have an attribute location and time, which was the location of the organization itself and during the daytime, if the file was to be accessed only inside the organization. As a user attribute, a workflow task was not an object attribute. The relation between the workflow task and the objects would be represented through the association relationship between the user and object attributes

- Object Attributes (OA): A finite set of object attributes that is denoted as; $\{oa_1, oa_2, oa_3, ...\}$.

6.1.5 Spatio-temporal Zones

STZone was one of the core components of the authorization model, which was linked to model entities and restricted relationships. A spatio-temporal zone abstracted location and time representation into a single unit. STZone was one of the core components of the authorization model, which was linked to model entities and restrict relationships. This meant the entity availability was proportional to spatio-temporal zones that defined where and when the an entity was accessible.

Definition 2. [Spatio-temporal Zone] *A spatio-temporal zone STZone is a pair of the form* $< l, d >$ *where* l *and* d *represent the logical location and the time interval, respectively.*

An example of a spatio-temporal zone could be *z = (HomeOffice, [6:00 p.m. to 8:00 a.m])*. The set of spatio-temporal zones was denoted by $STZones$ which represented a set of pairs for locations and intervals in an organization that defined where and when some resources were available. An example of a spatio-temporal zone set would be *{(HomeOffice,[6:00 p.m. to 8:00 a.m.]), (DeptOffice, [8:00 a.m. to 6:00 p.m.])}*. The set $(STZones)$ appeared in the model as object attributes and/or user attributes.

- Spatio-Temporal Zones $(STZones)$: A finite set of user spatio-temporal zones that is denoted by,
  $\{z_1, z_2, z_3, ...\}$, where $STZones \subseteq UA \cup OA$.

A spatio-temporal zone $< l, d >$ were specified at *minimal granularity* if l and d are specified at minimal location granularity and minimal temporal granularity, respectively. In other words, STZone $< l, d >$ had the location and time interval that did not contain other location or interval, respectively, in the context of the applications. Formally, in STZone $< l, d >$, location l is minimal *iff* $\neg\exists\ l' \in P,\ l' \neq l$ such that $l' \subseteq l$, and time interval d is minimal *iff* $\neg\exists\ d' \in I,\ d' \neq d$ such that $d' \subseteq d$.

STZone contents may be defined as special classes (types) of constraints, such as temporal, spatial, and strong constraints. The *universal zones* content defined strong constraints that should hold at any time and in any locations, (i.e., $z_u =< anywhere, anytime >$). The second content of

zones was *temporal zones* that expressed temporal constraints that should hold at any location but during a certain period of time i, (i.e., $z_i =< anywhere, i >$). The *location zones* content specified the location constraints that should hold at any time but in location l, (i.e., $z_l =< l, anytime >$).

Functions *ZInt* and *ZLoc* were defined to return the interval and location in a zone, respectively. These functions were important to elaborate on the content of a zone in order to define granular constraints. STZones, *ZInt*, and *ZLoc* were formally defined as following:

- $STZones \subseteq L \times I$

- $ZInt : STZones \rightarrow I$

- $ZLoc : STzones \rightarrow L$

6.1.6 Tasks

Tasks were special purpose entities introduced into the model to incorporated access control for the workflows. Tasks represented the workflow tasks and in the model were represented as one of the user attributes. In other words, a set of tasks were the subset of user attributes set. The presence of a task as a user attribute would signify that the user had all other attributes necessary to perform the task. It was because a user would qualify to have the task as its attribute if and only if it had all the attributes required to perform the task. For example, in the mWDDSS workflow system, if a task 'T' of forming a VC and VS teams was to be performed by a Control Manager, then task 'T' would be an attribute of the user who already had the Control Manager as one of its attributes.

- Tasks(T): A finite set of workflow tasks denoted as , $\{T_1, T_2, T_3, ...\}$ where , $T \subseteq UA$

6.1.7 Operations

The operations represented the entire set of system actions that could be performed on the object resources and data by authorized users. The operations set comprised administrative operations that could be performed on policy elements and relations as well. Some operations relative to the

model could be read, write, update, execute, and so on. A resource operations example could be a bank teller reading and updating a piece of account information during working hours and inside a bank branch. An administrative operations example, on the other hand, could be a policy officer creating or deleting some policy data elements and relations in accordance to meet organizational security objectives.

- Operations (OP): A finite set of operations that is denoted by; $\{OP_1, OP_2, OP_3, ...\}$.

6.1.8 Policy Classes

The policy class was the container for all other entities. It helped to organize and distinguish different types of policies. When the system had more than one policy class, each class was a specific policy and permissions were granted only if it was allowable with respect to all the related policy classes. For example, a user who was a doctor for some hospital could also be a patient for some other hospital. If policies of two hospitals were represented as separate policy classes, any privilege given to the user should satisfy both the policy classes.

- Policy Class (PC) : A finite set of policy classes; $\{PC_1, PC_2, PC_3, ...\}$.

6.1.9 Policy Elements

We defined the set of all policy elements as a set PE, which helped to render access control decisions. Policy elements set PE not only included the users and objects of a system, but also attributes of those elements as well as policy classes to represent the authorization structure. The way in which policy elements could be structured and utilized to express policy is covered in subsequent sections.

- Policy Element (PE) : A finite set of policy elements; $PE = (U \cup UA \cup O \cup OA \cup PC)$

6.2 Relationships

Relationships bound the entities together to form a meaningful authorization structure. Thus, formed structure was analyzed to derive the access decision. Following are the relationships in our model and their features.

6.2.1 Assignment

The assignment relationship represented the container-ship between the entities. It entailed that the contained entity inherited properties from the container. The assignments were the means to represent the relationship between the entities [29]. For example, a user who was a student and lived in Denver would be assigned attribute student and Denver. The assignment implied that the user would have the functional properties of student and location properties of Denver. Based on the policy entities, various assignments existed in the access control model.

User-Attribute Assignment

A user may be assigned to one or more user attributes. The $UAAssign$ assignment was represented by $(u, ua) \in UAAssign$ to indicate that the user u was assigned to the user attribute ua. The $UAAssign$ assignment was location and time-dependent. That is, a user could be contained by a spatio-temporal zone as a user attribute. For example, a user could be the on-campus student only when he was on the campus during the semester. This requirement was expressed using the STZone attribute. Similarly, UAAssign was also responsible for the assignment of a workflow task to a user. As with all other attributes, a single task could be assigned to multiple users and vice versa.

- $UAAssign \subseteq U \times UA$

We defined the function $assignua$ which mapped each user to a set of attributes, noted that each user must be mapped to at least one attribute in a workflow template, otherwise the user had no access. Also, function $assignu$ gave the set of users assigned for a given attribute.

- $assignua : U \rightarrow 2^{UA}$

- $assignu : UA \rightarrow 2^{U}$

Object-Attribute Assignment

Like the UAA assignment, an object may be assigned to one or more object attributes represented as a binary relation $(o, oa) \in OAAssign$ to express that object o was assigned to the object

attribute oa. An object attribute may be contained by one or more spatio-temporal zones to govern access to the contained objects by authenticated users only in specific locations and time. The spatio-temporal zone of an object gave the location and time from which operations on that object could be performed by users. For example, a permission to perform a backup of servers could be executed only from the department after 10:00 p.m. on Friday nights.

- $OAAssign \subseteq O \times OA$

User Attribute-Policy Class Assignment

A user attribute was assigned to or contained by one or more policy classes to express and enforce distinct types of policies. The $(ua, pc) \in UAPCAssign$ assignment meant that the user attribute ua was assigned to or contained by the policy class pc, where $ua \in UA$ and $pc \in PC$. This assignment also derivatively signified that user access was controlled by the container policy class and the object access was protected by the container policy class. For example, a public health officer for some locality could also be a resident of that locality. If we had two organizations governing the policies of that locality, the officer might be bound by both the policy classes and, thus, assigned to both the policy classes. Here, the attribute public health officer could be assigned to policy class by organization A and resident could be assigned to policy class by organization B since the individual was the same.

- $UAPCAssign \subseteq UA \times PC$

Object Attribute-Policy Class Assignment

Like the $UAPCAssign$ assignment, an object attribute may be assigned to or contained by several policy classes, we expressed this assignment by $(oa, pc) \in OAPCAssign$, where $oa \in OA$ and $pc \in PC$. For example, the house to be sprayed might be the information of DDSS system and, thus, might have attribute infected houses and member of some policy class. At the same time, the house could also be a record of government information with different policy class, and, therefore, could be contained by the policy class maintained by the government as well.

- $OAPCAssign \subseteq OA \times PC$

Now, assignment A was the set of all the aforementioned assignment relationships,

$A = \{UAAssign \cup OAAssign \cup UAPCAssign \cup OAPCAssign\}$

6.2.2 Attribute Hierarchy

In many cases, two attributes might have some of the privileges in common. In other words, an individual might have some additional privilege than another individual with similar (but not the same) attribute. For example, a state-level clinician could have all the privilege that a city level clinician would have but the state-level clinician would be able to use his right throughout the state, whereas the city-level clinician would be bound to the particular city only. In other words, if a user was a state-level clinician, s/he by default would be a city-level clinician. Such a relationship was defined to be hierarchy and was handled by an assignment relationship in our model. The attribute hierarchy could be called dominance, the relation was partial order transitive, reflexive, anti-symmetric relation on attributes denoted by $\succeq$.

In a hierarchy between attributes, the contained attribute inherited all the properties of the container attribute, the relationship was represented as contained $\succeq$ container. When $a_1 \succeq a_2$ was defined in the attribute hierarchy relation, it was either $a_1 = a_2$ or $a_1 \succ a_2$ which meant attribute a_1 inherited attribute a_2. If $a_1 \not\succeq a_2$ it meant there is no inheritance relationship between these two attributes.

The spatio-temporal zones would almost always be the container of other attributes, since, the existence of other attributes might depend on the zone. The later condition in the attribute hierarchy was true for the task as well. It was because there were spatio-temporal constraints with each task. For example, the task of testing the mosquito sample could only be performed within a lab. Therefore, to represent the containment of the task with the spatio-temporal attributes in the hierarchy, the STZones would always be always the immediate container to a task, but this was not necessary for other attributes.

The user-attribute hierarchy (AH_u) and the object-attribute hierarchy (AH_o) were two variations of attribute hierarchy. The hierarchy was location and time-dependent. A jurisdiction officer could inherit the permissions of public health officers when s/he was in the field with the group. The attribute hierarchies were formally defined as follows.

1. $AH_u \subseteq UA \times UA$. AH_u is anti-symmetric, $\forall ua_1, ua_2 \in UA \bullet (ua_1, ua_2) \in AH_u \Rightarrow (ua_2, ua_1) \notin AH_u$

2. $AH_o \subseteq OA \times OA$. AH_o is also anti-symmetric, $\forall oa_1, oa_2 \in OA \bullet (oa_1, oa_2) \in AH_o \Rightarrow (oa_2, oa_1) \notin AH_o$

3. Relations AH_u and AH_o are disjoint; $AH_u \cap AH_o = \phi$

4. Spatio-temporal attributes were the containers of all attributes in the hierarchy: $\forall ua, ua^{'} \in UA, (ua, ua^{'}) \in AH_u \wedge \nexists ua^{''} \in UA \bullet (ua^{'}, ua^{''}) \in AH_u \Rightarrow ua^{'} \in STZones$, which meant the spatio-temporal attributes were the terminal (far-most) downstream entities in the attribute hierarchy, AH_u. This requirement was also is bound to the object-attribute hierarchy, AH_o.

5. Task attributes were always contained by either task or spatio-temporal attributes: $\forall ua, ua^{'} \in UA \bullet (ua, ua^{'}) \in AH_u \wedge ua \in T \Rightarrow (ua^{'} \in T) \vee (ua^{'} \in STZones)$

AH_u^+ and AH_o^+ are the transitive closure of the relations AH_u and AH_o respectively, provide a convenient way to determine whether an attribute was contained from another through hierarchy. Using AH_u as an example of the following discussion, the expression $ua_x \ AH_u^+ \ ua_y$ meant that ua_x was a contained attribute from container ua_y. For example, relation AH_u^+ was transitive on set UA since for all ua_x, ua_y, ua_z in UA, whenever $ua_x \ AH_u \ ua_y$ and $ua_y \ AH_u \ ua_z$ then $ua_x \ AH_u^+ \ ua_z$.

- $\forall ua_x, ua_y \in UA \bullet (ua_x, ua_y) \in AH_u^+ \Rightarrow \exists ua_1, ua_2, ua_3, ..., ua_n \in UA \mid n > 1 \wedge (ua_i, ua_{i+1}) \in AH_u$ *for* $i = 1, 2, 3, ..., n-1 \wedge ua_x = ua_1 \wedge ua_y = ua_n$

With the user attribute hierarchy relation, a user needs only to be assigned to the contained attribute in order to acquire all container attributes in the inheritance hierarchy, similarly for object attribute hierarchy. $Authua$ and $authoa$ were two functions that gave all attributes entitled to a user and an object, respectively, through the attribute hierarchy relations. These two functions aided in making access-control decisions and enforcing expressed policies. We said that attribute ua_i was authorized for a user u_i only if attribute ua_i was assigned to user u_i or ua_i was inherited by another contained attribute that was directly assigned to user u_i or attribute ua_i was available to user u_i through the transitive closure hierarchy, a similar concept applied to function $authoa$.

- $\forall \quad u \in U \bullet authua(u) = \{ua \in UA \mid (u, ua) \in UAAssign \vee [\exists ua \quad AH_u^+ \quad ua^{'} \wedge (u, ua^{'}) \in UAAssign]\}$

- $\forall \quad o \in O \bullet authoa(o) = \{oa \in OA \mid (o, oa) \in OAAssign \vee [\exists oa \quad AH_o^+ \quad oa^{'} \wedge (o, oa^{'}) \in OAAssign]\}$

6.2.3 Attribute Enabling-usage

Enabling attribute was the means used to express that a user was given the authorization to operate in one or more attributes. Here again, an attribute was enabled by location and time constraints; spatio-temporal constraints determined when the attributes could be enabled or disabled. When a user needed to access some resources, the enabling of the attributes took place. The enabling required the attributes to be assigned as a prerequisite. The assignment of attributes happened offline, while the enabling was performed on the fly during the time a user effectively performing some operations on protected system resources, so it was dynamic. For example, the attribute of a doctor-trainee could only be enabled in a hospital during the training period. The functionality was achieved in our model through transitive closure and containership of zonal attributes with other attributes. As mentioned before, the zonal attribute assigned to the user: directly or by inheritance, were there to represent the zones on which the contained attributes could be enabled.

Therefore, an assigned attribute was used if and only if the current zone of the user was exactly equal to the user attribute for the same user. Additionally, a workflow task was enabled for the

user to perform, if all the attributes required for the task were enabled and there was no other concurrent task attribute executed by the user. We defined $enableua$ to formalize the enabling of user attributes. Function $enableua$ mapped each user to a currently enabled attributes note that a user might not have any enabled attributes, whereby a user was mapped to an empty set. Also, function $usageu$ mapped an attribute to users who were currently using that attribute in a workflow instance.

- $enableua : U \rightarrow 2^{UA}$

- $usageu : UA \rightarrow 2^{U}$

6.2.4 Association

The association relationship $ASSOC$ allowed representing access control rights from a user attribute to an object attribute. The entities involved in an association was different in administrative accesses. Administrative access was defined by the administrative association $AASSOC$ relationship. In administrative access rights, the relationship could exist from user attribute to any other policy elements PE, where normally association was between user attribute and an object attribute. The association relationship represented that a UA would have permission to perform OPs (operation list) on object attribute OA. Similar, meaning was inferred for the administrative association. For example, if a student had permission to check out the books from the library, an association relationship would be established from user attribute student to the object attribute library books, check out being an operation in the system. Similarly, if an admin had a privilege to delete the user from the system, an association relationship would be established between the user attribute admin to the most superior attribute of the user, so the admin would have permission to delete all the available users in the system. Or if the requirement was to give permission to delete certain user but not all, the association would be done to the container that contained the desired users.

Although an association relationship was defined to be established between any user attributes to any object attributes, we restricted the association to exclude the zonal attributes in case of non-

administrative association relationship. It was because the zonal attribute mainly aided on spatio-temporal constraint and having an association relation between zonal attributes would not convey a logical relationship. For example, having an association between one zone (say z1) which acted as a container for all users in that zone and the object of that zone would just mean that every user within a zone would have access to every object in the zone which was not a very logical real life scenario. Association has been restricted from spatio-temporal attributes to prevent unnecessary access. At the same time, allowing association relationship for the task attributes provided us with extra granularity. The assignment of the task attribute requires pre-assignment of all other attributes required to perform the task, which might make the association relationship redundant. However, there might be situations where similar attributes needed to perform different actions for a different task in a workflow. The situation was handled by having an association relationship between the task type user attributes and the object attributes.

- $ASSOC \subseteq UA' \times OP \times OA'$, where $(UA' \subseteq UA) \wedge (OA' \subseteq OA) \wedge (STZones \nsubseteq (UA' \cup OA'))$

- $AASSOC \subseteq UA \times OP \times PE$.

6.2.5 Processing

A user was the active entity of a system that could cause information flow between objects or alter system state, possibly to an insecure state. Therefore, processing of an access right expressed the state of an active user who was currently requesting some access rights on objects, or in fact, a user exercising some operation on sensitive objects. These aspects of security were addressed through the $PROC$ relationship that captured user access requests or activities on system objects for a particular workflow instance. The processing relationship was also location and time-dependent, a user may be granted an access request from a certain location and during a specific time period. For example, it would be reasonable to demand that a user was authorized to report a damaged facility as part of an activity repair for a workflow instance during the daytime and at the house location. This relation was formally defined as follows:

- $PROC \subseteq U \times OP \times O$

6.2.6 Prohibitions

Prohibitions represented the inverse of the association to incorporate the existing exception into the system, the suppression of access rights. A certain user attribute was defined as the prohibition against accessing certain types of objects. The properties of an object attribute in complement set (OA^C of OA) defined the prohibitions allotted to users. For example, a specific student who had been suspended for a week would not be able to access the resources throughout the suspension period. Such a case was handled by defining a prohibition to the particular student. Prohibitions could be formed and revoked through administrative operations overriding privileges that would otherwise allow access to an object. Similar to an association, prohibitions as well was restricted to exclude the zonal attributes.

- $PROHIB \subseteq UA \times OP \times OA^C$, where OA^C is the complement set of elements not in OA ($OA^C \cap OA = \phi$), $OA^C = \{oa \in OA^C \mid oa \notin OA\}$, where $UA \notin (STZones) \wedge OA \notin STZones$

A tuple $(ua, op, oac) \in PROHIB$, where $ua \in UA$, $op \in OP$, $oac \in OA^C$, denoted that any user contained by one or more user attributes in ua could not exercise the operations in op on any object that was contained by at least one of the object attributes in oac. This formalization was useful if we wanted to define a prohibition on a specific user attribute. Prohibition on a user would prohibit all the operations executing on behalf of a user, whereas defining a prohibition on user attribute would deny access for just that attribute of the user and allow other operations to be executed, in other words, access modes allowed and denied to specific users.

6.2.7 Obligations

Obligations denoted administrative actions performed automatically upon event triggers, resulting in automatic changes to policy. Events were the means by which administrative actions were triggered automatically. An event took place each time an access operation to an object on

behalf of a user, denoted by $< u, op, o >$, executes successfully. Obligations specify an event pattern specified conditions(refereed by set $PATT$) that, if matched with an event context trigger, a set of administrative actions, also called as a response (referred by set $RESP$), to change the state of the policy. Information related to a requested access event, such as an identifier of the user, access operation, an object identifier of the triggering event was conveyed as part of the event pattern. Thereby, an event pattern, $PATT$, and a response, $RESP$ were two main components defined an obligation, which was expressed in as $\Box PATT \xrightarrow{\text{execute}} RESP$, where $\Box$ represented *whenever*.

When an event occurred, conditions for the event pattern were expressed as logical expressions that employ information from event context as well as the policy elements and relations to specify the triggering conditions. The response to a triggered event was the means by which an invocation of an administrative command occurred; every response invocation needed to pass data items from event pattern to the administrative commands. Obligations in policy were formalized as follows.

- $OBLIG \subseteq U \times PATT \times RESP$

The obligations were mostly useful to handle dynamic situations, especially with environmental conditions. For example, often times a user changed its location while exercising access rights on objects and reached a new zone (say z_2) from the old zone (say z_1), it no more acquired the attribute z_1 but z_2. To handle such changes, an administrative action to update the zone attribute for the user from z_1 to z_2 or define a prohibition on z_1 required adding an association on z_2 and a prohibition on z_1 as the response for the event of the zone change.

