Article

Instantiation and Implementation of HEAD Metamodel in an Industrial Environment: Non-IoT and IoT Case Studies

Nadine Kashmar [1,*], Mehdi Adda [2], Hussein Ibrahim [1], Jean-François Morin [3] and Tony Ducheman [3]

1 Centre de Recherche et d'Innovation en Intelligence Énergétique, 175 Rue de la Vérendrye, Sept-Îles, QC G4R 5B7, Canada; hussein.ibrahim@cegepsi.ca
2 Département de Mathématiques, Informatique et Génie, Université du Québec à Rimouski, 300 Allée des Ursulines, Rimouski, QC G5L 3A1, Canada; mehdi_adda@uqar.ca
3 Institut Technologique de Maintenance Industrielle, 175 Rue de la Vérendrye, Sept-Îles, QC G4R 5B7, Canada; jean-francois.morin@itmi.ca (J.-F.M.); tony.ducheman@itmi.ca (T.D.)
* Correspondence: nadine.kashmar@cegepsi.ca

Abstract: Access to resources can take many forms: digital access via an onsite network, through an external site, website, etc., or physical access to labs, machines, information repositories, etc. Whether access to resources is digital or physical, it must be allowed, denied, revoked, or disabled using robust and coherent access control (AC) models. What makes the process of AC more complicated is the emergence of digital transformation technologies and pervasive systems such as the internet of things (IoT) and industry 4.0 systems, especially with the growing demand for transparency in users' interaction with various applications and services. Controlling access and ensuring security and cybersecurity in IoT and industry 4.0 environments is a challenging task. This is due to the increasing distribution of resources and the massive presence of cyber-threats and cyber-attacks. To ensure the security and privacy of users in industry sectors, we need an advanced AC metamodel that defines all the required components and attributes to derive various instances of AC models and follow the new and increasing demand for AC requirements due to continuous technology upgrades. Due to the several limitations in the existing metamodels and their inability to answer the current AC requirements, we have developed a Hierarchical, Extensible, Advanced, Dynamic (HEAD) AC metamodel with significant features that overcome the existing metamodels' limitations. In this paper, the HEAD metamodel is employed to specify the needed AC policies for two case studies inspired by the computing environment of Institut Technologique de Maintenance Industrielle (ITMI)-Sept-Îles, QC, Canada; the first is for ITMI's local (non-IoT) environment and the second for ITMI's IoT environment. For each case study, the required AC model is derived using the domain-specific language (DSL) of HEAD metamodel, then Xtend notation (an expressive dialect of Java) is utilized to generate the needed Java code which represents the concrete instance of the derived AC model. At the system level, to get the needed AC rules, Cypher statements are generated and then injected into the Neo4j database to represent the Next Generation Access Control (NGAC) policy as a graph. NGAC framework is used as an enforcement point for the rules generated by each case study. The results show that the HEAD metamodel can be adapted and integrated into various local and distributed environments. It can serve as a unified framework, answer current AC requirements and follow policy upgrades. To demonstrate that the HEAD metamodel can be implemented on other platforms, we implement an administrator panel using VB.NET and SQL.

Keywords: access control; metamodel; security and privacy; policy; DSL; cybersecurity; digital transformation; IoT; industry 4.0; NGAC; Neo4j; graph database; Cypher query language

check for updates

Citation: Kashmar, N.; Adda, M.; Ibrahim, H.; Morin, J.-F.; Ducheman, T. Instantiation and Implementation of HEAD Metamodel in an Industrial Environment: Non-IoT and IoT Case Studies. *Electronics* **2023**, *12*, 3216. https://doi.org/10.3390/electronics12153216

Academic Editor: Cheonshik Kim

Received: 30 June 2023
Revised: 21 July 2023
Accepted: 21 July 2023
Published: 25 July 2023

1. Introduction

Continuous advancement in technology in general and the internet of things (IoT) in particular has increased the need for security and privacy. This fact imposes organizations and industry sectors to rethink how to control access to their logical and physical resources through modern and enhanced access control (AC) approaches [1–3]. AC is defined as the process of restricting access to resources based on a predefined set of rules known as AC policies. It consists of two main stages: authentication and authorization. Authentication is the process where an unknown subject (or user), after verifying his identity, becomes a known user. This process is insufficient to protect resources. Authorization is the process of allowing/denying authenticated users access to a resource after checking the regulations and policies of an organization or industry sector [4]. It includes the following phases [5,6]:

- defining a security policy (set of AC rules);
- selecting an AC model that matches the defined policy;
- implementing the model and enforcing the defined AC rules.

In any organization or industry sector, one of the essential administrative responsibilities is to set policy rules to control access to objects accessed and administered in a domain. To mitigate security threats accompanied by information technologies, AC rules must consider possible threats in a domain. They should be concise and easy to understand so that everyone can follow the guidelines they set forth. AC rules outline the way a domain works with information and resources, and once they are created the domain's AC policy is customized [7,8]. In today's digital environment, IoT and industry 4.0, users need effective protection for their resources distributed everywhere, and their ability to access information anytime, and from anywhere [9–12]. Consequently, the evolution of AC policy languages should follow the development of computing environments and information systems (IS) [13,14]. Developing an effective policy language is increasingly challenging due to the highly dynamic and heterogeneous structures of the current networking generation [5,15,16]. AC policy language should respond to the increasing demand for authorizing and controlling access to resources. It should express the organization's set of rules derived from business requirements, plans, guidelines, etc.