6.3 Constraints

A constraint was an essential aspect of a workflow authorization model and was sometimes argued to be the principle the motivation for it. Constraints provided a means of adapting access control models to the specifics of administrative and security policies in an organization. For mobile workflows, a constraint was a defined relationship among access control entities or environmental conditions (i.e., location and time attributes) related to those entities. Access control

constraints provided flexibility and granularity of expressing a security policy for performing tasks in mobile workflows. Without this flexibility and granularity, there was a greater risk that a workflow participant may be granted more access to resources that were needed because of the limited control over the types of authorizations that could be allowed.

We, therefore, discussed the most frequently mentioned constraints in the context of mobile workflows that our access control model should be capable to express and incorporate. This section discusses the following types of spatial-time constraints on attribute relationships for mobile workflows: mutually exclusive, trigger, dependency, and cardinality constraints.

6.3.1 Separation of Duty

The separation of duty (SoD) aimed to reduce the danger of fraud by distributing responsibility. In our model, SoD confirmed constraints defined on attribute assignment and enabling, that is, a user must not have two mutually exclusive attributes or capabilities at the same time. The static and dynamic SoD relations were represented using the relationships $SSoD$ and $DSoD$, which connected the conflicting attributes, which were in certain spatio-temporal zones as a top-level attribute, $SoD = \{SSoD \cup DSoD\}$, the set of all conflicting attributes. The same individual should not be assigned to conflicting attributes defined by $SSoD$ in a specific location for some duration. For example, the same user should not be assigned to a billing clerk and account receivable clerk at the same time at a specific trade corporation.

1. $SSoD \subseteq UA \times UA$
2. $DSoD \subseteq UA \times UA$, where $SSoD \cap DSoD = \phi$
3. $SoD = \{SSoD \cup DSoD\}$

The $SSoD$ and $DSoD$ relation are symmetric in contrast with role hierarchy relation.

- $\forall ua_1, ua_2 \in UA \bullet (ua_1, ua_2) \in SSoD \Rightarrow (ua_2, ua_1) \in SSoD$
- $\forall ua_1, ua_2 \in UA \bullet (ua_1, ua_2) \in DSoD \Rightarrow (ua_2, ua_1) \in DSoD$

However, the $SSoD$ constraint may be violated through a role hierarchy relation. For example, a billing supervisor may be a container attribute of the two conflicting contained attributes of a billing clerk and account clerk at the same time and in the same accounting department. Therefore, the $SSoD$ and $DSoD$ relations were symmetric in contrast with role hierarchy relation. Therefore, two conflicting attributes in $SSoD$ or $DSoD$ must not directly or indirectly have a hierarchy relation.

- $\forall ua_1, ua_2 \in UA \bullet (ua_1, ua_2) \in SSoD \Rightarrow (ua_2, ua_1) \vee (ua_1, ua_2) \notin AH_u$

- $\forall ua_1, ua_2 \in UA \bullet (ua_1, ua_2) \in DSoD \Rightarrow (ua_2, ua_1) \vee (ua_1, ua_2) \notin AH_u$

$DSoD$ was the means by which two conflicting usages of attributes could not be executed by the user in some spatio-temporal zones by the same user. For example, the simultaneous execution of cashier and cashier supervisor was forbidden during the working hours in the same store to deter such a user from committing fraud. The $DSoD$ constraints were expressed in a similar manner to the $SSoD$ constraints. The only difference was that this container prevented the execution of conflicting attributes that were connected by $DSoD$ in some top-level zones by making use of obligations as follows.

- $\forall u \in U, \exists ua_1, ua_2 \in enableua(u), (ua_1, op_1, oa_1) \wedge (ua_2, op_2, oa_2) \in ASSOC \bullet ua_1, ua_2 \in DSoD \Rightarrow (ua_1, op_1, oa_1) \vee (ua_2, op_2, oa_2) \in PROHIB$

This constraint expressed the $PROHIB$ suppression of access rights op_1 and op_2 for user u, contained by conflicting ua_1 and ua_2 attributes in $DSoD$, on any objects pertaining with oa_1 and oa_2 attributes, regardless of the access rights from user attributes to object attributes defined in relationship $ASSOC$.

The separation of duty defined in the attributes would also determine the separation of duty amongst the tasks. If a user could not be assigned two attributes then they could not be assigned to tasks that required those attributes. Therefore, we did not define the separation of tasks between the users as a different relationship but let the inheritance come into play to handle both through

already existing separation of duty in the system. Presence of static and dynamic separation of duty in the system would be able to handle the separation of tasks between the users.

6.3.2 Trigger Constraint

Trigger constraint was useful in cases where we had to perform some action instantly after an event. For the example workflow being considered throughout this model, a trigger constraint could handle a situation where a mobile user crossed the infected zone, s/he would no more be able to update the palatine record. The trigger constraint was represented as the obligation too.

- $\forall u \in U, ua \in enableua(u) \wedge z \in currentzone(u), \exists (ua, op, oa) \in ASSOC \bullet (z, ua) \notin AH_u \Rightarrow (ua, op, oa) \in PROHIB$

While a user u exercising access rights op on an object with attribute oa using some user attribute ua, if a user u moved into an invalid zone z, $(z, ua) \notin AH_u$, then the system must precluded the user from exercising the access rights defined by relationship $ASSOC$, until the user moved back to the valid zone. $(z, ua) \notin AH_u$ meant that zone z was not defined by policy, $z \notin STZones$, or there was no containment relationship between container z and ua attributes in AH_u, where $STZones \subseteq UA \cup OA$.

6.3.3 Dependency of Activities

Dependency of Activities (DoA) prevented a workflow user from exercising an activity in a workflow instance depending on performing other activities by the same or different user; in other words, DoA defined constraints on operations OP. For example, the dependency between review proposal and submit proposals activities, user u_1 must review a proposal before user u_2 submit the proposal. The DoA constraint may be location and time restrictive. Referring to an example from mobile workers who had to visit customer's house to perform ad-hoc maintenance work (e.g., fixing heating system); a mobile worker order damaged parts only if the worker had performed the inspection and reported the damage from the customer house and during the activity repair time.

- $DoA \subseteq OP \times OP$

The DoA relation was symmetric similar to role hierarchy relation, $(op_1, op_2) \in DoA$ meant that op_2 depended on op_1, but not the opposite.

- $\forall op_1, op_2 \in OP \bullet (op_1, op_2) \in DoA \Rightarrow (op_2, op_1) \notin DoA$

We defined the $COMP$ relationships that recorded events that occurred in workflow engine hosted on a stationary backend server by authenticated users in the communications (i.e., transactions) between the workflow engine and the users of that workflow; it was a data collection method that automatically captured the operation type, content object, or time of transactions made by the user from mobile devices with the workflow client application. In other words, $COMP$ defined all operations accomplished by a user on a particular object for a given workflow instance, such as $(u, op, o) \in COMP$, which meant user u performed operation op on object o. The DoA constraint governed the use of computing resources by making use of $COMP$ and $PROC$, a user attempted to access resources were intercepted and allowed or disallowed based on current DoA.

- $\forall u \in U, op_1, op_2 \in OP, (op_1, op_2) \in DoA \bullet (u, op_1, o) \notin COMP \Rightarrow (u, op_2, o) \notin PROC$

6.3.4 Cardinality Constraint

Another constraint type was cardinality constraints. Cardinality referred to defining a maximum number with respect to attribute, this was denoted by an attribute-usage cardinality constraint. One such constraint limited the maximum number of users that could be assigned to a given attribute. The numerical limitation was subject to the organizational policies. For example, a project leader or a department head would typically be limited to a single user. Thus, cardinality constraints defined a constraint on the relationship between model entities. These constraints limited the number of elements that could be connected to each other in an authorization model.

For workflows, cardinality constraints were dynamic as the numerical restriction may vary depending upon workflow instances. For example, in mWDDS workflow, the number of public health officiates that could operate in VC or SC depended on the size of the infected area, which varied from one workflow instance to another. The following formalism of attribute-usage cardinality

constraint limited the number of users that had access to attributes. Our model defined two types of cardinality for each attribute: cardinality constraints on attribute assignment (for a workflow template) and enabling (for a workflow instance).

- $assigncard : UA \rightarrow \mathbb{N}_{>0}$, $assigncard(ua)$ denoted the cardinality of attribute ua, i.e., the maximum number of users authorized for that attribute in a workflow template.

- $usagecard : UA \rightarrow \mathbb{N}_{>0}$, $usagecard(ua)$ limited the maximum number of used attributes by a collaborative user per a workflow instance. The usage attribute carnality must not exceed the assignment cardinality, $\forall ua \in UA \bullet |usagecard(ua)| \leq |assigncard(ua)|$.

- The attribute cardinality must be specified for every attribute: $\forall ua \in UA \bullet assigncard(ua) \neq \phi \wedge usagecard(ua) \neq \phi$

The number of authorized users for any attribute in the assignment or usage relationship did not exceed the cardinality of that attribute. Formally:

- Cardinality Constrains on Attribute-Assignment: $\forall ua \in UA \bullet |assignu(ua)| \leq assigncard(ua)$

- Cardinality Constrains on Attribute-Activation: $\forall ua \in UA \bullet |usageu(ua)| \leq usagecard(ua)$

6.4 Check Access

After having the policy in place, a user was granted or denied access to an object based on the attributes of the user and object. This operation was location and time-dependent, it checked whether a user was authorized to perform some operation on an object during a certain time and from a certain location. A user was allowed to fire a missile if he was assigned the role of a top-secret commander and he was in the controller room of the missile during a severe crisis period. Thus, a user could access an object in a certain zone if that user had operated on an attribute with appropriate access right in relationship $ASSOC$. Check access function $checkAccess$ was formalized as follows.

- $\forall u \in U, op \in OP, o \in O \bullet checkAccess(u, op, o) \Rightarrow \{\text{“}GRANTED\text{”}, \text{“}DENIED\text{”}\}$

With respect to the assignment and association relationship created between the entities, a request was granted if there existed an association and no obligation or prohibition for it.

- $\forall u \in U, op \in OP, o \in O \bullet checkAccess(u, op, o) == \text{“}GRANTED\text{”} \Rightarrow \exists (ua, op, oa) \in ASSOC \wedge (ua, op, oa) \notin (PROHIB \cup OBLIG)$, where $(u, ua) \in UAAssign \wedge (o, oa) \in OAAssign$.

6.4.1 Relation derivation

We could use existing relations to derive another relation in order to process access requests in accordance with policy. From the association relationship, we could see that we did not assign user u to operation op on object o, such as $< u, op, o >$, however, we were checking for the existence of $(ua, op, oa) \in ASSOC$ during check access, where $(u, ua) \in UAAssign \wedge (o, oa) \in OAAssign$. Checking access request or exercising access rights was done by derivation from the assignment relationship, such as from $ASSOC$ and

- $\forall u \in U, ua \in UA, o \in O, oa \in OA, op \in OP \bullet (ua, op, oa) \in ASSOC \wedge (u, ua) \in UAAssign \wedge (o, oa) \in OAAssign \Rightarrow (u, op, o) \in PROC$

Similar derivation, from the attribute hierarchy relationships that existed between the attribute and the policy elements itself, we could derive the associations between policy elements. Derivation from attribute hierarchy to the association was formalized as follows.

- $\forall ua_1, ua_2 \in UA, oa \in OA \bullet ua_1 \succeq ua_2 \wedge (ua_1, op, oa) \in ASSOC \Rightarrow (ua_2, op, oa) \in ASSOC$

Above mentioned derivations verifed that the query submitted $< u, op, o >$ belonged or did not belong to the set ASSOC. Similarly, relation derivation could be used to verify if $< u, op, o >$ belonged or did not belong to the set $(PROHIB \cup OBLIG)$. If it was found that the query belonged to the ASSOC and did not belong to either of PROHIB or OBLIG than we could say

that the attributes required for the operation on the object existed for the user. However, due to the spatio-temporal constraints present in our model, the existence of the attribute was not enough to have access, but the attribute enabling and disabling decided the access. The process would be followed by the enabling, and upon successful enabling, the user would be granted the permission or denied on failure.

- enbaling()=$\{success if\, currentzone(u) \in AH_u^+\} \Rightarrow AccessGranted, else\, AccessDenied$

After the enabling of the attributes and confirmation of access rights, the user would be able to perform a task for the workflow. Through PROC and COMP function, we could keep track of task being completed and, thus, disable the task for the users after completion, thereby, revoking the permissions associated.

6.5 Authorization Graph

An authorization graph denoted by AG was the visual representation of the policy structure. Authorization graph was a set of items connected by edges, each item was called a vertex or node, edges represent the relations between items. The AG graph did not allow self-loops, adjacency was irreflexive. The authorization graph depicted all the policy elements and their relationship including prohibition constraints. The dynamic constraints, like obligation and SoD, would be mentioned through some other measures rather than the edges or nodes of the graph. Policy elements in the model were represented as the nodes in the AG graph. In the actual graph database, the nodes could have their own labels and, therefore, distinguishable.

Here, the graph was marked to distinguish between different kinds of element the node was representing. Similarly, all the relationship would be represented as the directed edges between the nodes. The assignment relationship was represented as the directed edge to the container. The direction also showed that the contained attribute was inheriting the properties from the container. The association would be represented as the (dotted) undirected edges from one policy element to the other. The association relationship would not be directed, this in turn, aided on having two

types of interpretation of the association relationship, i.e., from the perspective of the user or the object.

The association relationship had operations labeled onto it representing the set of operations that could be performed due to the existence of that edge between the components (nodes). The prohibitions relationship would have all the inverse behavior of the association in the graph, i.e., would be represented as a (dotted) undirected edge and have prohibition of operations labeled into it in order to express the fact that both of prohibition and association had opposite functional behavior.

Definition 3. [Authorization Graph AG**]** *An authorization graph* AG *is a pair* $< V, E >$*, where* V *is a set of vertices, and* E *is a set of edges between the vertices or nodes.*

Authorization graph had two types of edges, $E^{'}$ directed edges and $E^{''}$ (dotted) undirected edges augmented with labels, L was the set of labels that denoted to the type of association operation, prohibition operations, or SoD constraints between two adjacent vertices, thus, $E = \{E^{'} \bigcup E^{''}\}$. The set of $E^{'}$ directed edges between the vertices was asymmetric, $E^{'} \subseteq \{(u, v)|u, v \in V \wedge u \neq v\}$, That is, each edge could be followed from one “outgoing" vertex, such as u, to another “target" vertex, such as v. The set of $E^{''}$ (dotted) undirected edges was symmetric, $E^{''} \subseteq \{\{u, l, v\}|u, v \in V \wedge l \in L \wedge u \neq v\}$, that is, each edge connected two vertices with a labels representing the type of association/constraint between vertices. When $e^{''} \in E^{''}$ this meant that $e = (ua, op_1, oa)$ such that $e^{''}$ was a (dotted) undirected edge between two vertices such as ua and oa that had an association operation op_1.

Thus, the developed authorization graph allowed us to see the flow of the properties between the policy elements through the relationship defined in the model. Similarly, it allowed a better understanding of the hierarchy and inheritance. Authorization graph also helped in the visualization of restriction applied by giving permission into a different level of inheritance. Ultimately, AG supported the understanding and justification of the containership between the elements and its significance on mapping the access control to a real-world scenario..

Table 6.1: Input Data Structure for authorization graph

N	Data Set	Description
1	U	Set of users in the system
2	UA	Set of user attributes in the system
3	O	Set of objects in the system
4	OA	Set of object attributes in the system
6	PC	Set of policy classes in the system
7	$A^{'}$	Set of all relationships in the system: $A^{'} = \{A \cup AH_u \cup AH_o\}$
8	$ASSOC$	Set of associations between attributes in the system
9	$PROHIB$	Set of prohibitions in the system, inverse of $ASSOC$
10	SoD	Set of SoD conflicting constraints between attributes in the system

We had devised an algorithm to develop the authorization graph AG, which took the policy information as the input and returned the authorization graph AG. Table 6.1 shows the Data Structure required for the AG algorithm, 1 was the pseudo-algorithm for the construction of the authorization graph.

Let us explain the algorithm through an example. We showed an example of policy information in the form provided in Table 6.1. Algorithm 1 took the policy information as an input data structure and generated the authorization graph AG as shown in Figure 6.2.

Lines $3 - 10$ of algorithm 1 explore elements in sets U, O, UA, OA, and PC and construct nodes marked by the elements in those sets. The algorithm traversed through the relationships in set $A^{'}$ in lines $11 - 18$ to create directed edges into $E^{'}$ between nodes in V. Undirected (dotted) edges between nodes that represented associations, prohibition was created into $E^{''}$ in lines $19 - 35$. The SOD constraints were noted into a file in line $36 - 43$. Note in lines $27 - 34$ that label l was concatenated with annotation PR to express on the edge the suspension of the operation represented by label l. Lines 44, 45, 46, and 47 form set E of edges, incorporated sets V and E in graph AG, rendered graph AG, and returned the object AG graph, respectively.

A Real-world mobile Workflow System: mWDDSS

The DDSS mentioned in Chapter 3 had certain job functions. Here we present some of them and their representation of in an authorization graph following our model.

Algorithm 1: Constructing Authorization Graph

input : Data Structure
output : Authorization graph "AG''

$AG = \phi \rightarrow V = \phi \wedge E = \phi$; // $E^{'} = \phi \wedge E^{''} = \phi$

foreach $(u \in Users; o \in Objects; ua \in UA; oa \in OA; pc \in PC)$ **do**
    $V = V \cup \{u\}$; // adds marked user u node to set V
    $V = V \cup \{o\}$; // adds marked object o node to set V
    $V = V \cup \{ua\}$; // adds marked user attribute ua node to set V
    $V = V \cup \{oa\}$; // adds marked object attribute oa node to set V
    $V = V \cup \{pc\}$; // adds marked policy class pc node to set V
end
if $A^{'} \neq \phi$ **then**
    foreach $(x, y) \in A^{'}$ **do**
        if $(x \in V \wedge y \in V)$ **then**
            // check whether x and y are nodes in V
            $E^{'} = E^{'} \cup \{(x, y)\}$;// adds a directed edge from node x to node y
        end
    end
end
if $ASSOC \neq \phi$ **then**
    foreach $(x, l, y) \in ASSOC$ **do**
        if $(x \in V \wedge y \in V)$ **then**
            // check whether x and y are nodes in V
            $E^{''} = E^{''} \cup \{x, l, y\}$;// adds an undirected edge between node x and y with label l, such as op
        end
    end
end
if $PROHIB \neq \phi$ **then**
    foreach $(x, l, y) \in PROHIB$ **do**
        if $(x \in V \wedge y \in V)$ **then**
            // check whether x and y are nodes in V
            $l^{'} =$ "PR" $+ l$; // concat lable l with word "PR" to indicate the suppression of $ASSOC$
            $E^{''} = E^{''} \cup \{x, l^{'}, y\}$;// adds an undirected edge between node x and y with label $l^{'}$, such as $PRop$
        end
    end
end
if $SoD \neq \phi$ **then**
    foreach $(x, y) \in SoD$ **do**
        if $(x \in V \wedge y \in V)$ **then**
            // check whether x and y are nodes in V
            write into SOD.txt file x, y
        end
    end
end
$E = \{E^{'} \bigcup E^{''}\}$;
$AG \leftarrow < V, E >$; // creates authorization graph for vertices V and edges E
$display(AG)$; // renders the AG graph
$return(AG)$;

Table 6.2: Input Data Structure for Example Authorization Graph

Set U	$\{u_1, u_2, u_3, u_4\}$
Set UA	$\{ua_1, ua_2, ua_3, ua_4\}$
Set O	$\{o_1, o_2, o_3\}$
Set OA	$\{oa_1, oa_2, oa_3\}$
Set PC	$\{pc_1\}$
Relation $A^{'}$	$\{(u_1, ua_1), (u_2, ua_2), (ua_1, ua_3), (u_4, ua_4), (u_4, ua_1), (ua_4, ua_3), (ua_2, ua_3), (u_3, ua_3), (ua_3, pc_1), \quad (o_1, oa_1), (o_2, oa_1), (o_3, oa_2), (oa_1, oa_3), (oa_2, oa_3), (oa_3, pc_1)\}$
Relation $ASSOC$	$\{(ua_2, op_1, oa_2), (ua_3, op_2, oa_3), (ua_3, op_3, oa_3), (ua_4, op_3, oa_2)\}$
Relation $PROHIB$	$\{(ua_1, PRop_3, oa_1)\}$
Relation SoD	$\{(ua_4, ua_1)\}$

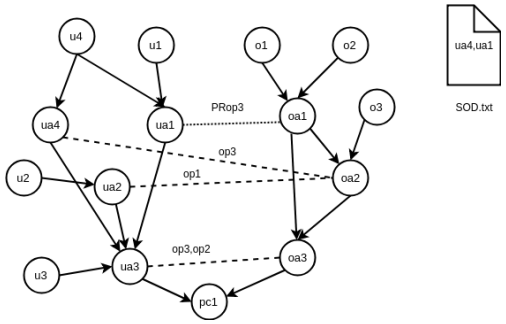


Figure 6.2: Example of Authorization Graph from Algorithm

Personal managers were responsible for assigning attributes, tasks, and privileges to users (based on the attributes). Clinicians reviewed patient personal information (e.g., names, gender, date of birth), premises of the patient (e.g., residence, and optionally work or school), past hospitalization and treatment information (e.g., clinic, physicians, disease), and clinical findings (e.g., presence/absence of fever, nausea, or headache). Laboratory technicians collected laboratory test data including the type of samples, method(s) used to test the samples, framework for interpreting test results, and interpreted results. Epidemiologists accessed patient and laboratory test information with regard to evaluating and changing health safety standards and programs. Vector control team members sprayed houses in infected areas. Vector surveillance team members performed mosquito collection and testing tasks intended for developing insecticide resistance methods. Vector control and surveillance members were provided with needed materials to use for their tasks. Such material allocation was done by material managers at the state or city level. Material managers were also responsible for updating the local materials inventories. Vector manager designated the tasks that must be performed by vector control and surveillance teams.