There are several common AC models implemented in various centralized and distributed computing environments to control access to resources. These include discretionary access control (DAC), mandatory access control (MAC), role-based access control (RBAC), and attribute-based access control (ABAC) [17–19]. As well, hybrid models, extended models, and abstracted models are implemented to enhance the features of common models as well as to enable the definition of a larger set of AC policies. A recent topic in this area is AC metamodels. They usually describe generic concepts and serve as unifying frameworks. AC models are instances of AC metamodels that are used to specify, define, and explain what, how, and when particular objects (resources) can be accessed by given individuals (subjects) [15,18,20]. Due to the ubiquitous nature of current computing environments, particularly industry 4.0, it is essential that advanced AC metamodels be developed to meet the increasing demand for AC requirements. In light of the limitations of the existing AC metamodels [6,18], we propose a Hierarchical, Extensible, Advanced and Dynamic (HEAD) AC metamodel [15]. Compared to the other proposed works in this domain, e.g., [21–23], HEAD metamodel considers the challenging requirements which accompany the various technology progressions, for example, the need (1) to create hierarchical AC components that fit the diverse hierarchical organizational structures, (2) to define and upgrade a larger set of AC policies, (3) to have several AC solutions within the same organization (e.g., local network, IoT, and cloud), and many other challenges [20]. The domain-specific language (DSL) [24] of the HEAD metamodel—which is defined using Eclipse (xtext) [25,26]—is generic enough to include the heterogeneity of AC models, this allows defining larger sets of AC policies. It supports the hierarchy of all components to conform to organizational hierarchy. The derived models of the HEAD metamodel can be extended to follow policy updates. Also, it is dynamic since various components/attributes can be added to follow technology progressions and include several AC solutions in an organization.

In this paper, for the validation of the HEAD metamodel, we propose two case studies that include different conditions (e.g., static and dynamic AC policies, different contexts, etc.), and technical aspects (e.g., local and distributed devices, sensors, etc.) to show how our proposed metamodel can provide the necessary AC requirements and how it can be adapted to different computing environments. The existing case studies are extremely limited and barely contain the necessary elements we need to demonstrate our solution. In two case studies, we demonstrate our solution in the computing environment of Institut technologique de maintenance Industrielle (ITMI) of Sept-Îles, Quebec. The first case study deals with ITMI's local computing environment (in this paper, we refer to it as a non-IoT environment), while the second case study deals with their IoT environment. The first case study deals with ITMI's local computing environment (in this paper, we refer to it as a non-IoT environment), while the second case study deals with their IoT environment. Hence, we use the DSL of HEAD metamodel presented and explained in [15] to derive the AC models required for each case study. The derived models of each case are encoded using Eclipse Xtend notation (an expressive dialect of Java) to represent the concrete instance of AC policies and generate the needed Java code to express the AC rules as Cypher queries. As an enforcement point and to verify the accuracy and the coherence of the concrete instances of AC policies, Cypher queries are injected into the Neo4j database to represent the generated rules in the form of Next Generation Access Control (NGAC) which is developed by National Institute of Standards and Technology (NIST) [27,28] policy graph—the objects, the relationships between them, and the subjects that interact with the system in a form that adheres to the semantics of ITMI. NGAC policy graph is constructed with the basic policy elements and a fixed set of relationships [27,29]. The NGAC policy elements represented in the NGAC policy graph are illustrated in Figure 1:

- the authenticated users (U), and their user attributes (UA).
- the objects (O) that need to be accessed, and their attributes (OA).
- the policy class(es) (PC).
- the assignment relationships between U–UA, UA–UA, O–OA, OA–OA, UA–OA, and UA–OA to the appropriate PC(s).
- the association relationships to define the access rights associated with some UAs to perform operations on some objects specified by OAs.

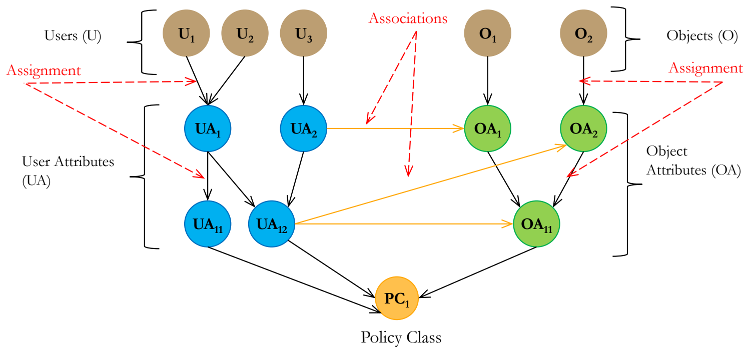


Figure 1. The NGAC policy Graph

Both case studies show that the HEAD AC metamodel can be adapted to meet AC requirements in non-IoT and IoT environments. However, compared to already published works in this domain, this paper's contribution can be summarized as follows:

- Presents two industrial case studies illustrating the implementation of a new and advanced metamodel (named HEAD metamodel) from the definition of AC rules to the enforcement of policies. None of the existing works in this domain present a real case study or address all the needed phases to enforce the defined AC rules.
- Provides modern AC tools and shows how they can be adapted to different computing environments, especially industrial environments. Others in this field have only provided some explanations for manually illustrating some simple examples. They lack a detailed illustration of the tools available and how to enforce AC rules.
- It helps researchers in the domain to understand how AC metamodels work in real industrial contexts (or any computing environment), from the definition of informal

policies to their enforcement. Compared to other works they only focus on one phase and confuse issues that cut across multiple phases. The originality of our work is demonstrated through the validation of two case studies on a real subject and related explanations. Also, it can be used as a centerpiece for researchers to tackle other industrial case studies in the domain.

The remainder of this paper is organized as follows: Section 2 reviews some of the related work in this domain. A summary of the HEAD metamodel and its characteristics is explained in Section 3. To illustrate the relevance of our metamodel, the subject of study is described in detail in Section 4, then the first case study—ITMI's local (non-IoT) environment—is presented in Section 5, and the second case study—ITMI's IoT environment—is presented in Section 6. In Section 7, we show another example of how the HEAD metamodel can be implemented using VB.NET and SQL databases to generate AC rules. In Section 8 we explain the evaluation and validation of the HEAD metamodel. Section 9 concludes this paper with future perspectives.