Health care professionals were only allowed to access their patients' records in the areas where the dengue case occurred and during the course of dengue. Moreover, to prevent fraudulent entries, the vector control and surveillance teams must perform their tasks in the designated places and at specified times. Due to the significance hierarchy between the attributes (for example: A user having an attribute State Hospital Clinician would also have City Hospital Clinician City resided in a city within the state) the access control model captured it in terms of containership between attributes. To bind the actions to certain zones, zones were considered an attribute of this system.

Below we present some fictional user and their access policy with respect to their attributes including workflow task. The policy is graphically represented in Figure 6.3.

Users: $\{Alice(u1), Bob(u2), Dave(u3), Shan(u4), Tim(u5), Shelly(u6), Phil(u7), Lara(u8), Evan(u9)\}$

User and Object attributes (Spatio-temporal zones): $\{z_0 :< Colorado, DayTime >,$
$z_1 :< HeadOffice, DayTime >, z_2 :< FortCollins, DayTime >,$
$z_3 :< HouseAddress, DayTime >, z_4 :< InfectedArea, DayTime >,$
$z_5 :< Lab, DayTime >, z_6 :< AnyWhere, Anytime >\}$
$Daytime = [8am - 5pm], Anytime = [11:59am - 11:59pm]$ *and zone* z_6 *is the zone that contains all other zones.*

User Attributes: $\{Manager(M), StateOfficer(SO), Jurisdiction\ Officer(JO),$
$LabTechnician(LT), VectorControlTeamMember\ (VCT),$
$VectorSurveillanceTeamMember(VST), PublicHealthOfficer(PHO)\}$

Objects: $\{UserName(o1), UserAddress(o2), ThresholdTime\ (o3), ThresholdValue(o4),$
$Equipment(o5), Equipment\ Amount(o6), InfectionRate(o7), Infectedarea(o8),$
$SampleId(o9), MethodUsed(o10), AssignedTask(o11), AssignedPersonel(o12)\}$

Object Attributes: $\{UserData(UD), EnvironmentalData\ (ED), WareHouseData(WHD),$
$VectorData(VD), LabTechnicianData(LTD)\}$

Task: $\{FormJurisdiction(FJ), CheckThreshold(CT), Activate\ Response(AR),$
$ActivateVCandVSTeam(AVCST), PerformTest\ (PT), SprayHouses(SH),$
$CollectMosquito(CM), Release\ Material(RM)\}$

Operations: $\{Read/Write/Update/UserData(op1), Read/Write\ /Update/$
$LabTechnicianData(op2), ReadVectorData(op3), Read/Write/UpdateEnvironmental$
$Data(op4), Read\ LabTechnicianData(op5), Read/Write/Update\ WareHouseData(op6),$
$Read/Write/UpdateVectorData(op7)\}$

Policy Classes: $\{DDSSAccess\}$

User-Attribute Assignment:$\{(u1, M), (u2, SO), (u3, JO), (u4, VCT), (u5, VCT), (u6, VCT), (u7, VCT), (u8, VCT), (u4, VST), (u5, VsT), (u6, VST), (u7, VST), (u8, VST), (u9, LT)\}$

Object-Attribute Assignment:$\{(o1, UD), (o2, UD), (o3, ED), (o4, ED), (o5, WHD), (o6, WHD), (o7, ED), (o8, ED), (o9, LTD), (o10, LTD), (o11, VD), (o12, VD)\}$

Attribute Hierarchy: $\{(M, FJ), (M, CT), (M, AR), (SO, RM), (JO, AVCST), (VCT, PHO), (VST, PHO), (VCT, SH), (VST, CM), (LT, PT), (FJ, z_1), (CT, z_1), (AR, z_1), (RM, z_0), (AVCST, z_2), (PHO, z_2), (SH, z_3), (CM, z_4), (PT, z_5), (UD, z_1), (ED, z_1), (ED, z_4), (WHD, z_0), (VD, z_2), (LTD, z_5), (z_1, z_6), (z_0, z_6), (z_5, z_6), (z_3, z_6), (z_2, z_6), (z_4, z_6)\}$

UserAttribute-PolicyClass-Assignments:$\{(z_6, DDSSAccess)\}$

ObjectAttribute-PolicyClass-Assignments: $\{(z_6, DDSSAccess)\}$

Associations:$\{(LT, op2, LTD), (SO, op6, WHD), (JO, op3, VD), (VST, op4, ED), (VCT, op5, LTD), (M, op4, ED), (AR, op1, UD), (FJ, op1, UD), (FJ, op7, VD)\}$

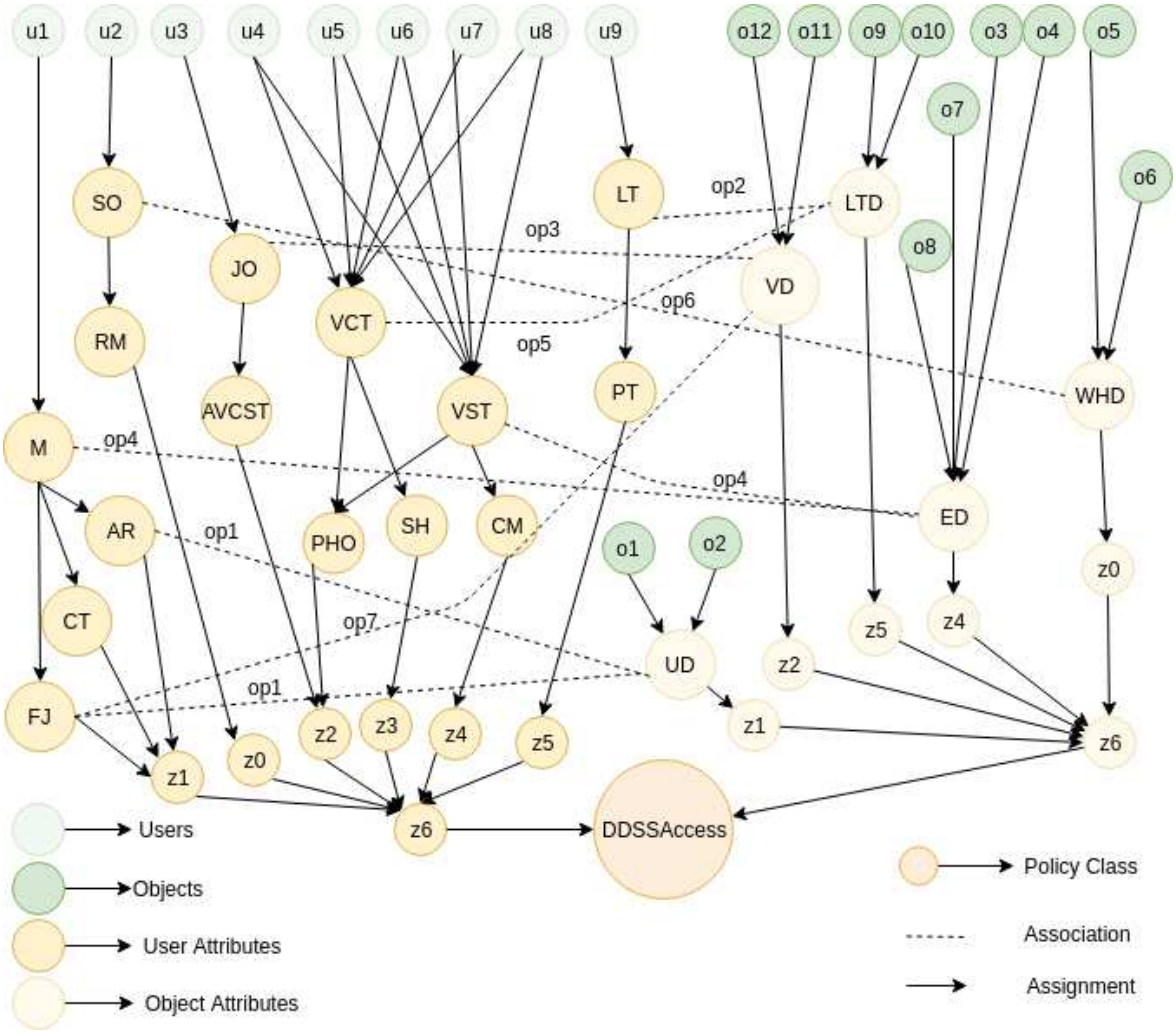


Figure 6.3: Authorization graph for DDSS

Chapter 7

Analysis of Workflow with Authorization Constraints

The proposed model in Chapter 6 was well-suited for many mobile workflow policies, however, by itself, did not guarantee that a system remained secure in all situations. The model supported the specification of various spatio-temporal features that might interact in a subtle way resulting in inconsistencies and conflicts. For example, incorrect spatio-temporal constraints or task control flows may prevent a workflow participant from invoking its job. Therefore, an analysis approach of the workflow model, such as the one introduced in Chapter 6, was a compulsory task needed to ensure that inconsistencies or security violations did not occur when a given application was using our model. The policy flaws such as incompleteness and inconsistency were hard to be detected by manual policy inspection. In addition, when a policy had numerous rules, manual inspection was tedious and error-prone.

Automaton verification such as using model checking techniques was an important task needed to detect policy flaws. The automaton checker would disclose any event that caused the system transition to an invalid state, which did not conform with underlying policy rules. For example, deadlocks may exist in a workflow in case no participant was available to perform a task, while some tasks were waiting for the completion of that task in the first place. Such a problem arose due to an error in the business rules specification and task authorizations. In our proposed mobile workflow model, the attribute-based access control model governed user access to tasks, data, and services. That is, tasks, objects, and applications were associated with attributes that defined where and when a user could have access.

Thus, it was important to analyze the interactions of the dependencies and authorization constraints and verify properties of the system through an automated verification tool. In our case, we used Timed Coloured Petri Nets (TCPN). TCPN was widely utilized to perform an exhaustive search in model space in order to verify security properties of concurrent and distributed actions. For the complete reference on the TCPN, please refer to [42]. The analysis step will detected

conflicts between tasks, improper execution of tasks by unauthorized users, violation of dependencies, and deadlocks. Towards this end, we advocated the use of the TCPN to develop and analyze the CPN model representing mobile workflow with authorization constraints. TCPN had toolbox support for graphically representing the workflow model and performing simulation and formal analysis [43]. It has been used successfully for the formal analysis of real-time concurrent systems, such as our mobile workflow model.

7.1 Background: Timed Colored Petri Nets(TCPN)

A classical Petri-nets (PN) were widely used for describing and studying systems that were characterized as being concurrent, asynchronous, distributed, parallel, and non-deterministic [1]. A classical Petri net was a bipartite graph and mathematical modeling tool. Petri nets have had an intuitive graphical representation that made it easy to see the basic structure of a complex PN model, i.e., understand how the individual processes interact with each other. It consisted of three basic components: places (that usually represent the passive elements of the system), transitions (that represent the active elements), and arcs that connect them. Two types of arcs existed: input arcs connect places with transitions, while output arcs connect transitions with places.

Places could contain tokens. The current state of the modeled system (the marking) was given by the distribution of tokens in each place. Transitions were active components that model activities, which could occur (the transition fires), thus, changing the state of the system (the marking of the Petri net). When transitions were enabled, they were allowed to fire, which meant that all the preconditions for the activity must be fulfilled (there were enough tokens available in the input places). Whenever a transition fired, it removed tokens from its input places and added some at all of its output places. The number of tokens removed and added to output places depended on the cardinality of each arc. Furthermore, transitions and arcs could be augmented with guards and expressions respectively.

Colored Petri-Net extended Petri Nets by combining the strengths of ordinary Petri Nets with the strengths of a high-level programming language to provide more expressiveness. CPN also

had a formal, mathematical representation with a well-defined syntax and semantics. Basic Petri nets provided the primitives for process interaction, while the programming language provided the primitives for the definition of data types and the manipulations of data values. Colored Petri Nets (CPN) allowed tokens to be associated with colors, i.e., data types through color sets.

The Timed Colored Petri-Nets(TCPN) extended furthermore by allowing the inclusion of the time in the model. The color sets were timed by appending "timed" keyword, while declaring the colored sets. It further made it possible to assign an integer number to each timed token and would represent the time in which those tokens would be available. The TCPN introduced a feature to incorporate with a transition either a constant or functional time, which represented the amount of time taken for the completion of an activity.

TCPNs allowed different tools for simulation of the model. It could be simulated interactively observing effects on each step or automatically to some final state after a specified number of steps. TCPNs also offered more formal verification methods, known as State Space analysis. In this way, it was possible to prove, in the mathematical sense of the world, that a system had a certain set of behavioral properties. A state space analysis represented all possible executions of the model being analyzed. This made it possible to verify systems, i.e., prove that different behavioral properties were present or absent in a model.

As our model had a temporal component associated with it, the Timed Colored Petri Net allowed us to introduce the effect during the simulation and verification.

7.2 Formal Definition: TCPN

The Timed Colored Petri Nets formally were composed of ten tuples:

$$TCPN = (Tm, Cs, P, T, A, N, C, G, E, IN, M)$$

satisfying the following requirements:

(i) Tm was a set of real integer that represented time. Tm appended with CS to form a timed color set. Tm also represented the global time clock for TCPN. Every page in a net would have access to the global clock and changed the state as permitted by the global clock. If no transaction was active for a certain time (say tm_1), the global clock was incremented to the next time a transaction was active. Tm could also be a transaction inscription for which it would increase the time for the token by the inscribed value, in other words, would introduce a time delay.

(ii) CS was a finite set of non-empty types called color sets (also referred as colset). They could be timed or untimed with at least one of the color set in the system being timed, i.e., $n(CS) * Timed > 0$

(iii) P was a finite set of places

(iv) T was a finite set of transitions

(v) $A = Ia \cup Oa \cup IOa$ was a finite set of arcs such that $P \bigcap T = P \bigcap A = T \bigcap A = \oslash$; where Ia is a set of input arcs, Oa is a set of output arcs and IOa is a set of input output arcs.

(vi) $N : A \rightarrow P \times T \bigcup T \times P$ is a node function maps each arc with place and transition

(vii) $C : P \rightarrow CS$ was a color function that mapped a place to a color type

(viii) G was a guard function. It was defined from T into expressions such that $\forall t \in T : [(Type(G(t)) = Bool) \wedge Type(Var(G(t))) \subseteq CS]$

(ix) E was an arc function. It is defined from A into expressions such that $\forall a \in A : [(Type(E(a)) = C(p(s))_{ms}) \wedge Type(Var(E(a))) \subseteq CS]$, where $p(a)$ was the place of $N(a)$ and C_{MS} denoted the set of all multi-sets over C. More precisely, this function associated each arc with an expression, which must be of type $C(p)_{ms}$ where P was an input/output place belonging to a given arc. Furthermore, all variables in such expressions must also be of place color $C(p)$ type. Evaluation of the arc expressions for a given transition let us know what token was to be taken from the transitions input place as well as what token was to be placed into the output place.

(x) IN was an initialization function. It was defined from P into expressions such that $\forall p \in P : [(Type(IN(p)) = C(p(s))_{ms}) \wedge Var(IN(p)) = \oslash]$. It mapped each place into an expression, which must be of color type $C(p)_{ms}$. Each place was assigned a multi-set of tokens, and relevant color conforms to the color of the place.

To enable transitions in the TCPN model, we must obtain initial marking by evaluating the initialization functions. Arc expressions specified a collection of tokens that were added to (or removed from) the places. A transition could be enabled when each of its input places contained the multi-set specified by the input arc inscription and the guard transition evaluated to be true. The tokens inscribed to flow to the transaction, if time, would not be available until the global time was greater than or equal to the token time. Upon firing an enabled transition, the tokens were removed from the input places and probably added to the output places of the occurring transitions. The number and color of the tokens were determined by the output arc expressions, evaluated for the occurring bindings. The time delay, if any in the transitions, would be imposed on the tokens being passed away from the transitions.

In the following, we constructed the transformed TCPN model of our task-attribute based access control policy for mobile workflow systems based on the above elements of TCPN.

7.3 TCPN based model for mobile Workflow System

Using TCPN allowed us to model the behavior of the task control flows and perform the model simulation. As discussed in Chapter 6, our access control model for the real-time workflow system (Dengue Decision Support System) was based on user, object, and their attributes. The attributes in addition to their roles and hierarchies also included the location and time (known as STZones as single units) and workflow tasks that they were allowed to perform after verification of all other required attributes. Based on these, our TCPN model simply represented user as the timed tokens that were moved to places, when a transition was enabled. A sample example is shown in Figure 7.1. The relationship, flow, and constraints were preserved in the model by following a constant template, which is described below in detail.

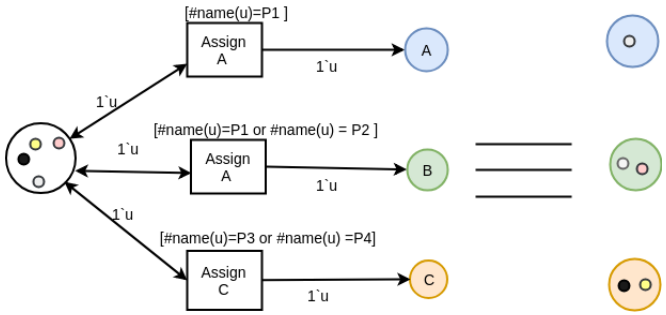


Figure 7.1: Example TCPN

Relevant TCPN elements

- Color set(CS)
  - colset USER = record name:STRING timed;
  - colset threshold = INT timed;
  - colset StartEnd = bool;
  - colset INT = int;

  The tokens matching the above color sets were:

  USER token : $< u >$ ::colset USER

  As we could see that the USER was a timed colset, every user token was timed. The time factor to the user token would constrain a user token to be available before time.

  Threshold token :$< th >$ :: colset threshold

  The threshold token represented the input from the environment. In particular, it specified the infection rate of the area being analyzed and, thus, the workflow should start operating if the input was greater than a standard value.

  The timed feature of threshold type token helped us to model the real-time scenario where the threshold value was updated every interval of time. And once the input value was higher

than the threshold, the response was enabled, but not multiple times for each input greater than the threshold.

StartEnd : $< se >$:: colset StartEnd

Counter: $< c >$:: colset INT

- Places(P)
  - User Attribute (UA): The place represented the user attribute of the users and held a user token. In real life scenarios, we could have these attributes being assigned to different users in the system. In which case, our model would represent the complete phenomena as shown in the left side of the Figure 7.1. However, with the guard functions in each transition, it was a static phenomenon. Also, since we were modeling a task-attribute based access control, our assumption would be that a user possessed certain attributes and used any combination to exercise the required task permissions. Therefore, in our model, we simplified the model by representing a user with an attribute as a token in the user attribute place. In this way, we reduced the complexity and had an efficient representation of our model at the same time.
  - Zone Attribute (ZA): The place represented the physical locations. The presence of a token in this place signified that the mobile user was now in the required STZone place and, thereby, could enable the user attribute from which it had arrived into the place. It could be further sent to the task attribute since it had obtained all the attribute necessary for the task by fulfilling the spatial-temporal constraints necessary.
  - WorkFlow Task Attribute (TA): The place represented the individual task of the workflow and held a timed user token. Presence of a token in this place, at any state of the TCPN model, would signify that the user or multiset of users had all the attributes

required to perform the task and, thus, were eligible to perform the task. This state was considered to be the processing state where the users were into the task.

  - Start and Sink (SS): These places represented the start and sink of the workflow. They held boolean type token. The model consisted of a single token in the start type place, which was moved by a transition and enabled the operation on the workflow. Our goal was to have one or more boolean type token in the Sink type place. The number of the token to be present depended on the number of the path that could be followed to the end and the number of user tokens required to perform each end task.
  - Counter(C): These places were present in the system to hold a single INT type token that got updated every time a transaction was fired. The counter token was there to track the number of tokens supplied to the successor TA or ZA or UA where ever necessary.

- Transitions (Ts)

  - Assign (Assn): Since our model had an assignment for users to attributes, the assign transition would be responsible for assigning a task when all the task to a user who possessed all the required attribute and inactive state. The assign type transition moved the tokens from ZA to TA representing the user being assigned the task.
  - Enable (Enb): Since, enabling depended on spatio-temporal constraint, enabling transitions to move the user tokens from UA to ZA such that token now enabled the transition that would move the token to the task. In other words, the user got the necessary zonal attribute to exercise the user attributes and was ready to be assigned to the task.

- Arc, Arc Expression and Guard:

  - Our TCPN model had input and output arcs with mostly user token as expressions, which made the arcs simple. In very few cases, the arcs were required to be both input and output. Such cases were represented as two arcs (input and output) joining the same elements. The guard functions were given to some transaction to specify certain

conditions at which they were to be enabled. For example, infection should be greater than a threshold or only some user was allowed to fire some transaction.

In the template TCPN model given in Figure 7.1, the circles represented the places. Places in the TCPN template held tokens of type colset USER. The small filled tokens represented the tokens of user type. The color coding was done to represent the same user being assigned to different attribute or vice versa. As a result, there could be multiple tokens of the same type in the system. Also, in the example, there were four input tokens but has five tokens in the system in total because one user was being assigned to two different user attributes. The transitions were represented in the rectangles. The arcs directed towards the transactions were the input arc and directed away from the transitions were the output arcs. The inscription "1`u" was the arc expression that represented 1 unit of user token was being moved, where "u" was a USER type variable. The inscription above the transition enclosed within in the square braces was the guard function which, in this example, stated the name tuple of record "u" should be as mentioned for the transition to fire or only the token with such behavior could fire the transition.

7.3.1 Workflow Control Flow in TCPN

The workflow task in our model was ordered in six different ways. Each order is described below for TCPN modeling and shown in Figure 7.2

A. Sequential (S): The sequential order was when one task was performed after another. If T1 and T2 were sequential such that T2 came after T1, TCPN modeled it by placing a transition between them such that the transition placed a token from T1 to T2.

B. Conditional Repeats (CR): The conditional repeats were modeled in TCPN by a guard function in the transitions. If T1 needed to be repeated conditionally then a transition was placed before T1 that took input for conditions and placed a token to T1 whenever satisfied.

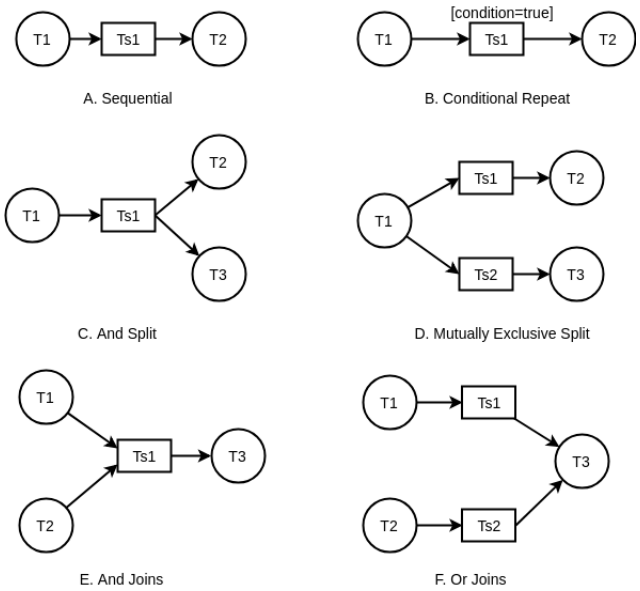


Figure 7.2: ControlFlow in TCPN model

C. And-Split (AS): In the case of and-split, all the branches from a task should be performed. If T1 was splitting to two independent task T2 and T3, a transition was placed in between the branching task (T1) and branches (T2, T3) task and which received one input token and multiplies to all the branches.

D. Mutually Exclusive-Split (EX): In the case of mutually exclusive-split, one of the branches from a task should be executed. If task T1 splits to two task T2 and T3, a transition was placed in between the branching task (T1) and each of the branches (T2, T3) task, such that, each of the branches required a different token to fire the transition and, therefore, received a token.

E. And-Joins (AJ): The and-join required completion of all the merging task. If T1 and T2 joined to T3, to achieve this a single transition was placed that received tokens from all the merging tasks (T1, T2) such that transition was fired only when all the merging transitions were completed.

F. Or-Joins (OJ): The Or-joins allowed the same task to be performed from a different path but not all the merging task needed to be completed. Therefore, if T1 and T2 merged to T3, a

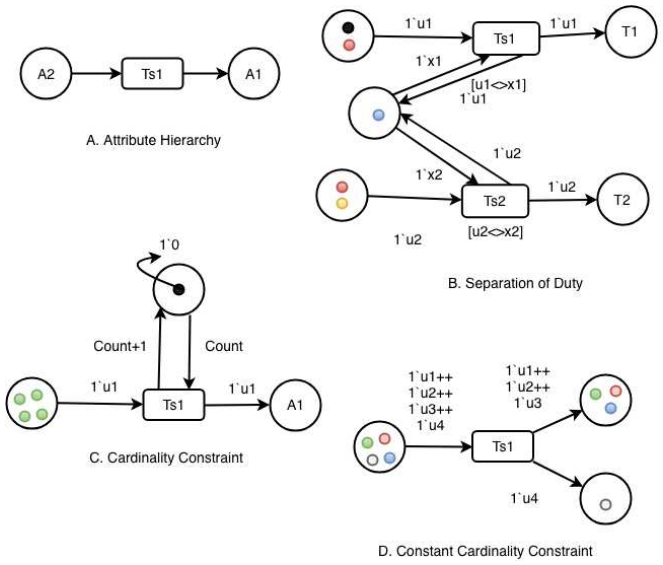


Figure 7.3: Special cases in TCPN model

transition was placed after each merging task (T1, T2).

7.3.2 Special Cases

Apart from, the elements and their flow, TCPN model also needed to incorporate special cases like attribute hierarchy, separation of duty in task, and cardinality constraints.

A. Attribute Hierarchy: Attribute hierarchy would allow a senior attribute to exercise the privilege given to the junior level attribute. The hierarchy was represented in TCPN by placing a transition between two attributes. If A1 was a senior attribute and should be able to exercise all the permissions from A2 then, user token would be on attribute A1 only through A2, such before having used A1, it would use A2, thus, having both the attributes as shown in Figure 7.3.

B. Separation of Duty in Tasks: For separation of duty in the task, we followed the guidelines provided by [44] and their model is shown in Figure 7.3 with a small modification since our model did not have a start and end for the task.

C. Cardinality Constraints: Cardinality constraints controlled the number of tokens that could be present in a place (attributes or task). It essentially meant that some attributes or task should be assigned to at most a given number of users and no more. To implement this in TCPN, we added a counter place to the transition that transferred tokens to the designated place and also guarded specifying the limit of the counter tokens as shown in Figure 7.3. However, in different places, where more tokens were available at once, the cardinality constraints could be represented by the arc inscription as well. If an arc was inscribed with the multiset of variables, it would refer to the cardinality constraints exactly equal to the specified number as shown in Figure 7.3 d. However, this could be done if some tokens were available in the system, or it might cause deadlock, otherwise.

7.3.3 Hierarchical Model

The TCPN net developed by following the template described above was a very huge net, due to which there was an obstruction on viewing and understanding the flow of the model. As a solution to this, CPNs supported a mechanism for construction of large system models in a hierarchical manner equivalent to the complete and complex model by introducing the concept of subpage transition, ports, and sockets as described on [45]. Thus, developed hierarchical model followed the structure as shown in Figure 7.4. The hierarchy mechanism of modules allowed us to model different levels of abstraction that were inherent in workflows. Hierarchy divided up and represented a complex system as several connected sub-models. CPN Tools allowed both a bottom-up approach and a top-down approach for instantiating hierarchical net models. Thus, the graphical representation of the hierarchical model made it easy to see the whole structure of the workflow model and understand the individual sub-components interaction.

Each of the units shown in Figure 7.4 are separate TCPN pages with unique properties enclosed within. The DDSS was the main page that represented a higher level or abstraction level of the DDSS system, i.e., a workflow was started where a manager monitored the infection rate of the jurisdiction, form the jurisdiction group, and upon receiving an infection rate greater than the predetermined threshold, the response was enabled. The workflow ended upon successful execution

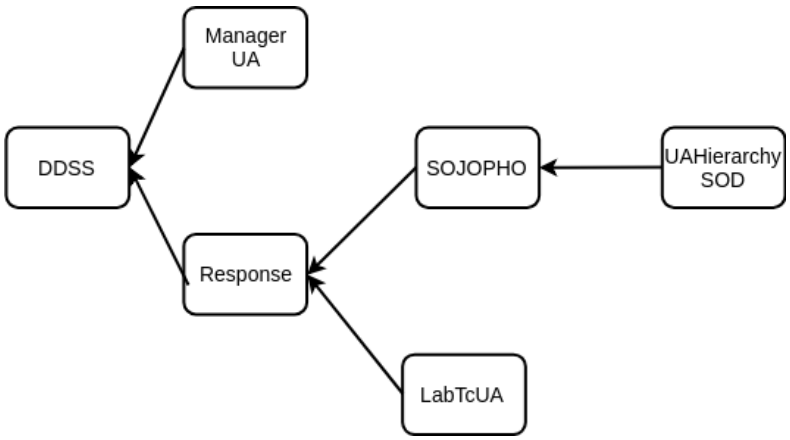


Figure 7.4: Hierarchical Structure for TCPN Model

of the response. The response was a separate model, which maintained the control flow of the task. The ManagerUA was responsible for verifying that the Manager had gotten the spatio-temporal attribute necessary to use the Manager attribute and upon successful activation could perform tasks in the sub-models of the DDSS workflow. Similarly, SOJOPHO and LabTcUA were responsible for activation of the state officer, jurisdiction officer, public health officers, and lab technicians required to perform the tasks within Response. UAHierarchySOD, in addition to activation of the attributes, also captured the hierarchy between attributes and separation of duty between the tasks. Each unit moved token as shown by the direction of the arcs and, thus, enabled or disabled the transitions in a separate net. This can be viewed in Figure 7.5.

The transitions that represented the subpage (double-bordered) would be connected by ports and sockets. The sockets were double bordered as well and acted as either of input, output, or input/output socket. The type of sockets could be distinguished by the tag presented on them. Each of these sockets was connected to corresponding port in the main module and had to be of the same type. We could see the same names of ports and sockets in our model, which was not necessary but helped visualize it better. We could also think of the ports and sockets as a pipeline that connected two pages, such that any incoming arc to the substitution transition would flow the data to the input port of the subpage and the output arc would flow the data from the output port of the subpage, while the path was two way for input/output arcs. This guaranteed that the input to the substitution transition was the input to the subpage and output of the substitution transition

was the output of the subpage. Since we could have multiple input or output ports in a subpage, we had to predetermine the port and socket pair. For example, UAHierarchySOD in Figure 7.5e was a substitution transition with subpage shown in Figure 7.5f. The input port FortCollins in Figure 7.5f was connected to the socket FortCollins in Figure 7.5e. Similarly, the output ports SparyHouse and CollectMosquito on Figure 7.5f were connected to the sockets SprayHouse and Collect Mosquito in Figure 7.5e. Likewise, the input/output port, Enable VC and VS Team, were connected to the respective socket.

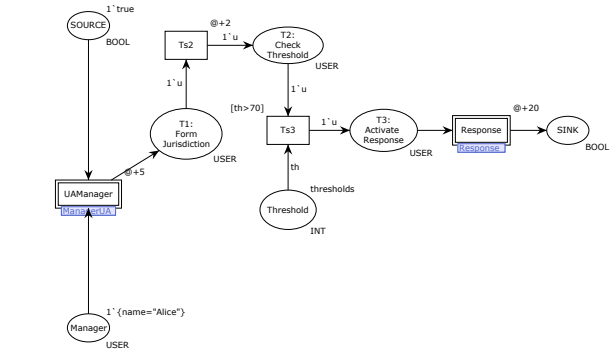
7.3.4 Model Simulation

TCPN models could be simulated interactively or automatically. With interactive simulation, it was possible to see the effects of the individual steps straightforwardly on the graphical representation of the CPN. This meant that it was easy for us to investigate the different states and choose between the enabled transitions. The automatic simulations were similar to program executions that enabled us to execute the CPN models as fast and efficient as possible, without detailed human interaction and inspection. This meant that the user could investigate the application-specific view of the current state and activities in the system missed in the interactive simulations.

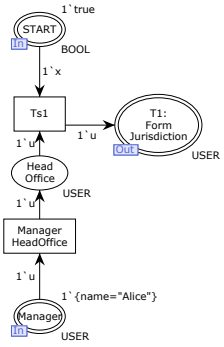
Typically, for models with very large state spaces as in our realistic DDSS workflow, it was often useful to analyze all the restricted behavior of the workflow in order to increase our confidence in the correctness of the TCPN model. CPNs offered more formal verification methods, known as State Space analysis. In this way, it was possible to prove, in the mathematical sense of the word, that a system had a certain set of behavioral properties. A state space analysis represented all possible executions of the model being analyzed. This made it possible to verify systems, i.e., prove that different behavioral properties were present or absent in a model.

7.4 Model Analysis

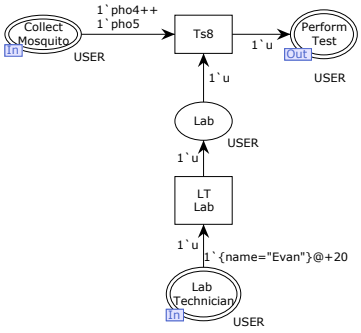
TCPN allowed us to investigate the behavior of the CPN model using simulation and state space analysis. TCPN generated all possible reachable states as well as the values of environmental vari-



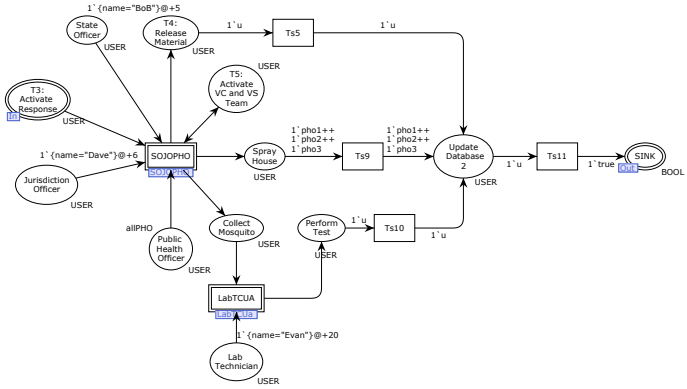
(a) DDSS



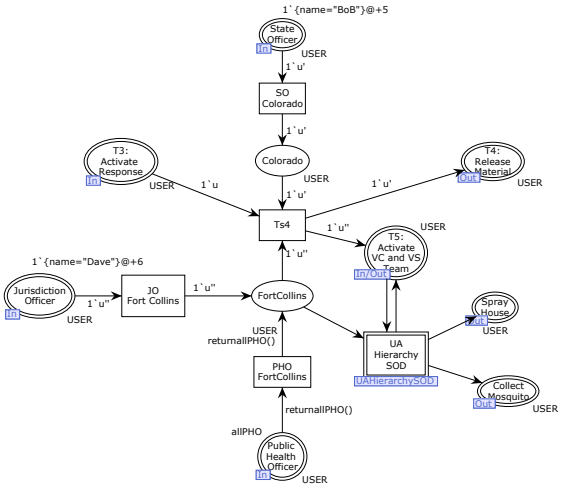
(b) ManagerUA



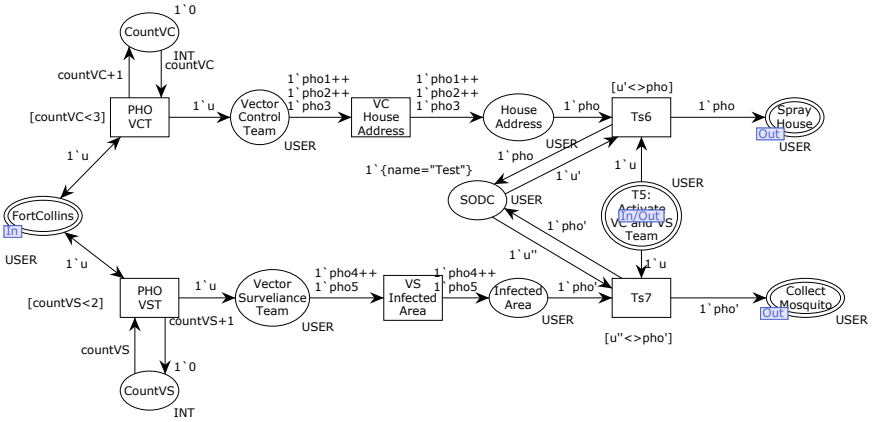
(c) LabTCUA



(d) Response



(e) SOJOPHO



(f) UAHierarchySOD

Figure 7.5: Hierarchical TCPN model

ables that caused the system change. Once the TCPN model was created, the toolbox allowed us to do a state space analysis. The TCPN toolbox generated all possible states of the workflow model. For the analysis of the TCPN model, we were interested in the determination of deadlocks in the DDSS workflow, violation of constraints like separation of duty, and the hierarchy in attributes. In order to do so, we needed to calculate the state space and build a state space graph that showed every possible state of the model and also a path to reach any state. The occurrence graph analysis elaborated all reachable states of the DDSS workflow. Paper in [46] provided details about how to perform the state space analysis.

7.4.1 State Space (Reachability) Graph

We were generally interested in what might happen when transitions might continually fire in arbitrary order. We said that a marking $M^{'}$ was reachable from a marking M in one step or in more than one step in the path between these two marking. The basic idea behind state space graph analysis was to construct a weighted directed graph which had a node for each reachable marking and an arc for each possible state change (occurring binding elements), i.e., different binding between arc expression variable and tokens in each marking.

The state space graph of this CPN model could be computed fully automatically using state space tool and made it possible to automatically verify the model policy, i.e., proved in the mathematical sense of the word that the model possessed a certain formally specified property. However, there could be an infinite number of reachable markings, generated a state space explosion. To obtain a finite number of reachable markings, it was possible to limit the initial number of tokens, which may be present in the TCPN net (this was called a bounded CPN net) or created partial state space graph by limiting the time of creating reachable marking, but this affected the analysis power of model leaving some errors undetected.

The state space graph presented in Figure 7.7 contains information of all reachable states (markings) of the TCPN model in Figure 7.11b. The marking represented the system being in a certain state. The state changed when tokens were moved following the binding elements. Fig-

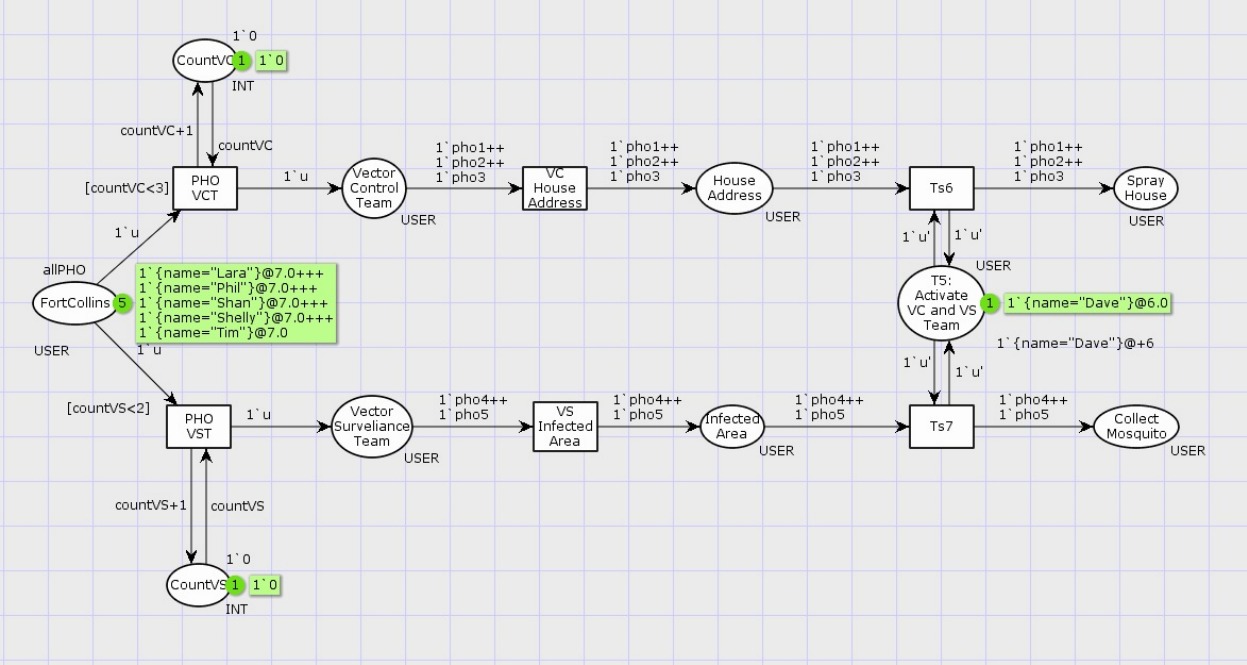


Figure 7.6: UAHierarchySOD with Initial markings

ure 7.6 shows the initial state with marking, which underwent different changes and results to the states given in Figure 7.7.