2. Related Works

In the field of information security, the main concern for organizations and industry sectors is to keep the secrecy of their information and prevent illegal access to their logical and physical resources, especially in the presence of cyber-criminals and cyber-attacks [30,31]. Currently, resources are no longer located within local areas, they are distributed on different sites and need to be accessed via various private and public networks. The emergence of evolving technologies and digital transformation urges organizations to protect their private and public networks, especially with the evolution of Industry 4.0 and the IoT concept [32–34]. In this context, various AC models and metamodels are implemented in different computing environments to protect resources and information from unauthorized access. Unfortunately, they have limited features and are insufficient to meet the current AC requirements and follow the needed technology upgrades [17,20]. In this section, we review some of the proposed AC methods in this domain. We summarize the existing AC metamodels [17,18,20] proposed to instantiate different AC models.

To prevent insider threats in organizations a function-based AC method inspired by functional encryption is proposed in [35]. It stores access authorizations as a three-dimensional tensor where users can invoke authorized commands at different levels such as data segments. They are authorized to use a command on an object and forbidden to use it on another object. In [36], Qi et al. propose a scalable industry data AC system for the RFID-enabled supply chain to provide an item-level data AC mechanism that defines and enforces AC policies based on both the participants' role attributes and the products' RFID tag attributes. For industrial automation and control systems (IACS) and similar automation systems of the smart energy grid, an eXtended Access Control Markup Language (XACML)-based AC system is described by Ruland et al. [37] to protect connected devices and associated safety-relevant settings from unauthorized access. Their proposed AC method is composed of a two-stage AC schema. They first evaluate policies based on XACML, then use information about the system's behavior to prevent malicious or accidental operations from having a negative impact on system stability. The system design and implementation consider safety requirements (e.g., timing requirements, the availability…) to enable integration in safety-critical environments.

For secure information sharing in an IIoT, in [13], Ulltveit-Moe et al. investigate how to secure information sharing with external vendors, and they identify the necessary security requirements in this domain. Also, they propose a roadmap to improve security in IIoT which investigates short-term and long-term solutions for IIoT devices. The former is mainly based on integrating existing appropriate practices such as firewalls, intrusion detection systems, etc. The latter outlines a long-term solution with fine-grained AC for sharing data between external entities that supports cloud-based data storage. Ray et al. [29] show how AC can be specified using an ABAC model for remote healthcare monitoring (RHM). They present healthcare AC requirements using an example, then explain how

NGAC can be used for specifying AC policies. Alagar et al. [38] propose a context-sensitive RBAC (CRABC) scheme for securing the environment of the healthcare IoT industry which is composed of large-scale connectivity of medical devices, patients, physicians, etc., with the vastness of information collection and sharing. CRBAC combines roles, attributes, and context to formulate context constraints to enforce access and flow controls. Another AC model for the IoT in healthcare, named Enhanced Context-aware Capability based AC (ECCAPAC), is proposed by Ahamed et al. [39] to make the medical network resilient against crypto attacks by taking into account the trust value of the object based on relevance and node importance.

Despite that the proposed AC models in various industrial computing environments come up with solutions for dedicated scenarios or case studies, they have limited features compared to the increasing demand for security and privacy in the current computing environments. The current reality of networking environments imposes the need to define new components/attributes for the existing AC models in order to upgrade the existing AC policies and allow specifying and enforcing a larger set of static and dynamic AC policies. As with this feature, the proposed models do not consider hierarchy defining multiple levels of components (e.g., role hierarchy).

The evolution of pervasive ISs and intelligent manufacturing has had a substantial influence on the future of the industry. Industry 4.0 (or IIoT) is the modernization of conventional manufacturing through modern technology. In smart industries, several physical and cyber technologies are integrated to improve productivity, quality, performance, and management in the age of IIoT [40]. All of this raises the challenge to develop AC metamodels that serve as unifying frameworks for deriving advanced AC models able to follow technology upgrades. This would allow for defining and enforcing a larger set of static and dynamic AC policies. In [20], we summarize the development stages of AC methods starting from common AC models, hybrid models, extending AC models, abstracting AC models, and reaching AC metamodels which is the recent research issue in this domain. As described in [17,18] some metamodels are proposed as generic by using AC features of some common models, some other metamodels are proposed as hybrid metamodels by combining features of some AC models, and some other metamodels are proposed as metamodel extensions for some existing metamodels. The proposed metamodels provide some development approaches in the field, but they have several limitations since (1) they are not generic enough to derive the needed AC models (common models, hybrid models, and other models); (2) they are not flexible and not dynamic enough to follow technology upgrades; (3) they are not extensible where the derived models and the defined policies cannot be extended; (4) they do not support the hierarchy feature for all components (e.g., role, action, context, etc.); (5) collaboration and interoperability between AC models are not addressed; and (6) they do not consider the concept of migrating AC policies from one model to another. To address these limitations, we have designed and developed the HEAD metamodel [15] where its advanced features allow deriving different models (existing and even non-existent models). In this paper, we use the HEAD metamodel to derive the needed AC models for two case studies inspired by ITMI's environment in order to show how it can be adapted to various industrial and organizational computing environments. We also show how various components and attributes for static and dynamic AC policies can be defined in addition to the common ones, and to illustrate how it supports the component hierarchy.