In Figure 7.7, each node was inscribed with three integers. The topmost integer was the node or state number and the two integers separated by a colon gave the number of predecessor and successor states. Node 1 corresponded to the initial state marking, and the figure shows all markings reachable by the occurrence of at most ten binding elements starting in the initial marking. The number of predecessors of the state in Node 1 equaled to zero meant none of the tokens had been moved to a new place. For each node, the directed arrows connected predecessor states to its successor states. The sharply cornered rectangle associated with each node was called a descriptor, which gave information about the marking of the individual places in that state represented by the node. The node descriptor listed the places that had an empty and non-empty marking and the current values stored in each place at that specific state. The rectangular arc descriptor associated with each arc gave information about the corresponding binding element. The node and arc descriptors had a default content. Each arc labeled with a binding element (t, b), where t was the transition name and b was the binding of variables with values, from a node representing a marking M_1 to

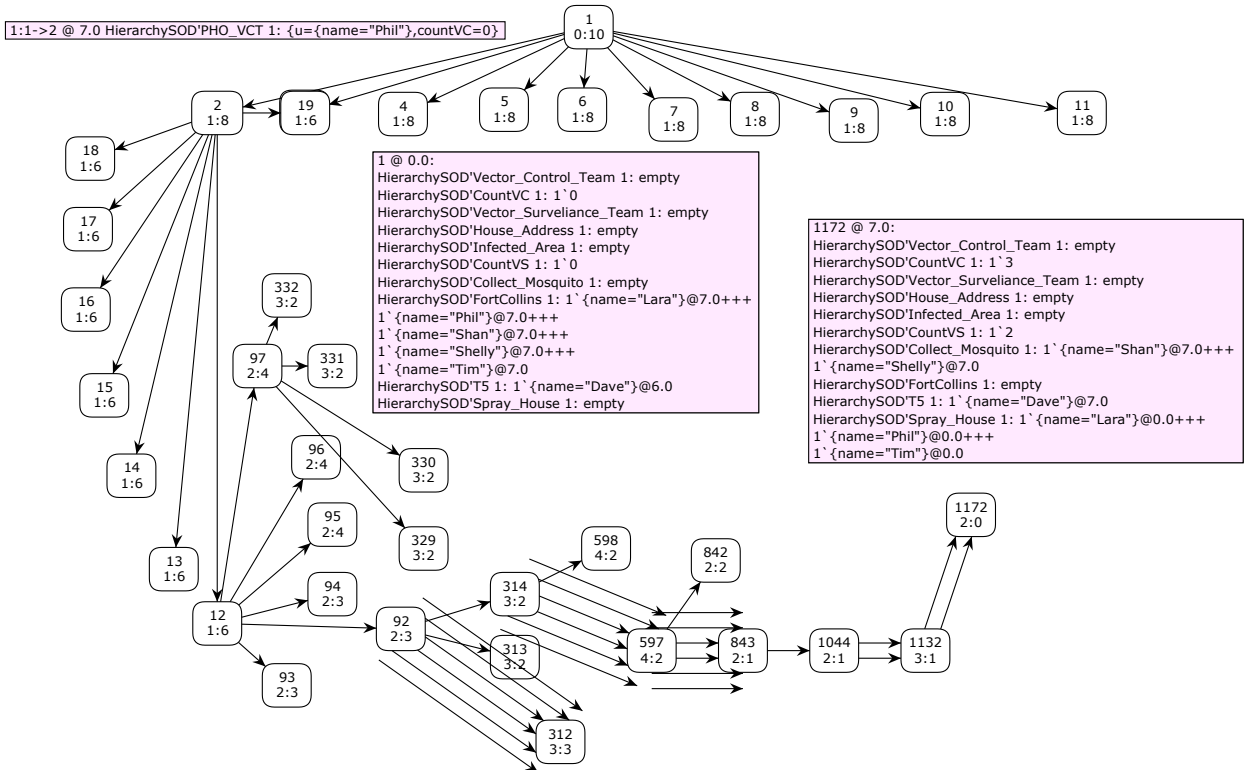


Figure 7.7: State Space Graph:UAHierarchySOD

a node representing a marking M_2 if and only if the binding (t, b) was enabled in M_1 and the occurrence of (t, b) in M_1 led to the marking M_2.

For instance, in the initial marking of the state space graph in Figure 7.7, we had the binding element at place FortCollins that enabled a transition PHOVCT. The transition moved a user token with name "Phil" to place Vector Control Team. At the same time, the CountVC in moved and incremented to reach state 2. By visual observation of the SCC graph, we were able to identify all the system state changes including the dead states. In addition, we could generate a state space report which contained information such as boundedness properties and liveness properties. For a small system, with a few states, based on the observation of the state space graph and the report, we would be able to verify that all the workflow policy constraints were satisfied. However, for a larger number of places, transitions, and constraints that increased the complexity of the net, we may not be able to visually detect invalid workflow stated in the state space graph that caused system inconsistencies, state space queries were needed in this case.

7.4.2 Analysis of Hierarchical TCPN

The path in the state space graph basically showed the binding information, which would change the state of the model. After the state space information was available, the TCPN tool provided us with the feature to generate a standard report that included the statistics, boundedness properties, home properties, liveness properties, and fairness properties. The statistics provided us with information about a number of nodes and arcs in the state space graph including the time taken to generate it and status of the graph.

The boundedness properties allowed us to observe the minimum and the maximum number of tokens and their values in each place amongst all the existing states. The home properties focused on the structure of the graph like: Is the graph strongly connected? Is it possible to obtain a certain state from all the reachable states? The liveness properties, probably the important one in terms of the existence of the model, focused on liveness of the markings and transactions. They revealed the information related to whether some transaction was never fired or some markings had not enabled binding elements. Similarly, the fairness properties looked into the fact that some transition might have a greater contribution than the other for the infinite existence of the system that was being modeled. However, in our case, our model was based on a single instance of the workflow and not considering an infinite iteration but a single run. This information will provided us with initial insights into the model.

If the model was very large, then the state space graph might not be fully calculated because, by default, TCPN tool would spend at most 300 CPU seconds for the calculation in which case the status of the report could be partial. All the properties being observed would then be based on the partial graph as well. Thus, it was necessary to confirm that the analysis was being done on complete state space information to draw a reliable conclusion from it. As our model was large and had larger state space, the default setting would provide a partial report. To overcome this, the state space was calculated after deactivating all the default settings for stop and branching options.

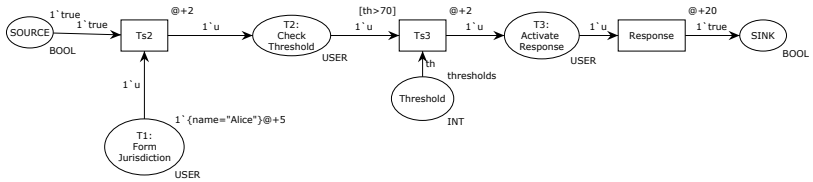
The calculation was run on a Windows platform using 6GB RAM, Intel(R) Core(TM) 2Duo CPU @2.83 GHz. However, for the TCPN model given in Figure 7.5, the state space calculation

took more than a week time period and was still not complete. The reason behind the infinite calculation could be the state space explosion or the inability of the TCPN toolbox to handle certain hierarchical component. Since we were never able to get a state space information for the model, we could not look further into the specifics of the problem. Also, this amount of time period was not feasible for any real-time systems, therefore, we considered slicing the model for the verification.

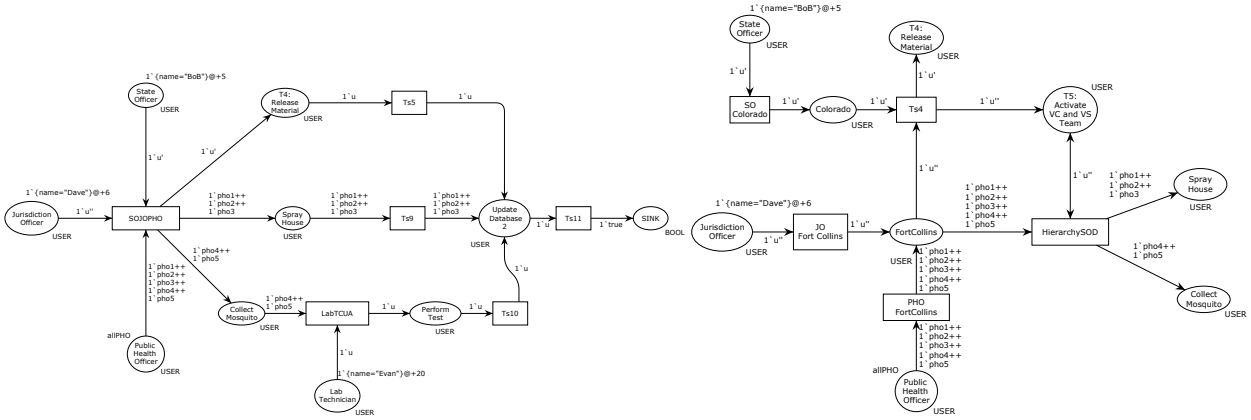
Slicing the Hierarchical TCPN

To avoid partial state space and state space explosion, we developed a TCPN model for each of the problems that we tried to model in the workflow. The sub- TCPN models were populated using values from the access control graph representing the workflow access control policies of DDSS. The slicing of the model was done in such a way that the hierarchy of the model was preserved. Therefore, the input/output pattern for each of the nets would be the same, however, they were not physically connected to each other. It further referred that the inputs were provided to each sub-model, assuming that the output from the sender-model was sent correctly. Our slicing approach was based on the facts presented in our previous work [47], which stated that if every sub model verified the property being analyzed than it would be true for the complete model. Therefore, we could safely, mark the inputs to different models as all the models would be verified for the correctness. In order to develop a sliced model, different nets were created as shown in Figure 7.8. The hierarchical model and the sliced model represented the same concept except the physical binding between ports and sockets. This allowed us to remove some of the arrange the models to fit the scenario with fewer notations. For example, the Start port had been removed from Figure 7.5b and Manager place had been removed from Figure 7.5a since the Manager had been assigned the required attribute and task in Figure 7.8d such that we could mark place from Jurisdiction in sub-model of Figure 7.8a upon success full verification of sub-model in Figure 7.8d.

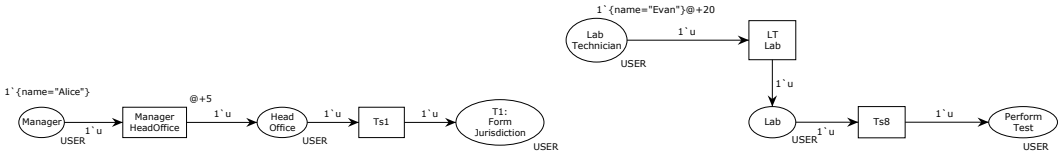
After slicing the hierarchical model, each individual nets was run for state space information. The result for the standard report is listed in Table 7.1. The table, however, does not show the fairness property because our model was based on a single instance that caused the fairness prop-



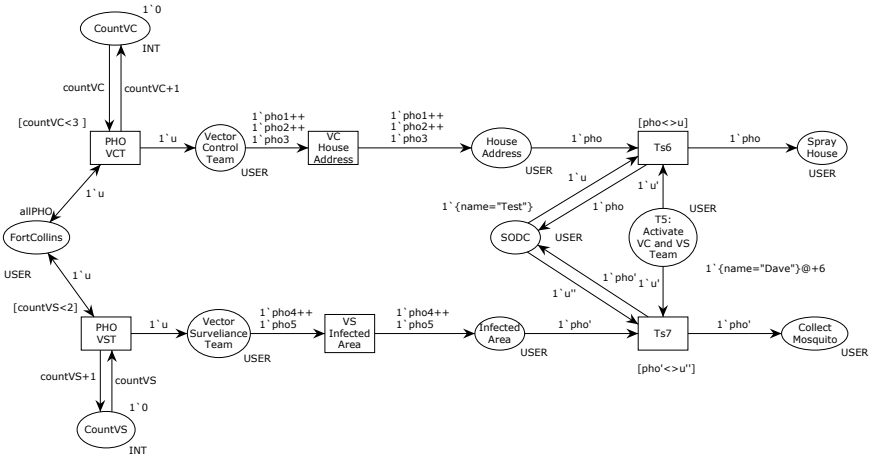
(a) DDSS



(b) Response **(c)** SOJOPHO



(d) ManagerUA **(e)** LabTCUA



(f) UAHierarchySOD

Figure 7.8: Hierarchical TCPN model after Slicing

Table 7.1: Standard State Space report for Sliced Model

		DDSS	Manager UA	Response	SOJOPHO	LabTCUA	UAHierarchySOD
Statistics(State Space)	**Nodes**	4	3	1019	80	3	20986
	Arcs	3	2	4702	1339	2	79160
	Secs	0	0	1	0	0	202
	Status	Full	Full	Full	Full	Full	Full
Statistics(Scc Graph)	**Nodes**	4	3	1019	80	3	20986
	Arcs	3	2	4702	1339	2	79160
	Secs	0	0	0	0	0	2
Liveness Properties	**Dead Markings**	1	1	1	60	1	4000
	Dead Transition Instances	None	None	None	None	None	None

erty to state "no infinite sequence occurrence" for each of the nets and therefore is not significant. From Table 7.1, we could observe that the combined net was fairly large (considering the number of nodes and arcs) and the time taken to compute the information was a small finite number. We could see a few dead markings being reported that actually showed the number of final states possible for each of the nets. The larger nets, UAHierarchySOD and Response, ended up having a larger number of possible final states as shown in the report (by the dead markings). Similarly, for all the nets, there were no dead transition instances, which meant that each of the transitions was fired at least once. Another thing that was left out from Table 7.1 was the boundedness properties, which would provide information about upper and lower limit bounds for each place instances. The inspection of those tokens showed that the upper and lower multiset were derived without considering the time factor and, therefore, were not reliable for our analysis of Time Petri Nets for inconsistencies in the UAHierarchySOD and SOJOPHO nets. The upper and lower integer bounds, however, revealed an inconsistency. For some places, the upper bound was less than desired.

The standard report led us to believe that the model might have some errors within it, therefore, we further devised queries to run across state space graph, using the ML language to derive conclusions about the properties we were interested in. The queries used to explore the specific property and their logistics are shown below.

7.4.3 State-Space Verification Queries

Verifying all the reachable states in the state space graph was a time consuming and error-prone task. To solve this problem, we created queries using the Standard ML language that allowed us

to select a subset of reachable states having the properties that we were interested in. We wrote state space queries to investigate the properties of the DDSS TCPN model. These queries made it possible to investigate the reachability, boundedness, liveness, and fairness properties in the DDSS workflow using standard queries; and non-standard queries could be drafted by writing TCPN ML functions. We made state-space queries by creating auxiliary text containing the query functions and then, by using the evaluate ML tools, we were able to evaluate the text. In particular, we were interested in writing quires to detect the following problems with the workflow specification:

- DeadLock

  We considered a system to be in deadlock if some task was not successfully completed. There could be many reasons for a system to be in deadlock. One of the very common reasons could be the unavailability of the user to perform a task. Furthermore, users might not be available because they had been assigned to more than one task at the same time, or the spatial-temporal constraint did not satisfy, or the modeling was erroneous such that some tokens or transitions were never triggered. Our goal was to devise a standard query to detect a deadlock, if present, irrespective to the cause behind it. Therefore, we focused on the number of tokens to be present in the sink node at the end of the transactions. This was based on the fact that, if everything ran as it was supposed to be, we could evaluate the expected number of tokens in the sink node. For our model, since the sink nodes were preceded by a task that was or-joined to three other tasks, our expected number of token in the sink node was five(for our complete model). For the sliced model, there were more that one sink or source nodes and different estimates for each of them. Therefore, we showed below our query to determine the presence of deadlock in the complete model. The query traversed the entire SCCGraph. While doing so, it checked if the node did not produce any successor and the number of tokens in such node was as desired. In other words, the query was looking for the end nodes (final states) that might not have received all the required tokens. If it found any inconsistent final state, the state would be added to a list. The list was

our output from the query. We modified the name of sink nodes and numbers of each net to determine if there had been a deadlock in the net.

```
SearchNodes(
EntireGraph,
fn n => (length(OutArcs(n)))=0 andalso
not((size(Mark.DDSS'SINK 1 n))=5),
NoLimit,
fn n =>n,
[],
op::)
```

The query returned a null list of nodes for all the nets except for sub-net "UAHierarchySOD" which is shown in Figure 7.9 Although the standard report indicated there was no dead transition on this instance, there was some transition that did not fire as they had to be. To investigate further, we drew a partial graph for the nodes that were having an issue and carefully inspected the erroneous state and the predecessor state. The state information for one of the erroneous state and its predecessor are shown in Figure 7.10. We found that the place "T5" from Figure 7.8f had empty tokens. T5 in our case was the task of activating the vector control team and the vector surveillance team. We knew that only one user, i.e., the Jurisdiction officer, was responsible for performing the task "T5". As a result, when the token "Jurisdiction officer" would activate either of Ts6 or Ts7 once, there would be no token left to activate the transitions again. Addition of separation of duty constraint for those tasks further caused tokens to move one at a time and, therefore, not enough users were available to perform the task. As a correction, we changed the output arcs connecting "T5" to "Ts6" and "Ts7" to input/output arc such that the Jurisdiction officer performed the task and also was available for other activation. The deadlock query was run once more after new state space was calculated and the verification of no deadlock in the model was done as it now

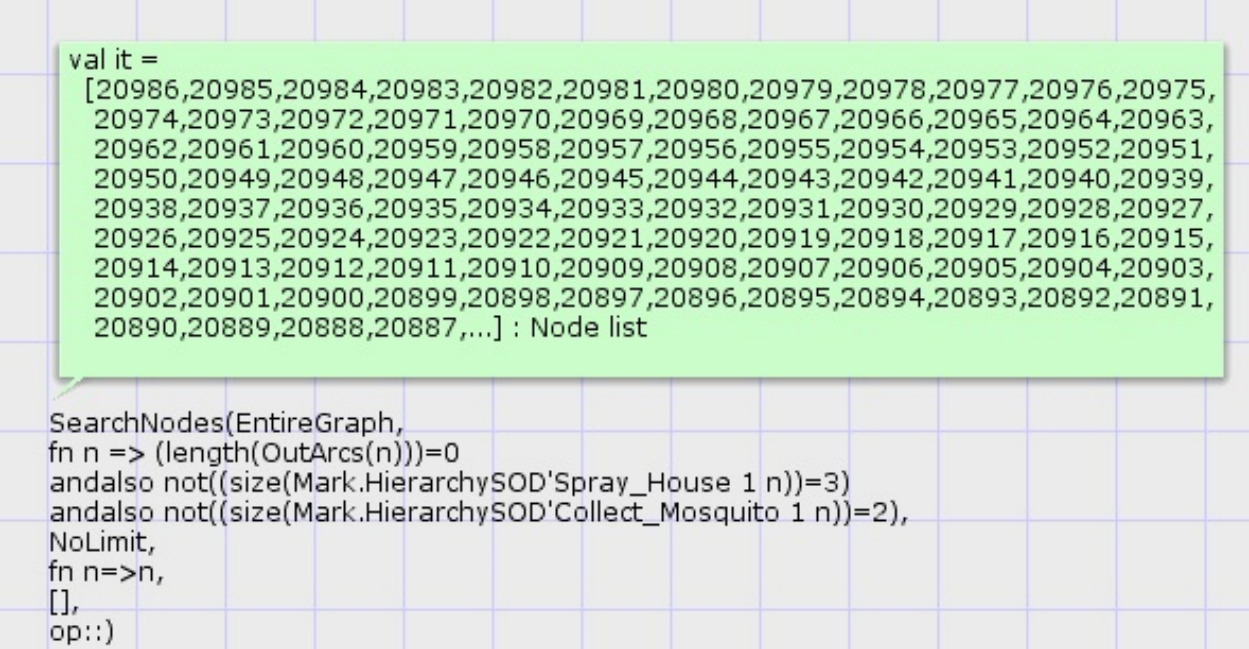


Figure 7.9: Output to Deadlock Detection Query for UAHierarchySOD

returned the empty node list, which indicated there were no final states where the sink nodes did not have enough tokens, i.e., five user tokens.

- Separation of Duty violation

  The separation of duty constraint existed because there were some users who were eligible for multiple tasks, but they were allowed to perform only one of the task in any instance. Such tasks in our system were “Spray House” and “Collect Mosquito” as shown by end places in Figure 7.8f. Both of those tasks could be performed by any member of Public Health Officer, again represented as the places in Figure 7.8f, as all the Public Health Officer were equally eligible to a member of the vector control team or vector surveillance team. Our query to check for separation of duty violation would see if, at any state when both of those tasks were assigned to some user, they were different. The following is the query for our complete model. The query traversed the entire SCCGraph in search of states where there were common token in both task places: “Spray House” and “Collect Mosquito” when there was at least one token present in them at the same time. The output of the query was the list

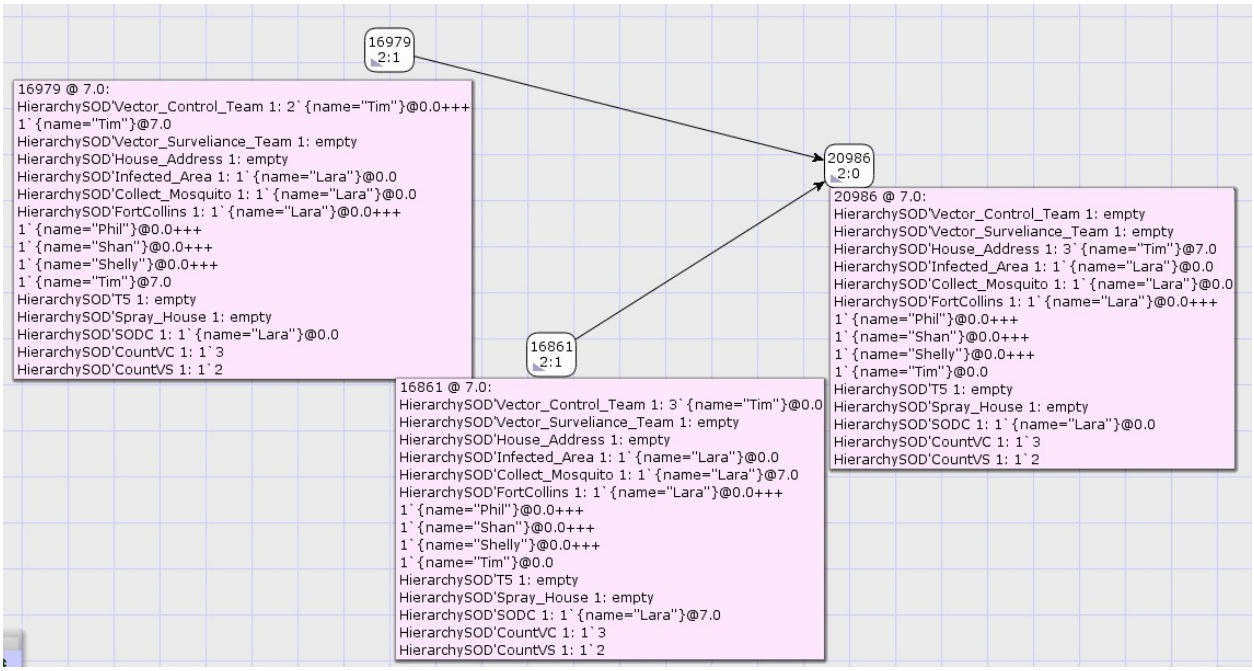


Figure 7.10: State Information of Partial Graph

of states that violated the separation of duty constraints or nodes found to meet the condition in the search described before. In case of the sliced model, we did not need to verify it in each net but “Response”,“SOJOPHO”, “UAHierarchySOD”, which contained the task that required the separation of duty.

```
SearchNodes(
EntireGraph,
fn n => not (List.null(intersect
(Mark.UAHierarchySOD’Spray_House 1 n)
(Mark.UAHierarchySOD’
Collect_Mosquito 1 n)))
andalso
size(Mark.UAHierarchySOD’
Spray_House 1 n)>0
andalso
```

```
size(Mark.UAHierarchySOD'
Collect_Mosquito 1 n)>0,
NoLimit,
fn n => intersect
(Mark.UAHierarchySOD'
Spray_House 1 n)
(Mark.UAHierarchySOD'
Collect_Mosquito 1 n),
[],
op::)
```

When the query was run on each of the nets, we found that the correction measure applied for deadlock situation also helped us to find the separation of duty constraint violation. It was because, with tokens only assigned to one of the tasks, there would be no such states where the same token would be present in both the task as only one user token could be assigned to any one of the tasks due to lack of "Jurisdiction Officer" type token to enable the transition any further. After the correction was applied, all nets returned an empty list showing there were no common tokens except for the UAHierarchySOD sub-net shown in Figure 7.8f. We found that the separation of duty constraint model as in [44] was not able to effectively implementing the separation of duty.

Furthermore, we observed that separation of duty constraints for any task could be reflected in the attributes required for that task. For example, if T1 and T2 could not be assigned to the same user in one instance, we could also say that the attributes required for T1 (say a1, a2) and attributes required for T2 (say a3, a4) could not be obtained for the same user in that instance. This helped us to simplify the model and effectively implement the separation of duty to the task through other attributes. Therefore, as a correction measure, we removed the SODC from Figure 7.8f and moved the separation of duty to attributes vector control team and vector surveillance team as shown in Figure 7.11b. As before, the query was run

again on improved state space and verified for the correctness. Since the query resulted in an empty list, we could say that there was no violation of the separation of duty constraints.

- Hierarchy violation

  One of the requirements of our example was that the Jurisdiction Officer and all the Public Health Officers were enabled in Fort Collins. The Public Health Officers were further allowed to work as either of Vector control Team or Vector Surveillance Team. The same was not true for Jurisdiction Officer. Therefore, our query was designed to find if any inconsistency existed in a hierarchy that existed for attribute Public Health Officer and Vector Control or Surveillance Team. As we could not use the upper multiset, due to the time factors, we tried to look at whether there were any state markings of Spray Houses or Collect Mosquito places that had common timed tokens to Jurisdiction officer and T5 had common timed tokens to public health officer. Furthermore, we devised a separate query for each of the cases so that it was easier to see which hierarchy was under violation. The query given below is for the complete connected model. Each query traversed the entire SCC Graph and looked at whether any assigned token to the conflicting places were the same at a certain time. The output of the query was the list of states that had conflicting user tokens. Basically, we were looking at whether Jurisdiction officer had been assigned to Public Health officer any point of time and vice-versa. In the sliced nets, since we were looking for the violation in SOJOPHO only, the query needed no changes.

```
SearchNodes(
EntireGraph,
fn n =>
size(Mark.SOJOPHO'Collect_Mosquito 1 n)>0
andalso not(List.null(intersect
(Mark.SOJOPHO'Collect_Mosquito 1 n)
[{name="Dave"}@6.0,{name="Dave"}@7.0])),
```

```
NoLimit,
fn n => n
[],
op::)

SearchNodes(
EntireGraph,
fn n =>
size(Mark.SOJOPHO'Spray_House 1 n)>0
andalso not(List.null(intersect
(Mark.SOJOPHO'Spray_House 1 n)
[{name="Dave"}@6.0,{name="Dave"}@7.0])),
NoLimit,
fn n => n
[],
op::)

SearchNodes(
EntireGraph,
fn n =>
size(Mark.SOJOPHO'
Jurisdiction_Officer 1 n)>0
andalso not(List.null(intersect
(Mark.SOJOPHO'Jurisdiction_Officer 1 n)
[{name="Shan"}@7.0,{name="Tim"}@7.0,
{name="Shelly"}@7.0,{name="Phil"}@7.0,
{name="Lara"}@7.0])),
```

```
NoLimit,
fn n => n
[],
op::)
```

In the query, we added the desired tokens to be avoided (e.g., Dave@6 and Dave@7) manually because the boundedness multiset would also consider the tokens ahead of time. We compared the tokens that were available for the transition. Also, in the first and second query, we saw two instances of "Dave" that were timed differently. It was because Dave was available after time stamp 6 but we would not want Dave to interfere with the hierarchy any time after that as well. The query ran on the state space, each returned a null list that, in turn, indicated that there were no states where the hierarchy was violated.

- Cardinality constraint violation

  There were a few cardinality constraints in the DDSS workflow. In order to check for the violation of the cardinality constraint, we looked into the number of tokens at those places at each state and compared them with the cardinality constraint. For each place, we wrote a different query. Following is a query to check for place "Spray House", which was required to have at most three tokens. The query traverses the entire SCCGraph and returned the states where the number of tokens was greater than the required number (three in the given query). The output was the list of states where the condition was found to be true. For the sliced model, it was necessary to verify it in every net whereas, for the connected model, verification on page UAHierarchySOD was sufficient due to the port socket connection.

```
SearchNodes(
EntireGraph,
fn n => size(Mark.UAHierarchySOD'
Spray_House 1 n)) <=3),
```

```
NoLimit,
fn n =>n,
[],
op::)
```

The query was run for all the places to verify the correct number of tokens present. All of the queries returned an empty list indicating that there were no states where the cardinality constraint was violated.

7.4.4 Hierarchical Model After application of Correction measures identified from analysis of Sliced Model

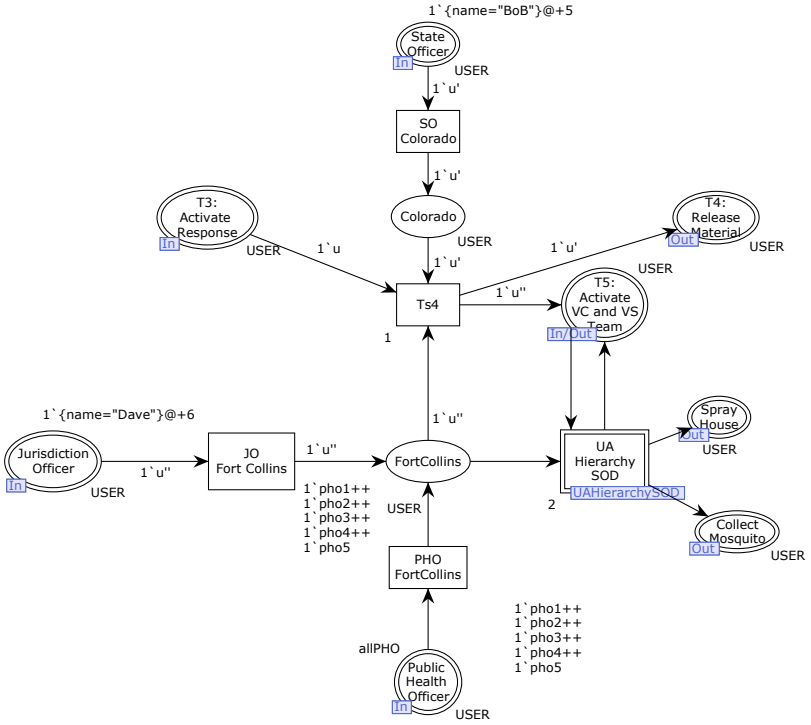
After analysis of the sliced model, we figured different problems that existed in the system and verified the correctness by addressing the problems. The correction measures were, thereafter, applied to the hierarchical model as well. After the correction, the model was run for calculation of state space. Unlike, before, the calculation was completed in about fifty five hours, which was an improvement of the performance itself. Since we were able to generate the state space standard report, we further verified that the sliced model was able to find all the flaws in the model and, therefore, the complete hierarchical model should be out of flaw as well. To do so, we ran all the queries devised before. All the properties were verified on the complete model except the Hierarchy violation. This was due to the fact that, the sliced model, here, was mimicking the behavior of the hierarchical model, during which it tended to miss the smaller details. For example, in Figure 7.5e, the substitution transition ”UAHierarchySOD” was taking input from place “Fort Collins”, “T5” and providing output to places “Spray House”, “Collect Mosquito” and “T5”. However, these are four separate port a socket combination. To be more specific, each place in Figure 7.5e was connected to the places in Figure 7.5f with the same names i.e., tokens could move in and out only through this connected places. The same behavior, when replicated as the sliced model, the substitution transition was now a normal transition and behaved as one single transfer point. It further caused a cycle within i.e., HierarchySOD transition in Figure 7.8f could

Table 7.2: Table for Standard Report (Hierarchical Model) After all corrections are performed

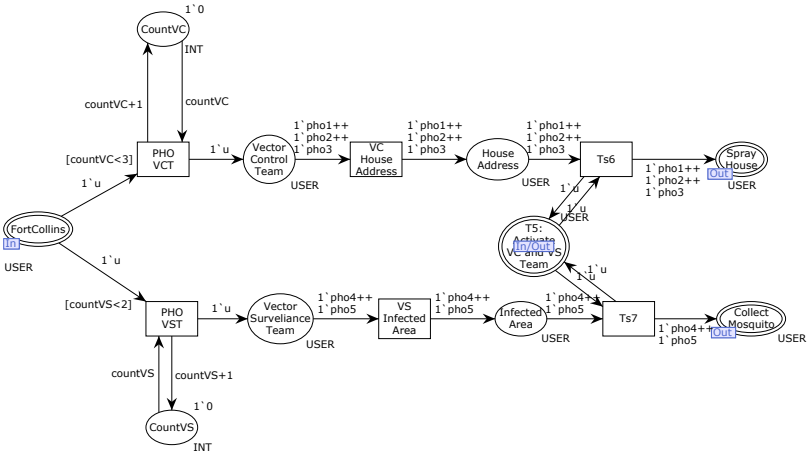
Statistics(State Space)	**Nodes**	157136
	Arcs	796544
	Secs	15130
	Status	Full
Statistics(Scc Graph)	**Nodes**	157136
	Arcs	796544
	Secs	120
Liveness Properties	**Dead Markings**	6
	Dead Transition Instances	None

not be fired until a token had been received from place "T5" that further required transition "Ts4" to fire before transition "HierarchySOD" every time. On the contrary, either of transition "Ts4" of net UAHierarchySOD could be fired from a token in place "FortCollins" in case of the hierarchical model shown in Figure 7.4. Therefore, when token type "Jurisdiction Officer" was available, it was able to fire either of the transaction in the hierarchical model but only "T4" in case of sliced model i.e., Figure 7.8f. Therefore, the result from the analysis of the sliced model was different from the hierarchical model. In order to solve the problem, we could either guard the transition "UAHierarchySOD" in Hierarchical model, only allowing it to fire when the hierarchy was met or we could prioritize the transition "Ts4" over "UAHierarchySOD" such that Ts4 fired before UAHierarchySOD and let the temporal constraint come into play.

The latter choice, in addition to solving the hierarchical model issue, also added a feature to the model i.e., from the workflow control flow, the activation of VC and VS team was done before they could actually perform their job. Prioritizing the task further enforced this fact, which was why we chose the second approach as the solution. The resultant model is shown in Figure 7.11. However, only the changed nets are shown and all other nets would resemble Figure 7.5 After, the small change, the hierarchical model was re-analyzed and queried. The standard report is shown in Table 7.2 and the analysis showed all empty list indicating that the model was not violating any of the properties.



(a) SOJOPHOfinal



(b) UAHierarchySOD

Figure 7.11: Improved nets from Hierarchical MOdel

7.4.5 Model Performance

As we saw in our analysis, it was not possible to figure out the problems in the model via the hierarchical model. Due to the large nature of the model, the error would propagate and magnify, therefore, would lead to the state space explosion. Slicing the model into equivalent nets was a viable solution for the problem. It not only made the analysis possible but reduced the time for analysis. The slicing of the model made it possible to focus on the properties that we were concerned about. It further made the problem smaller, as if the problem was divided into smaller individual parts without affecting the output of the analysis. Slicing the model was equivalent to implementing a divide and conquer approach for the analysis itself. Also, we benefited from slicing as we could understand the problem better since the focus of the problem was now smaller and distributed. However, we found that, in order to have a reflective, replicative analysis, the sliced model should be able to preserve all the properties of the model being sliced. From Table 7.1 and Table 7.2, we can see that there was a huge difference in the time taken for the state space calculation. Since in the sliced model, there were different independent nets, we needed to consider the time taken for all the state space calculation. The improvement we had gained by slicing the model is shown in Table 7.3. The improvement in performance was due to the reduction of the number of states in the nets. A hierarchical model would consider every possible state and would create a larger state space. While the model, if sliced, different properties could be verified in different nets. In addition, once an input and output was verified for a net to satisfy some property, we did not need to consider every possible output as input for the next system but one. This gradually decreased the total state space required for the analysis of the problem and, therefore, enhanced the performances of the analysis without any decline in the quality.

Here we were considering that there was some time that was within the zero CPU time and since there were multiple nets taking those time, we assumed it to be around one in total.

Table 7.3: Table for Comparison of performance between two models

	Time taken [Sliced Model] (sec)	**Time taken [Hierarchical Model] (sec)**
DDSS	0	15130
Manager UA	0	
Response	1	
SOJOPHO	0	
LabTCUA	0	
UAHierarchySOD	202	
Total	$203 \approx 204$	15130
Gain	((15130-203)/15130)*100 = **98%**	

Chapter 8

The Enforcement Mechanism

In Chapter 6 we presented our model to effectively represent the spatio-temporal attribute based access control for the management of workflow task. We also verified the model by using TCPN in Chapter 7, which indicated the absence of inconsistencies with respect to the representation of the access control policies. However, it was necessary to study the suitability of our model in the real-world system. How the model should be enforced such that the mobile component was flexible as well as protected? Therefore, in this section, we further investigate the practicality of our model. In other words, we present the technical simulation of our high-level model perspective. While enforcement of attribute-based access control was still a challenge, we looked at the coordination of spatio-temporal component with workflow task. We tried to explore the possible bottlenecks and challenges to implement our model. We also analyzed the correctness of the authorization graph and analyzed the access response time to evaluate the feasibility. Furthermore, through the enforcement architecture, we investigated a possible configuration feature of the system. What possible components could be placed into the enforcement mechanism, their significance, and their communication protocol? The goal of having an enforcement simulation was to be able to predict the configuration flexibility when the proposed model was used for a real-world example and discover erroneous behavior beforehand.

8.1 Solution Architecture

As our model considerd the spatio-temporal factors, the architecture needed to verify the STZone of the mobile devices. Additionally, it had to deal with the changing STZone after the access was granted. In other words, the mobile device when entered an invalid zone, the access provided should be revoked. We followed a similar concept as in [47] to address these challenges. The mobile unit, with help of its GPS and STZone builder components, built an appropriate access request, thus, formed STZone also contained the timestamp of when the request was created.

The STZone was later verified at the Resources component by tallying it with the cell identity of the mobile devices. Altogether the components kept a balance between security and accuracy with respect to the STZones verification of the mobile device. Similarly, the authorization was adjusted with respect to the changing location and time by having pre-authorization and ongoing-authorization, i.e., pre-authorization decided if the access should be given to some object for a given request. After the request was granted, the ongoing authorization kept track of the current location and time. All the given privilege was revoked if ongoing authorization failed to confirm the valid location and time.

The system architecture shown in Figure 8.1 is responsible for handling all the above-mentioned task. All the three components were standalone components separating the different modules.

The mobile unit was installed in the mobile device which communicated with resources components with some access request. Considering the limited storage and processing capacity of the mobile unit, to avoid heavy processing within the device, rather transfer the required information was transferred to the Resources component. The access request contained information about which user, what object, what operation, and the spatio-temporal zone of the mobile device. The mobile device was tied to some user with user certificates, i.e., user could access the system only through their own device or by logging in with their credentials. These credentials were stored in the mobile unit itself for verification. They were never communicated to any other module through any of the communication media to maintain the anonymity of the user with other modules. The Resources components further forwarded the request to Access Control Module which determined if the request was valid or not. Before the validation of the actual request, the validation of the STZone was performed making use of STZone verification component. The verification required mapping the physical location to the logical location as described in Chapter 5. The Access Control Module was also responsible for monitoring the changes and notifying the Resources if the request was no longer valid. We could reduce the number of communication between different modules if the access request for ongoing authorization was triggered by the mobile unit. An authorization token was returned to the sender if the access control was granted. The information packaged

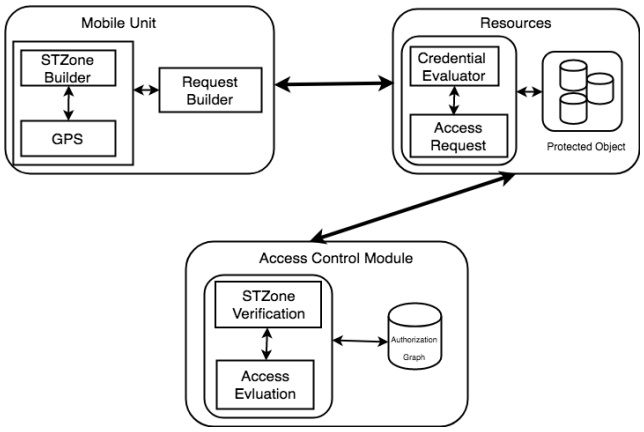


Figure 8.1: System Architecture

in the authorization token made it possible for the mobile unit to keep track of the zone the task allowed and trigger another request if the zone was no longer valid. Furthermore, the access control module needed to maintain the policy authorization graph. The graph was subject to change and the changes were allowed only by the administrator which was not shown in the system. Our assumption here was that we had a genuine, non-malicious administrator responsible for handling any changes to the policy graph.