3. HEAD Metamodel

In [15], we propose a hierarchical, extensible, advanced, and dynamic AC metamodel, named HEAD metamodel. This metamodel has advanced features compared to other AC metamodels proposed in the literature. Its distinct design and the new opportunities it opens in the domain are described in [20]. In this section, we briefly explain its meta-components (or meta-classes) and features to show how it can be employed to derive the needed AC models for two case studies (explained in Sections 5 and 6) inspired by ITMI's non-IoT and IoT computing environments, then generate the needed AC rules to enforce them. The meta-components (or meta-classes) of HEAD metamodel are the EXPLICIT (E_x), IMPLICIT (I_m), and SETTING (S_t). As shown in Figure 2 [15], E_x represents entities/classes within an organization or industry sector, such as subjects and objects. I_m represents the described entities/classes. I_m includes AUTHORIZATION UNIT (AU) entities/classes such as roles, security levels, etc., and PROCEDURAL UNIT (PU) entities/classes such as actions, permissions, etc. S_t represents the concepts needed to have more regulated access to resources, such as entities/classes of context, contextual conditions, etc. The relationships between HEAD metamodel components can be described as follows:

- between E_x and AU is to allow assigning zero or many (0..*), for example, subjects to roles, groups, etc.;
- between AU and PU, and PU and E_x is to describe which AUs (e.g., groups) can perform zero or many (0..*) PUs (e.g., actions) on some other E_x (e.g., objects);
- E_x and I_m could have zero or many (0..*) S_t (e.g., condition) to fulfill access requests;
- the self-association edge on each meta-class is to allow formulating hybrid models by associating, for example, AUs with other AUs, PUs with other PUs, etc.;
- the aggregation association for each meta-class allows the creation of hierarchies of the derived component(s).

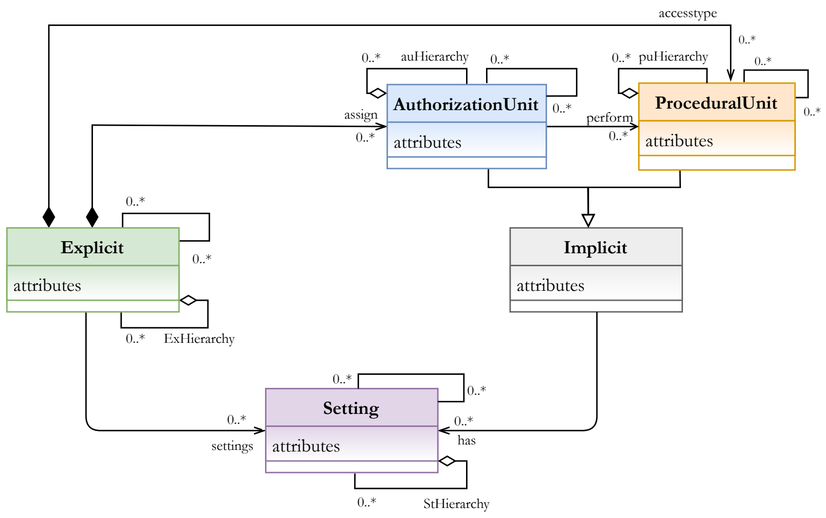


Figure 2. The HEAD Metamodel [15].

The DSL of the HEAD metamodel is explained with examples in [15], it can be used to derive various AC models (common AC, hybrid AC, and other models). For example, to derive the MAC model, subject and object classes must be instantiated from E_x, security level must be instantiated from AU, and operation class must be instantiated from PU. Another example, is the RBAC model where subject and object classes must be derived from E_x, role class from AU, and permission and action classes from PU. Using the HEAD metamodel, access to resources is determined by the attributes of the defined entities in the access request including subject, object, action, context, etc. It allows enforcing complex AC policies that involve an arbitrary combination of attributes with static, dynamic, and relationship values. Also, it allows deriving models of high granularity for access policies, fitting the various needs of AC requirements. This extends beyond the proposed AC metamodels in the literature [15,20].

4. The Subject of Study: Technological Institute for Industrial Maintenance (ITMI)

ITMI is a technology transfer center affiliated with the Cégep de Sept-Îles, QC, Canada specializing in industrial maintenance. In an industrial field where machines/devices reliability is of strategic importance, ITMI offers tailor-made support to North Shore and Quebec industries and provides technical services to reduce downtime, minimize maintenance costs, optimize productivity, lessen training costs, and increase success rates. Since its creation in 2008, ITMI has continued to grow and expand its fields of expertise. This has been done to adapt to industries' needs on the one hand and to technological changes on the other. Research efforts are centered on industry 4.0, IoT, embedded systems, artificial intelligence, energy intelligence, computer-aided design, and others. ITMI has laboratories with extensive research infrastructure and regularly updated machines/devices including industrial machinery, hydraulic and pneumatic, prototyping and 3D printing, and drones. A variety of projects are accomplished in these laboratories, which are accessed by workers in the departments of design and maintenance, digital audit, IoT and embedded systems, and information technology (IT). About 35 employees work in different departments at ITMI with expertise in artificial intelligence, informatics, machine learning, and other fields.

ITMI is a sub-division of the Department of Research and Innovation (DRI) at Sept-Îles and is directed by the main director of DRI. In addition to the main director, there are four types of workers (users) in each department: manager, adviser, specialist, and technician. Figure 3 illustrates the hierarchy of roles at ITMI. Each department manager, who is directed by the DRI director, supervises the advisers of each department. Advisers direct specialists and technicians. Experts, specialists, and technicians can share their expertise among departments based on ongoing projects. Ensuring the privacy of users and the security of resources is essential for ITMI since any exposure or intrusion to logical (e.g., the local and public databases) or physical (e.g., labs, machines, etc.) resources could result in serious consequences such as financial loss, and loss of reputation. Considering that ITMI's main role is to follow up, handle, and find solutions related to projects for other institutions, any intervention creates the possibility of ITMI's entire data being compromised by a single action. Hence, ITMI needs an AC method that answers all the needed requirements based on different situations and conditions. Also, AC methods must always be updated due to the nature of the given projects which are related to recent and different technologies.