8.2 Protocols for secure communication

The system architecture is shown in Figure 8.1 which reflects different modules communicating with each other. In order to make those communications secured over unsecured channels, we proposed the following protocol. The protocols tended to preserve the information being transferred from common security attack.

8.2.1 Assumptions on the system

- All the modules should have an agreement on public keys that should be used for encryption of the message and verification of the signatures.

- For each user registered in the system, a pair of public and private key should be created by the central authority. The public keys should be securely communicated with each party involved in the communication.

- All the communication should be time-stamped and, therefore, there should be an impartial synchronization of the clocks between the participating modules.

- All message packages should be hashed using MD5 hashing technique and signed using the recipient's public key and extracted using the private key.

8.2.2 Steps of the Protocol

1. A mobile unit should send an encrypted and digitally signed access request package M_1 to Resources server. M_1 should be created by hashing and signing with resources server's public key. The payload of M_1 should contain id of the user (ID_u), id of the device (ID_d), task being requested access to (T), Spatio-temporal Zone ($STZone$), Cell Identity (CI), Operation being requested(OP) and timestamp (TS). The resource server should validate the user certificate and STZone.

2. If the validation of user and STZone is confirmed, the resource server should send the message package M_2 to the access control module server. The payload to M_2 should contain id of the resource server (ID_rs),id of the user (ID_u), task being requested access to (T), Spatio-temporal Zone ($STZone$),Operation being requested (OP) and timestamp (TS).

3. The ACM server should validate the identity of Message M_2. If the validation is successful, the authorization graph should be evaluated for computation of access decision. If the permission is granted ACM should respond with message package M_3 that contains authorization token. If the permission is not granted, ACM should respond with message package M_5 with access denied message. The payload of M_3 is id of acm server (ID_as), id of user (ID_u), authorization token (AT), and timestamp (TS). The payload of M_5 should be id of acm server (ID_as), id of user (ID_u), access denied message(M_{ad}), and timestamp (TS)

4. The response package M_3 or M_5 should be validated and sent to the user. Either of M_4 or M_6 should be sent to the mobile unit with respect to received response package M_3 or M_5. The payload of M_4 should be id of the resource server (ID_rs),id of the user (ID_u), id of the device (ID_d),authorization token (AT), and timestamp (TS). The payload of M_6 containsM_4 should be id of the resource server (ID_rs),id of the user (ID_u), id of the device (ID_d), access denied message(M_{ad}), and timestamp (TS). The authorization token should contain the information of permission and valid zones for the permissions.

5. In case of ongoing authorization, the mobile unit would need to trigger an access request with updated STZone . The prevailing access would be revoked if acm server evaluated access denied. Similarly, the access would be continued if the evaluation returned authorization token and previous authorization token would be updated.

8.3 Specifications of Authorization Graph in Neo4j

Our model defined policy as different policy elements maintaining relationships with each other. The relationships that could exist in the authorization graph according to the model were the assignment, association, and prohibition. All of these relationships were directed and there could not be any cycles. For example, if there existed an assignment relationship from A to B, there could not be an assignment from B to A. Same went for association and prohibition. Additionally, the association and prohibition were always meant to be from user attributes to the object attributes and not the other way around. With all of these facts, we were able to conclude that our authorization graph was a Directed Acyclic Graph (DAG). Therefore, we utilized the power of graph database to preserve the logical representation of the authorization graph. Additionally, we chose Neo4j over all other graph databases.

Neo4j is a NoSQL native (graph specific storage and processing engine) graph database that implements a property graph model [48] to the storage label. In addition, it provides facilities such as ACID properties. A property graph model is comprised of nodes, relationship, properties, and labels. Nodes are the main data elements connected with each other via relationships. The nodes,

as well as the connecting relationship, can have properties of their own. In order to categorize the nodes into the related groups, labels are used. A node can have multiple labels at the same time. All the labels are indexed, in order to accelerate the process of finding nodes in the graph. Neo4j makes use of Cypher Query Language for its operation. Cypher is a declarative graph query language, inspired by SQL, with pattern-matching capabilities. Finally, Neo4j allows computation of a record's location in O(1) time. This is made possible by storing the file records into multiple node storage files. The records stored are fixed-sized (9 bytes). Therefore, if we have a node with an id equal to 100, then we know that the record begins 900 bytes into the file. Also, it maintains an index-free adjacency which is to have each node maintain a direct reference to its adjacent nodes. Maintaining an index-free node is cheaper than having a global index and allows query execution to be independent of the size of the graph but proportional to the amount of graph searched. The use of relationships instead of indexes further aided with fast traversal of the graph. As our authorization decision was based on traversal of the authorization graph, the features provided by Neo4j further motivated us to use it for storing and processing the policies.

Following are the list of components in our model and how they were being represented in the Neo4j graph database.

- Users: Users were represented as a node. They have a unique property called id which distinguished a user from others. Another property in the user node was the name which held the name of the user. The nodes were labeled "User".

- Objects: Similarly, the object was represented as the nodes labeled "Objects" and had a unique property called id and another property name which held the name of the object.

- User Attributes: The user attributes were nodes labeled "UserAttributes". They only had one property called name and was unique among the user attributes. In our model, we represented the zones and the task as the user attributes. However, it would be easier to be able to distinguish them from normal attributes as they bore some special characteristics in the model. Therefore, the task attributes were named as *task_TaskName*. In case of zone

attributes a separate label “Zone” was used. This was because zone attributes were also object attributes and having them represented as single node preserved space and also aided the processing of authorization graph during processing of access request.

- Object Attributes: Unlike user attributes, task were not the part of object attributes. The relationship between object and task were represented via an association relationship. Therefore, object attributes were simply represented as nodes labeled “ObjectAttributes” and having a property name. As discussed above, the nodes labeled “Zone” were also object attributes.

- Operations: The Operations were represented as nodes labeled “Operation”. They did not have any unique property. Each Operation node contained three boolean type properties called read, write, and update with values true or false depending on the availability of the privilege. The operation node, in general, could have been represented as the property of the association relationship. However, our algorithm worked better with node type representation and, therefore, the design decision was made to have an operation type node in the system. Also, our system assumed to have three kinds of operation which was represented as the properties of the “Operation" type node. It would be more efficient (in terms of space) to represent operations node having a single property called name in a system having numerous operations. As the complexity of processing remained the same, we decided to have operations as properties as we were working with three different kinds of operations.

- Policy Classes: Policy Classes were represented as nodes labeled“PolicyClass" and uniquely identified by its property name.

- Assignments: The assignments were represented as relationships labeled “Assignment" with no property.

- Associations: As association in our model existed between user attributes and object attributes and carried the information of the operations, we modeled association as two outgoing relationships, labeled “Association", from user attributes and object attributes connected to the operation node.

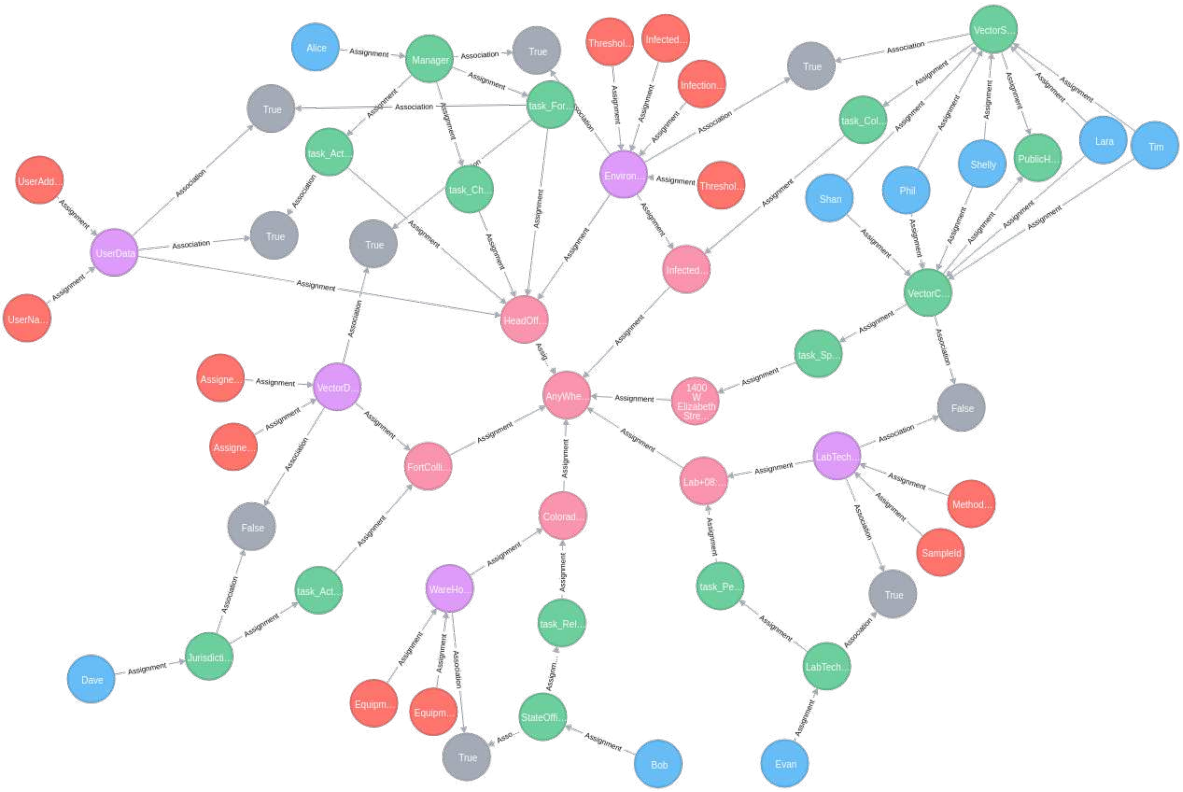


Figure 8.2: Neo4j Authorization Graph

- Prohibitions: Prohibitions were representation similar to associations labeled “Prohibitions".
- Obligations: The Obligations were event-based and, therefore, implemented as a transaction event handler in Neo4j which were equivalent in Triggers in relational databases.

Apart from the above-mentioned elements, the other part of the model like PROC and Separation of Duty were part of the implementation and not the graph database. Also, the above-mentioned representation was not the hard and fast rules to represent the access control policies but the one we found more flexible and useful. As Neo4j is a NoSQL database, we could have a varying number of attributes in each node with the same label and, therefore, could add or remove information as per the requirement. The Neo4j graph diagram for the example DDSS policy is shown in Figure 8.2

8.4 Experimental Setup

After having made the design decision for the representation of the policies in Neo4j, there were two major questions to be answered about the model. Firstly, is the model and representation able to answer the queries correctly? Secondly, is the performance good enough for a real-life system. In order to quench the answers for two different kinds of questions, we developed two separate setups and looked through the problems.

8.4.1 Algorithm for Processing of Authorization Graph

The algorithm to process the authorization graph was based on the Depth First Search of the authorization graph. The algorithm was implemented as the plugin to the Neo4j which eradicated the necessity of connecting to the server every time there was a requirement of data flow to and from. It was a basic algorithm used for the verification purpose and also aided to explore the required improvements.

The algorithm presented in 2 did not show the details of how the violation of SOD was checked. To do so, a different class was implemented in java which reads the file that contained used attributes and the file that contained the list of Separation of Duties and determined if the user had been using the attributes that violated the separation of duty constraint with the attributes that it intended to enable (usedAttributes in line 26 and 39). Also, the algorithm was designed to ignore the task attributes if the task name was not supplied. The feature was mainly introduced for the performance testing of the algorithm in which case the access request was being processed against a randomly generated graph which might or might not contain task as the attributes.

8.4.2 Correctness of the algorithm

To determine if algorithm 2 was actually giving us the correct response, i.e., true if all the attributes were present including the zone and task without violating the separation of duty constraints or else False, we created the authorization graph in Neo4j representing the components mentioned in Figure 8.2. We ran queries for each user, from each available zone and task to each

Algorithm 2: Check Access Request

input : userId, zone, objectId, accessType, task
output : True **or** False

```
userNode = Node with id userId
objectNode = Node with id ObjectId
userReachable = Nodes reachable from userNode through DFS
objectReachable = Nodes reachable from objectNode through DFS
accessNode = NULL
containedZone = NULL
taskNode = NULL
output = False
foreach ur ∈ userReachable do
    if objectReachable contains ur then
        if ur is operation Node and has accesstype true then
            accessNode = ur
        end
        else if ur is Zone and equal to zone then
            containedZone=ur
        end
        else if  name property of ur equals task then
            taskNode=ur
        end
    end
end
if task = NULL then
    if containedZone ≠ NULL ∧ accessNode ≠ NULL then
        foreach Nodes Adjacent to accessNode connect by "Association" do
            if endNode has label UserAttribute then
                usedAttributes= DFS in transpose of graph from endNode.
                if usedAttributes does not violates Separation of Duty then
                    output=True
                    Write used Attributes to file
                end
            end
        end
    end
end
else if taskNode ≠ NULL then
    if containedZone ≠ NULL ∧ accessNode ≠ NULL then
        foreach Nodes Adjacent to accessNode connect by "Association" do
            if endNode has label UserAttribute then
                usedAttributes= DFS in transpose of graph from taskNode
                if usedAttributes does not violates Separation of Duty then
                    output=True
                    Write used Attributes to file
                end
            end
        end
    end
end
write to PROC.txt access request, result and cause
return output
```

available resource. Our goal was to get the access request only on an appropriate object at appropriate zone and task which was what the result showed.

8.4.3 Performance of the algorithm

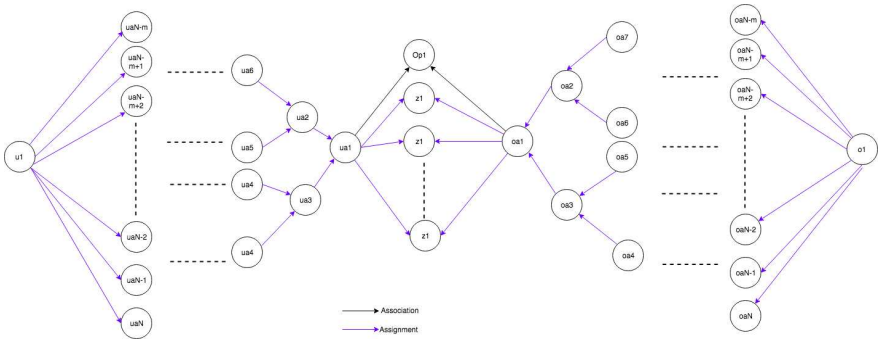
As we verified that the algorithm was working as expected for the real-world scenario that we have modeled, we wanted to look for the upper bounds for the performance of the algorithm. To do so, a custom graph was created, still following the specification but naming each element randomly. Also, four different experiments were performed to see the behavior with respect to the user, object, operation, and attributes.

1. *With respect to user*: To evaluate the performance of the algorithm with respect to the user, a base random graph was generated which contained few user attributes, object attributes that build a tree structure on each side of the graph. An operation node connected the root node from each user attribute to the object attribute. There was only one object node which connected to all the object attributes without any incoming edges. We added the user to the user attributes in a similar manner, one at a time, and record the time taken for an increasing number of users.

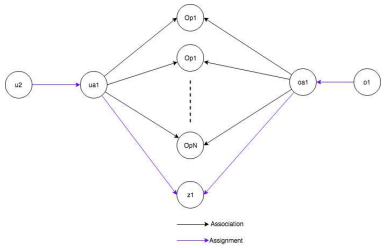
2. *With respect to object*: The authorization graph was built similar to the approach in ”with respect to the user" except the number of the user was one and the object was added to the graph every iteration and the time was recorded for access query to be processed.

3. *With respect to operation*: To measure the performance with respect to operation, we kept the number of the user, objects, and attributes constant and gradually increased the number of operation node by connecting new operation node connecting same user attribute and object attribute same iteration. This way all the operation nodes were reachable from all the users and object and, therefore, the worst case scenario.

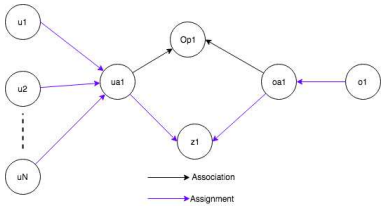
4. *With respect to attributes*: Similar to other approaches, we increased the number of attribute nodes in each iteration. Unlike others, to maintain the logical relationship and tree structure



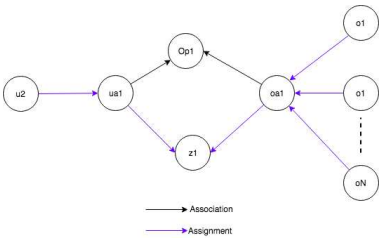
(a) With Respect To Attribute



(b) With Respect To Operation



(c) With Respect To User



(d) With Respect To Object

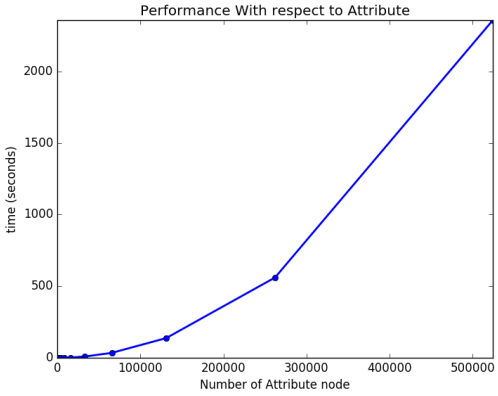
Figure 8.3: Experimental Setup

in either side of the authorization graph, the attributes grew by 2^n on each side where n was the number of iteration.

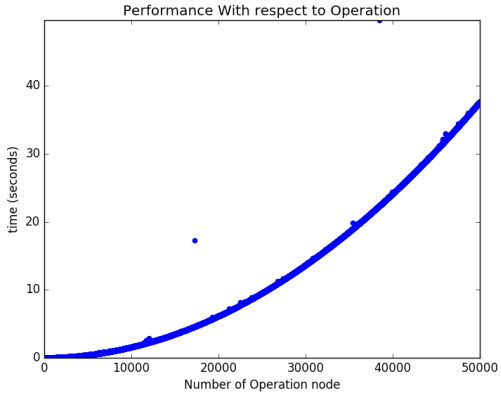
Figure 8.3 and Figure 8.5 show the experimental setups and their performance result, respectively.