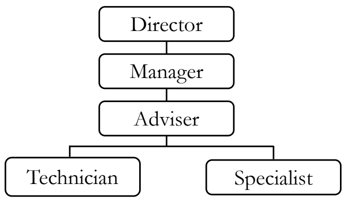


Figure 3. ITMI - Role heirarchy

In Sections 5 and 6, we present two case studies to show how the HEAD metamodel can be implemented to derive the required models that fit ITMI's AC requirements in its non-IoT (local) and IoT environments, generate the needed AC policies, then enforce them. Figure 4 summarizes the implementation phases of the HEAD metamodel, and the used tools to write and generate the necessary code for each phase. However:

- The HEAD metamodel: as illustrated in the figure below, the HEAD metamodel development steps are: unifying the heterogeneous components, meta-classes and their relationships, the DSL grammar definition of HEAD metamodel using Eclipse Xtext, and the meta-policy expression are explained in detail in [15].
- Phase 1: after investigating and identifying the AC policy parameters of each case study, in this phase we run the Runtime-Eclipse (xtext) then use the defined DSL grammar of the HEAD metamodel (of the previous phase) to specify and derive the required AC model(s) and express the AC policy(s).
- Phase 2: to represent the concrete instance of the derived AC model, Xtend notation is used to generate the required Java code at the system level. A case study's AC policy details are configured at this level (e.g., the existing resources in an organization, the constraints to access some resources, etc.). Since our aim in this paper is to use NGAC as an enforcement point and get NGAC authorization responses, we generate Cypher statements as Java output. We inject them into the Neo4j database to represent the NGAC policy graph.

- Phase 3: this phase enforces AC policy through Cypher queries written in Neo4j as NGAC inputs and NGAC authorization responses.

Note that all of the above phases are applied and presented in detail in each case study.

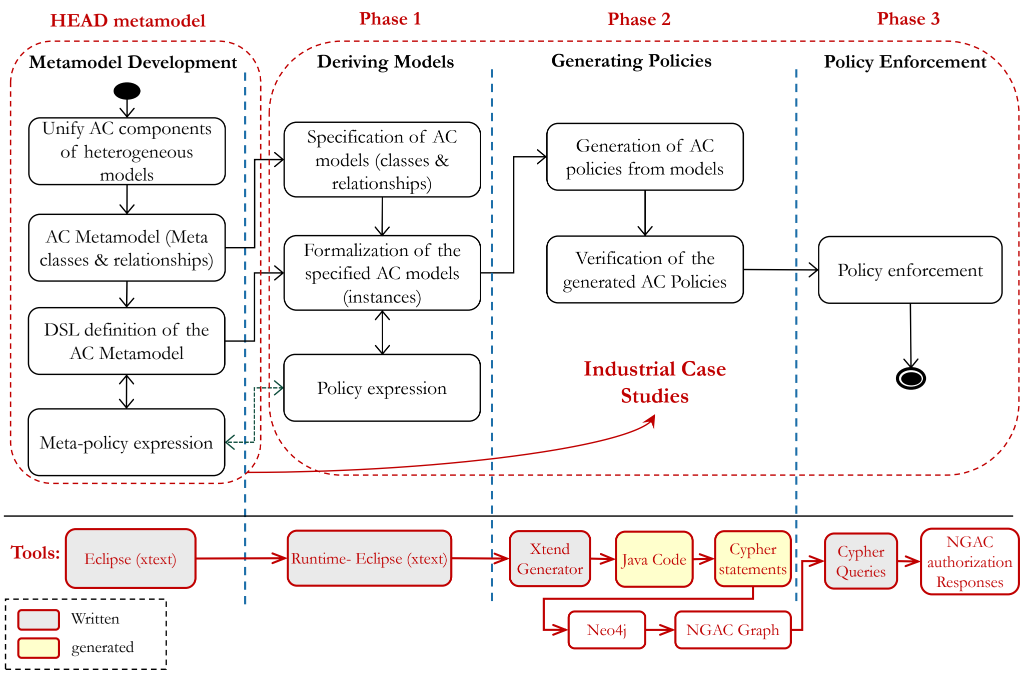

HEAD metamodel
Phase 1
Phase 2
Phase 3
Metamodel Development
Deriving Models
Generating Policies
Policy Enforcement
Unify AC components of heterogeneous models
AC Metamodel (Meta classes & relationships)
DSL definition of the AC Metamodel
Meta-policy expression
Specification of AC models (classes & relationships)
Formalization of the specified AC models (instances)
Policy expression
Generation of AC policies from models
Verification of the generated AC Policies
Industrial Case Studies
Policy enforcement
Tools:
Eclipse (xtext)
Runtime- Eclipse (xtext)
Xtend Generator
Java Code
Cypher statements
Neo4j
NGAC Graph
Cypher Queries
NGAC authorization Responses
Written
generated

Figure 4. The Case studies: The implementation phases and the used tools.

5. Case Study 1—ITMI: Non-IoT

In this case study, a rail robot needs to be implemented for a North Quebec Rail (NQR) project. To achieve it a set of tasks required to be implemented by a team. Figure 5 illustrates ITMI's local environment system architecture. To complete project tasks within the start and end dates, all users must be authenticated using passwords, fingerprints, or PIN codes to access the database or labs. Authenticated users are authorized, after checking the defined AC rules in the policy database, to access resources and perform operations according to their roles, groups, and other attributes of users.

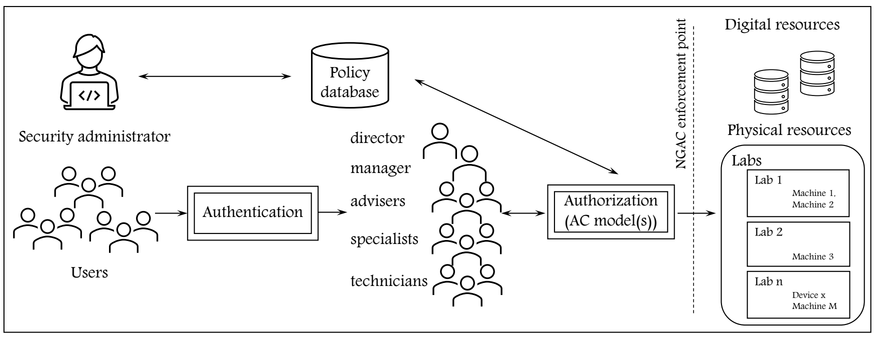


Figure 5. The system architecture of ITMI: non-IoT environment.

According to the project requirements, some users might be assigned to a certain role, while others may also be assigned to different groups, and some may share some tasks with each other. Access rights grant access to users with a certain role/group to some resources and allow them to perform some actions on the database (read, write, update . . .), and on machines (switching on/off, monitoring, modifying settings . . .). Access decisions could be allowed or denied based on the defined policy. However, in the following we describe the guidelines to perform the project tasks:

- The NQR project is administered by Roy, ITMI's director, who has full access to databases and labs. He manages the priority of project tasks and directs the manager in setting up the appropriate methodologies to implement them. Before running the project, he must confirm all details and tasks.
- According to the director's guidelines, Thomas, the manager who supervises advisers and has access to the database tables Projects and Tasks, creates the project record and sets the necessary tasks. Also, he updates the status of each project task (in progress, completed, on hold . . .) depending on the project's performance.
- The advisers, John and Sophia, choose groups of specialists and technicians based on their expertise. Then they assign to each group the necessary tasks, and determine which machines or equipment are required. They have full access to both the GroupTasks and Requirements tables in the database.
- Specialists are researchers with specialized knowledge in a particular field who work with technicians to complete projects. Technicians perform technical tasks such as maintenance, machines/devices installation, etc. For the NQR project, specialists and technicians have access to the AI lab where the Rail Robot machine is located, and to the 3D lab where the 3D printing device exists. To perform some of the required tasks we have the following groups:
  - GroupA: The specialists, Bob and Cathy, test the robot, then assess and write the obtained results into table Results. The technician, Peter, assists them in troubleshooting the robot and hardware/software problems.
  - GroupB: The specialists, Bob and Marc, read/analyze the results to check if any flaw(s) exist and might affect the rail robot's performance. The technician, Eva, operates the robot to match the analyzed results to the robot's performance.
  - GroupC: The specialists, Marc and Cathy, redesign and implement another prototype for the flawed part(s). The needed part(s) are created using a 3D printer. The technicians, Peter and Eva, help them replace damaged or flawed part(s), then verify robot system compatibility.

Advisers, specialists, and technicians are allowed to access resources during the work hours via private network. Their access rights must be revoked once the project is over.

5.1. The Challenge

ITMI frequently has ongoing projects with unexpected AC requirements. In this context, ITMI needs to ensure the rights to access data, device(s), etc., are provided to right user(s), especially since users assigned to the same role might have different tasks since they might also be assigned to different groups. Users' permissions need to be specified based on user-role, and user-group assignments. The solution must be scalable to accommodate ITMI's future growth.

5.2. The Solution: HEAD Metamodel

As described in Figure 4, the DSL grammar of HEAD metamodel is used to derive various instances of E_x, I_m (AUs and PUs), and S_t entities and answer the AC requirements of each case study. By investigating the above case study, we can find that:

- E_x = {subject (name, . . .); object (title, . . .)}.
- I_m are AUs = {role (type, . . .); group (number, . . .)}, and PUs = {permission (ptype, . . .); action (type, . . .)}
- S_t = {context-constraint (date, time . . .); constraint (projconfirm, task_status . . .)}.

5.2.1. Phase 1: Deriving Models

In this section, we investigate the best AC model that fits the AC requirements of this case study. Due to the above E_x, I_m, and S_t entities, we have the possibilities:

- RBAC model: RBAC entities are subject, object, role, permission, and action. Users are assigned to roles and groups, and in some situations we have attributes and environmental conditions. Hence, the RBAC model is not enough to satisfy the AC requirements, since we cannot find all the necessary entities.