8.4.4 Experimental Evaluation

The enforcement architecture as described in Figure 8.1 is our approach to keep the architecture general and implementable. All the modules acted as separate entities and communicated for total functionality. The communication protocol defined in Section 8.2 ensures that the communication whether via wired or wireless was secured against numerous attacks like eavesdropping, a man in the middle attack, replay, and modifications. However, we used a simple algorithm for hashing because of the contribution of the hashing technique to be more towards the consistency of the message being passed in terms of size than towards the security. Until the keys used for public key encryption were compromised, the hashing algorithm used was irrelevant to security. Therefore,

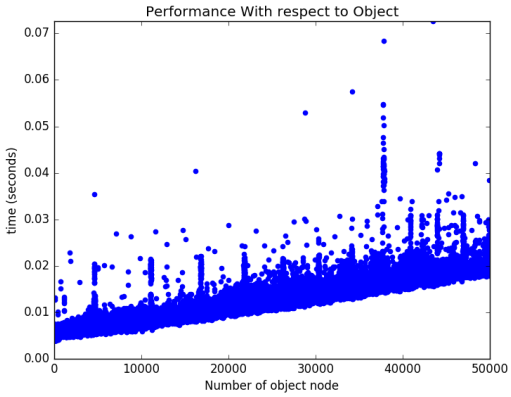


(a) Performance NX: With Respect To Attribute



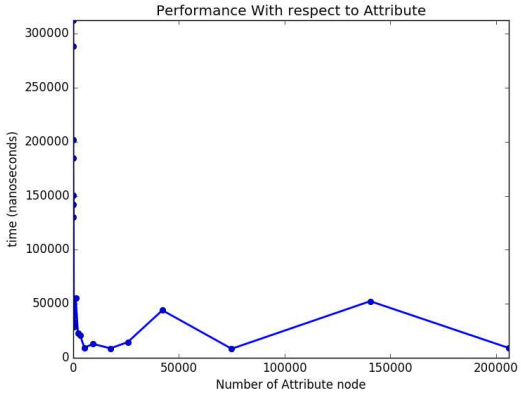
(b) Performance NX: With Respect To Operation

(c) Performance NX: With Respect To User

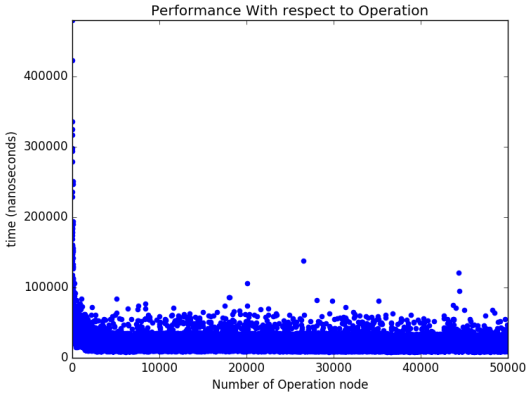


(d) Performance NX: With Respect To Object

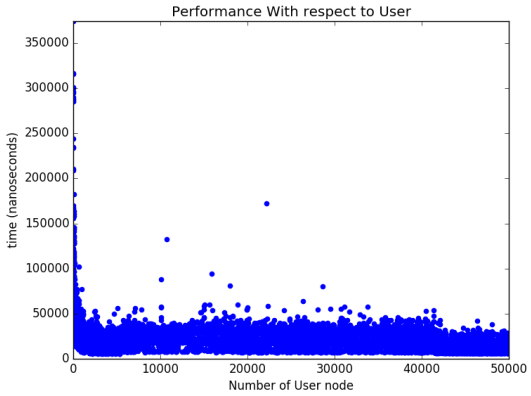
Figure 8.4: Performance Result using NetworkX



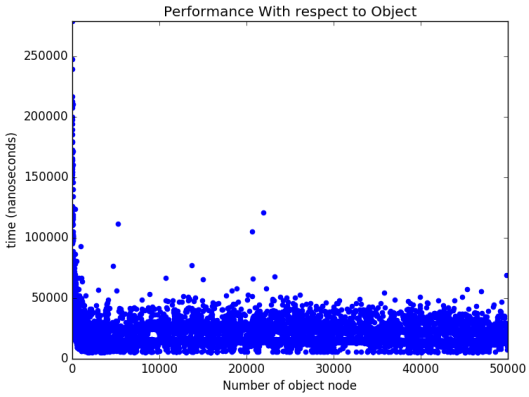
(a) Performance: With Respect To Attribute



(b) Performance: With Respect To Operation



(c) Performance: With Respect To User



(d) Performance: With Respect To Object

Figure 8.5: Performance Result

the combination tended to nurture the security perspective using the public key encryption and maintained a smaller size consistent payload. The protocol also aided in keeping the individual component independent of each other. Our work here just described a probable architecture for implementation of described task-based access control in mobile workflows. We have not implemented the actual architecture nor have performed any experiments to verify the feasibility and performance of the architecture itself. Therefore, the protocol itself was subject to change and open-ended to adaptation with the requirements of the system being implemented.

We used Neo4j for the representation of our authorization graph. As we have already mentioned, that the nature of our authorization graph resembled a Directed Acyclic Graph (DAG). The logical relationship in the authorization graph was better preserved with a graph data model. Additionally, as shown in a study [49] we could say that the graph databases showed better performance when the data were large and contained more relationships. On top of that, the study [50] had reported that Neo4j outperformed all the included graph databases in traversal payloads. The algorithm 2 was based on traversal of the graph which made Neo4j the most desirable platform.

The experimental setup was designed to explore the performance in the worst case scenarios. As our proposed model considered the location-time and different constraints, it was important to verify that those constraints were being considered in the computation of the access decision. In other words, the implementation of the model agreed with the definition of the model. Therefore, the algorithm was first verified for the correctness by using an example policy which was small and comparable. On the other hand, the performance could not be evaluated by a smaller example, thus, the random experimental setup as in Figure 8.3 was designed to represent the worst case authorization graph. The given algorithm 2 was a simple algorithm based on DFS. Theoretically, the time complexity of the algorithm was in square terms with respect to operation nodes and attribute nodes, user nodes and linear with respect to the object nodes. The complexity of the algorithm is shown in Figure 8.4 which was generated by replicating the algorithm 1 and the authorization graph is shown in Figure 8.2 using NetworkX package of python3. As our experiment was performed in the Neo4j platform where our algorithm was introduced as a plugin in the server, the

algorithm and data were located within the server. The result of our experiments as shown in Figure 8.5 shows a constant time complexity irrespective of the number of nodes of any type. The performance gain was due to the optimization done in the Neo4j itself. Furthermore, the native storage and processing of Neo4j made any traversal query less complex also shown by the result in Figure 8.5. As mentioned in [51], Neo4j followed an index-free adjacency leading to a low cost joins. The traversing of a relationship was O(1) in time. The advantage of using adjacency was more when the graph was traversed in the opposite direction. The native graph storage followed by Neo4j provided a supportive architecture for the processing. Each element was stored in different files. The graph structure was differentiated from property data to further add to the performance. Also, Neo4j tended to minimize the input-output operation by implementing the inline optimization, where a property data that could be encoded into the single record were directly in-lined in property store file instead of being pushed into the dynamic storage. Therefore, we could conclude that our approach of modeling the task-based access control with implementation architecture described in Figure 8.1 was feasible in the real-world system. Furthermore, the algorithm, despite being simple, performed very well with the combination of Neo4j platform.

Chapter 9

Conclusion and Future Work

9.1 Conclusion

Although organizations have been using a workflow management system to increase the efficiency of the business processes, the task has been more distributed and there has been a requirement of flexibility in the services provided. The inclusion of mobile devices has made the services flexible. At the same time, they have also introduced concerns about reliability and security. In order to have a flexible, secure, and manageable mobile service, there needs to be a proper system capable of controlling the access and flow of the tasks. Therefore, in this work, an architecture was proposed to incorporate the access control module and workflow management system together so that the access control policy could be used for management (handling the control flow) and enforcement (initialization and handling dynamic changes) of the workflow. The architecture that was proposed created an initial template and took feedback from the analyzer to correct any anomalies present. Similarly, to handle the dynamic situation like the change in location, the environmental feedback was used to perform the necessary changes. The access control module, which was a separate component of the architecture, was responsible for making all the authorization decisions. These decisions were very essential for the enforcement and continuation of the workflow plan. In addition, authorization decisions were also responsible for controlling the flow of the workflow. Therefore, the access control model played a very important role in our proposed system architecture.

The access decisions were made considering the workflow task, location, and time as well. Location and time were represented as an abstraction called “STZone” or spatio-temporal zone. The STZones were the logical mapping of the physical locations (i.e., represented in the form of longitudes and latitudes, at certain time instance or interval). The mapping function made use of containment and equality relations to compare and contrast between the two available logical

locations. The access control model itself was a hybrid of an existing framework and architecture called "Policy Machine" described in [52] and Role Based Access Control(RBAC). We introduced different attributes, relations, and constraints while following the specifications provided in [29] in an attempt to preserve the concept of a logical model. We combined it with the hierarchy and enable-disable feature of RBAC to handle the workflow management challenges. We also devised an algorithm to form an authorization graph given the model information.

The workflow authorization model was verified using TCPN. TCPN combined the strength of ordinary Petri Nets with high-level programming language and time, which made it more suitable for simulation of our mathematical model. The simulation was done by modeling the control flow between the processes from a real-world system called dengue decision support system. The constraints of DDSS (task assignment inclusive of time and location properties and other properties like separation of Duty) were modeled in coordination with the control flow. State space analysis was done on two types of model representation, to look into the structural and behavioral properties, namely, task assignment, Deadlock, violation of separation of duty constraint, violation of hierarchical constraints, and violation of cardinality constraints. The Hierarchical model represented the scenario as connected components, where each component was a portion of a bigger picture. The model was sliced using a slicing technique to mitigate the state space explosion, expand the search space, and be able to have a more effective and faster analysis. The sliced model, in contrast to the Hierarchical model, was not connected. Therefore, the verification of the model in the sliced model was based on the fact that, if the properties were verified in applicable nets, the output of the model could be safely used in other nets as an input. The approach reduced the size of the state space graph and, therefore, reduced the time taken for analysis. One thing to consider, however, was that the sliced model should mimic all the properties of the hierarchical model for a correct analysis.

The model checking helped to eradicate the inconsistencies in the model, but it was equally important to study the practicality of the model. It was critical to ensure that the model could be enforced for real system applications. We proposed an architecture to enforce our proposed

model into a system including mobile components. We devised the configuration between the systems involved and a communication protocol to ensure security. The proposed architecture contained different modules for distinct jobs and communicates with each other when required. The abstraction was obtained by having the resources module as the enforcement point and the access control module as the decision point. To make the decision, the access control module was provided with a DFS based algorithm to traverse the authorization graph and came up with an access control decision. We represented the authorization graph in Neo4j platform and the algorithm as its plugin. To verify the correctness of the algorithm, a sample authorization graph was created representing the DDSS system and system was queried for every possible combination of entities. The output was manually verified for the correctness. Similarly, the behavior of the algorithm with respect to the number of different entities in the policy was studied to analyze the performance in the real system. The performance evaluation showed a constant time behavior which made the system feasible for enforcement.

9.2 Future Work

In this work, we have made an approach to address the problem of modeling as well as the implementation of access control system dedicated to the management of the workflow systems. Although the system addressed the targeted problem, there existed an area of improvements in every sector. Firstly, we look forward to having a complete implementation. Currently, the implementation handles the access control but the architectures (both system architecture and enforcement architecture) are theoretical. We would like to explore the pros and cons of the architectures as well. The implementation of access control assumed that there existed a reliable administrative representation to develop and maintain the information required for the authorization graph. Since the authorization graph was an essential part of the whole system, we would like to explore the probability of introducing Artificial Intelligence (AI) for administrative actions and revoke a malfunctioning unit from the system. Also, even though our work here was dedicated to workflow management, the scope is not limited to it. We would like to investigate the generality of the model

and outreach different genres, for example, health care. Overall, we would like to see the limit of generalization and specialization of the model with completely enforced architecture.

Bibliography

[1] David Hollingsworth and UK Hampshire. Workflow management coalition: The workflow reference model. *Document Number TC00-1003*, 19:16, 1995.

[2] Ralf Salomon, Martin Lüder, and Gerald Bieber. ifall-a new embedded system for the detection of unexpected falls. In *Pervasive Computing and Communications (PerCOM) Workshops*, pages 286–291. Citeseer, 2010.

[3] Amiya K Maji, Arpita Mukhoty, Arun K Majumdar, Jayanta Mukhopadhyay, Shamik Sural, Soubhik Paul, and Bandana Majumdar. Security analysis and implementation of web-based telemedicine services with a four-tier architecture. In *Pervasive Computing Technologies for Healthcare, 2008. PervasiveHealth 2008. Second International Conference on*, pages 46–54. IEEE, 2008.

[4] Ravi S Sandhu, Edward J Coyne, Hal L Feinstein, and Charles E Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.

[5] Subhendu Aich, Samrat Mondal, Shamik Sural, and Arun Kumar Majumdar. Role based access control with spatiotemporal context for mobile applications. In *Transactions on Computational Science IV*, pages 177–199. Springer, 2009.

[6] Indrakshi Ray and Manachai Toahchoodee. A spatio-temporal role-based access control model. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 211–226. Springer, 2007.

[7] Subhendu Aich, Shamik Sural, and Arun K Majumdar. STARBAC: Spatiotemporal role based access control. In *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems"*, pages 1567–1582. Springer, 2007.

[8] Liang Chen and Jason Crampton. On spatio-temporal constraints and inheritance in role-based access control. In *Proceedings of the ACM Symposium on Information, Computer and Communications Security*, pages 205–216. ACM, 2008.

[9] Manachai Toahchoodee and Indrakshi Ray. On the formalization and analysis of a spatio-temporal role-based access control model. *Journal of Computer Security*, 19(3):399–452, 2011.

[10] D Richard Kuhn, Edward J Coyne, and Timothy R Weil. Adding attributes to role-based access control. *Computer*, 43(6):79–81, 2010.

[11] Vincent C Hu, David Ferraiolo, Rick Kuhn, Arthur R Friedman, Alan J Lang, Margaret M Cogdell, Adam Schnitzer, Kenneth Sandlin, Robert Miller, Karen Scarfone, et al. Guide to attribute based access control (abac) definition and considerations (draft). *NIST special publication*, 800(162), 2013.

[12] Xin Jin, Ram Krishnan, and Ravi Sandhu. A unified attribute-based access control model covering dac, mac and rbac. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 41–55. Springer, 2012.

[13] Michael Decker. A location-aware access control model for mobile workflow systems. *International Journal of Information Technology and Web Engineering (IJITWE)*, 4(1):50–66, 2009.

[14] Indrakshi Ray, Na Li, Robert France, and Dae-Kyoo Kim. Using uml to visualize role-based access control constraints. In *Proceedings of the ninth ACM symposium on Access control models and technologies*, pages 115–124. ACM, 2004.

[15] Karsten Sohr, Gail-Joon Ahn, Martin Gogolla, and Lars Migge. Specification and validation of authorisation constraints using uml and ocl. In *European Symposium on Research in Computer Security*, pages 64–79. Springer, 2005.

[16] Arjmand Samuel, Arif Ghafoor, and Elisa Bertino. A framework for specification and verification of generalized spatio-temporal role based access control model. 2007.

[17] Manachai Toahchoodee and Indrakshi Ray. Using alloy to analyse a spatio-temporal access control model supporting delegation. *IET Information Security*, 3(3):75–113, 2009.

[18] Basit Shafiq, Ammar Masood, James Joshi, and Arif Ghafoor. A role-based access control policy verification framework for real-time systems. In *Object-Oriented Real-Time Dependable Systems, 2005. WORDS 2005. 10th IEEE International Workshop on*, pages 13–20. IEEE, 2005.

[19] Hind Rakkay and Hanifa Boucheneb. Security analysis of role based access control models using colored petri nets and cpntools. In *Transactions on Computational Science IV*, pages 149–176. Springer, 2009.

[20] Christopher C Yang, Gondy Leroy, and Sophia Ananiadou. Smart health and wellbeing. *ACM Transactions on Management Information Systems (TMIS)*, 4(4):15, 2013.

[21] Suruchi Deodhar, Keith R Bisset, Jiangzhuo Chen, Yifei Ma, and Madhav V Marathe. An interactive, web-based high performance modeling environment for computational epidemiology. *ACM Transactions on Management Information Systems (TMIS)*, 5(2):7, 2014.

[22] M Lisa Yeo, Erik Rolland, Jackie Rees Ulmer, and Raymond A Patterson. Risk mitigation decisions for it security. *ACM Transactions on Management Information Systems (TMIS)*, 5(1):5, 2014.

[23] Xitong Guo, Sherry X Sun, and Doug Vogel. A dataflow perspective for business process integration. *ACM Transactions on Management Information Systems (TMIS)*, 5(4):22, 2015.

[24] Basit Shafiq, Arjmand Samuel, and Halima Ghafoor. A GTRBAC based system for dynamic workflow composition and management. In *Object-Oriented Real-Time Distributed Computing, 2005. ISORC 2005. Eighth IEEE International Symposium on*, pages 284–290. IEEE, 2005.

[25] Elisa Bertino, Piero Andrea Bonatti, and Elena Ferrari. TRBAC: A temporal role-based access control model. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):191–233, 2001.

[26] Gail-Joon Ahn, Ravi Sandhu, Myong Kang, and Joon Park. Injecting rbac to secure a web-based workflow system. In *Proceedings of the fifth ACM Workshop on Role-based access control*, pages 1–10. ACM, 2000.

[27] Peter Mell, James M Shook, and Serban Gavrila. Restricting insider access through efficient implementation of multi-policy access control systems. In *Proceedings of the 8th ACM CCS International Workshop on Managing Insider Security Threats*, pages 13–22. ACM, 2016.

[28] Arindam Roy, Shamik Sural, Arun Kumar Majumdar, Jaideep Vaidya, and Vijayalakshmi Atluri. On optimal employee assignment in constrained role-based access control systems. *ACM Transactions on Management Information Systems (TMIS)*, 7(4):10, 2017.

[29] David F Ferraiolo, Serban I Gavrila, and Wayne Jansen. Policy Machine: features, architecture, and specification. Technical report, 2015.

[30] Sejong Oh and Seog Park. Task–role-based access control model. *Information Systems*, 28(6):533–562, 2003.

[31] Sejong Oh and Seog Park. Task-role based access control (T-RBAC): an improved access control model for enterprise environment. In *International Conference on Database and Expert Systems Applications*, pages 264–273. Springer, 2000.

[32] Guoping Zhang and Jing Liu. A model of workflow-oriented attributed based access control. *International Journal of Computer Network and Information Security*, 3(1):47, 2011.

[33] James BD Joshi, Elisa Bertino, Usman Latif, and Arif Ghafoor. A generalized temporal role-based access control model. *IEEE Transactions on Knowledge and Data Engineering*, 17(1):4–23, 2005.

[34] Greg Milner. What is gps? *Journal of Technology in Human Services*, 34(1):9–12, 2016.

[35] John Whipple, William Arensman, and Marian Starr Boler. A public safety application of gps-enabled smartphones and the android operating system. In *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, pages 2059–2061. IEEE, 2009.

[36] Frank Sposaro and Gary Tyson. ifall: an android application for fall monitoring and response. In *Engineering in Medicine and Biology Society, 2009. EMBC 2009. Annual International Conference of the IEEE*, pages 6119–6122. IEEE, 2009.

[37] Axel Küpper. Location-based services. *Fundamental and operation, John Willey & Sons, Ltd*, 2005.

[38] Luca Mainetti, Luigi Patrono, and Ilaria Sergi. A survey on indoor positioning systems. In *Software, Telecommunications and Computer Networks (SoftCOM), 22nd International Conference on*, pages 111–120. IEEE, 2014.

[39] AKM Mahtab Hossain and Wee-Seng Soh. A survey of calibration-free indoor positioning systems. *Computer Communications*, 66:1–13, 2015.

[40] Kai Jansen, Nils Ole Tippenhauer, and Christina Pöpper. Multi-receiver gps spoofing detection: error models and realization. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pages 237–250. ACM, 2016.

[41] Todd E Humphreys, Brent M Ledvina, Mark L Psiaki, Brady W O'Hanlon, and Paul M Kintner Jr. Assessing the spoofing threat: Development of a portable gps civilian spoofer. In *Proceedings of the ION GNSS International Technical Meeting of the Satellite Division*, volume 55, page 56, 2008.

[42] Kurt Jensen and Lars M Kristensen. Timed coloured petri nets. In *Coloured Petri Nets*, pages 231–255. Springer, 2009.

[43] Kurt Jensen. *Coloured Petri nets: basic concepts, analysis methods and practical use*, volume 1. Springer Science & Business Media, 2013.

[44] Yahui Lu, Li Zhang, and Jiaguang Sun. Using colored petri nets to model and analyze workflow with separation of duty constraints. *The International Journal of Advanced Manufacturing Technology*, 40(1-2):179–192, 2009.

[45] Timed Color Petri Nets- Hierarhy (Top-down development). http://cpntools.org/2018/01/09/top-down-development. Accessed: 2018-06-15.

[46] Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. *The definition of standard ML: revised*. MIT press, 1997.

[47] Ramadan Abdunabi. *An access control framework for mobile applications*. PhD thesis, Colorado State University. Libraries, 2013.

[48] Justin J Miller. Graph database applications and concepts with neo4j. In *Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA*, volume 2324, page 36, 2013.

[49] Shalini Batra and Charu Tyagi. Comparative analysis of relational and graph databases. *International Journal of Soft Computing and Engineering (IJSCE)*, 2(2):509–512, 2012.

[50] Salim Jouili and Valentin Vansteenberghe. An empirical comparison of graph databases. In *Social Computing (SocialCom), 2013 International Conference on*, pages 708–715. IEEE, 2013.

[51] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph databases*. " O'Reilly Media, Inc.", 2013.

[52] David Ferraiolo, Vijayalakshmi Atluri, and Serban Gavrila. The policy Machine: A novel architecture and framework for access control policy specification and enforcement. *Journal of Systems Architecture*, 57(4):412–424, 2011.