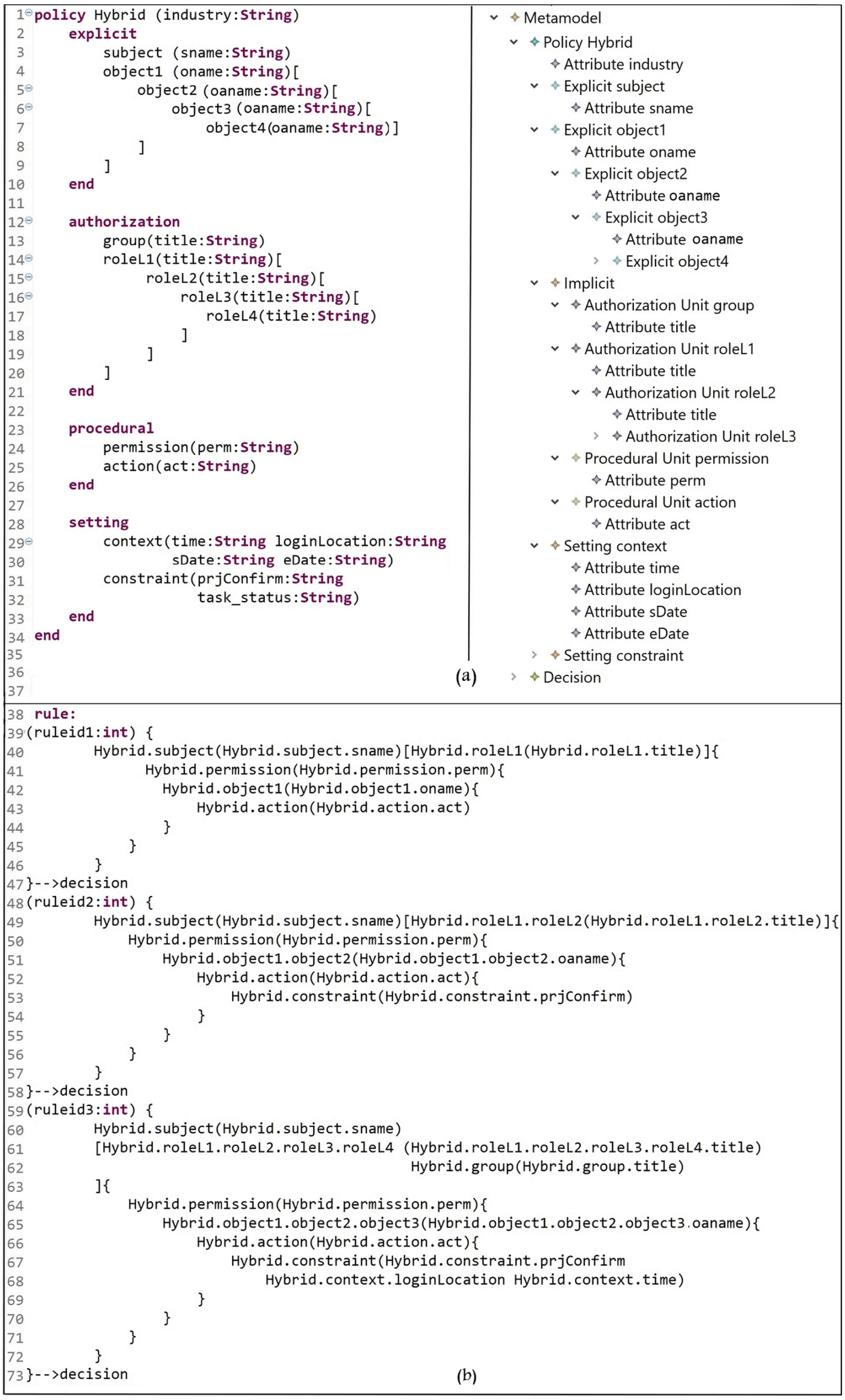
- ABAC model: ABAC entities are subject, object, action, and environmental (context) attributes. ITMI users are assigned to roles and groups. Therefore, with the ABAC model, the RBAC model should be considered, in addition to the group entity since users are assigned to roles and/or groups.
- Hybrid model: the notion of role reflects the importance of considering the RBAC model; the notion of attributes and the need to express dynamic AC rules reflect the importance of taking into account the ABAC model; also the group entity with its attributes imposes the need not to restrict the solution to RBAC or ABAC models only. Hence, the solution should consider future entities that need to be created/added due to upgrades.

Therefore, our solution in this case is to derive a hybrid model based on user-groups, RBAC, and ABAC. Using the DSL of the HEAD metamodel the formal user-groups, RBAC, and ABAC hybrid model can be derived by first instantiating the needed entities of E_x, AU, PU, and S_t, then expressing the rules as shown in Figure 6. In Figure 6a:

- line 1: the specification of policy model/class.
- lines 2 to 10: the block of creating E_x entities, starts and ends with the keywords explicit and end. Line 3: the creation of subject entity with the attribute(s). Lines 4 to 9: four levels of objects are created to represent the hierarchy of resources. We actually have three levels of object hierarchy, but since we use NGAC as an enforcement framework we create additional level to have four levels. 'object1' level expresses the main objects assigned to different object containers, and the other three levels are defined to represent the hierarchy object containers.
- lines 12 to 21: the block of creating AU entities, starts and ends with the keywords authorization and end. Line 13: the creation of group entity with the attribute(s). Lines 14 to 20: four levels of role hierarchy are created. For example, role1 refers to the director, role2 refers to the manager, etc.
- Lines 23 to 26: the block of creating PU entities, starts and ends with the keywords procedural and end. Line 24: the creation of permission entity with the needed attribute(s). Line 25: the creation of action entity with the attribute(s).
- Lines 28 to 33: the block of creating S_t entities, starts and ends with the keywords setting and end. Lines 29 and 30: the creation of context entity with the context attribute(s). Lines 31 and 32: the creation of constraint entity with the attribute(s).

Moreover, in Figure 6b we provide examples of three possible rule expressions:

- Lines 39 to 47: this rule expression is, for example, to express access rights for users who are assigned to the first-level of role hierarchy (e.g., director) and implicitly connected with the actions associated with their role (e.g., manager) without the administrator having to explicitly list the manager actions on objects, and this is also applied at the lower levels of hierarchy.
- Lines 48 to 58: to express, for example, the rule for users designated as assigned to the second-level of role hierarchy who are implicitly associated with their role(s) and perform some actions on objects with some constraints.
- Lines 59 to 73: the rule is expressed, for example, for users with lower-level roles, and who are also assigned to a specific group, to perform certain actions on certain objects of a certain level in a particular context under certain constraints.



```
policy Hybrid (industry:String)
    explicit
        subject (sname:String)
        object1 (oname:String)[
            object2 (oaname:String)[
                object3 (oaname:String)[
                    object4(oaname:String)]
            ]
        ]
    end

    authorization
        group(title:String)
        roleL1(title:String)[
            roleL2(title:String)[
                roleL3(title:String)[
                    roleL4(title:String)
                ]
            ]
        ]
    end

    procedural
        permission(perm:String)
        action(act:String)
    end

    setting
        context(time:String loginLocation:String
                sDate:String eDate:String)
        constraint(prjConfirm:String
                   task_status:String)
    end
end
```

- Metamodel
  - Policy Hybrid
    - Attribute industry
    - Explicit subject
      - Attribute sname
    - Explicit object1
      - Attribute oname
      - Explicit object2
        - Attribute oaname
        - Explicit object3
          - Attribute oaname
          - Explicit object4
    - Implicit
      - Authorization Unit group
        - Attribute title
      - Authorization Unit roleL1
        - Attribute title
        - Authorization Unit roleL2
          - Attribute title
          - Authorization Unit roleL3
      - Procedural Unit permission
        - Attribute perm
      - Procedural Unit action
        - Attribute act
    - Setting context
      - Attribute time
      - Attribute loginLocation
      - Attribute sDate
      - Attribute eDate
    - Setting constraint
  - Decision

(a)

```
rule:
(ruleid1:int) {
        Hybrid.subject(Hybrid.subject.sname)[Hybrid.roleL1(Hybrid.roleL1.title)]{
              Hybrid.permission(Hybrid.permission.perm){
                Hybrid.object1(Hybrid.object1.oname){
                    Hybrid.action(Hybrid.action.act)
                }
            }
        }
}-->decision
(ruleid2:int) {
        Hybrid.subject(Hybrid.subject.sname)[Hybrid.roleL1.roleL2(Hybrid.roleL1.roleL2.title)]{
            Hybrid.permission(Hybrid.permission.perm){
                Hybrid.object1.object2(Hybrid.object1.object2.oaname){
                    Hybrid.action(Hybrid.action.act){
                        Hybrid.constraint(Hybrid.constraint.prjConfirm)
                    }
                }
            }
        }
}-->decision
(ruleid3:int) {
        Hybrid.subject(Hybrid.subject.sname)
        [Hybrid.roleL1.roleL2.roleL3.roleL4 (Hybrid.roleL1.roleL2.roleL3.roleL4.title)
                                             Hybrid.group(Hybrid.group.title)
        ]{
            Hybrid.permission(Hybrid.permission.perm){
                Hybrid.object1.object2.object3(Hybrid.object1.object2.object3.oaname){
                    Hybrid.action(Hybrid.action.act){
                        Hybrid.constraint(Hybrid.constraint.prjConfirm
                            Hybrid.context.loginLocation Hybrid.context.time)
                    }
                }
            }
        }
}-->decision
```

(b)

Figure 6. Case Study 1: (**a**) A hybrid model based on user-groups, RBAC and ABAC entities/attributes; (**b**) rule expressions.

5.2.2. Phase 2: Generating Policies

The meta-policy is expressed in terms of the meta-components E_x, I_m, and S_t [15,20]:

$$Metapolicy = \langle E_x, AU, PU, S_t \rangle$$

Before expressing the formal policy, we need to describe the policy elements:

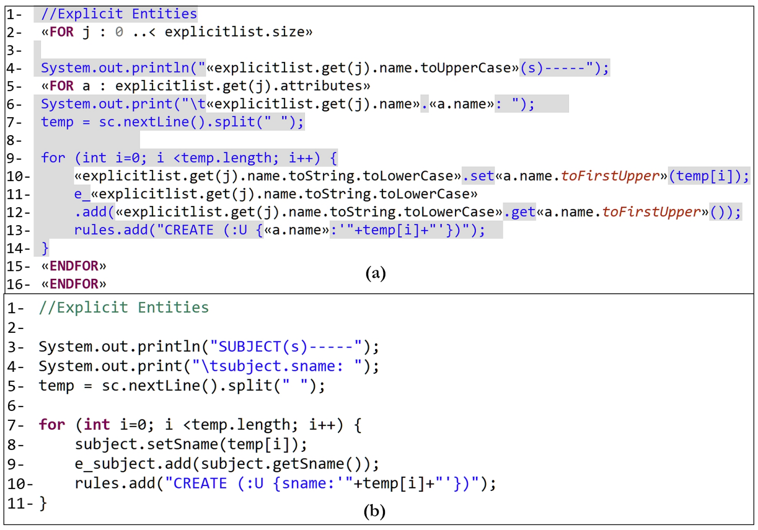
- Users: a set of entities who have accounts in the IS. Users = {Roy, Thomas, John, Sophia, Marc, Bob, Cathy, Peter, Eva}
- User Attributes: each user is associated with a set of attributes (e.g., Id, name . . .).
- Roles: specifies user's role. Role = {director, manager, adviser, specialist, technician}.
- Groups: specifies user's group. Group = {groupA, groupB, groupC}.
- Objects: a set of logical/physical objects that need protection. For example, database entities (projects, tasks . . .), labs, machines, devices, etc.
- Object attributes: object attributes include the project name, machine#, etc.
- Actions: a set of actions on logical objects (e.g., read, write, update, delete, etc.), and on physical objects (e.g., operate machines, test, troubleshoot, etc.).
- Permissions: users' permissions to perform actions on objects based on their roles (or groups), and if constraints and contextual constraints are true.
- Context: for a given context—for example, log in via private/public network, machine malfunction, system failure, etc.— context attributes include datetime, login location, failed password attempts, etc., contextual constraint expressions include context attributes which are important for authorization decisions.
- Constraints: expressions include the other various types of attributes, for example, subject attributes, object attributes, etc.

Based on the meta-policy expression, we express the policy of this case study as follows:

$$\begin{aligned} Policy = \langle & subject(name, \ldots); object(title, \ldots); role(type, \ldots); group(number, \ldots); \\ & permission(ptype, \ldots); action(type, \ldots); context(date, \ldots); \\ & constraints(roletype, title, \ldots) \rangle \end{aligned}$$

To represent the concrete instance of the derived model, we use Xtend notation to transform the user-groups, RBAC, and ABAC model into Java code. Hereafter, we need to generate Cypher statements as Java output to be injected into the Neo4j database to represent NGAC policy (as illustrated in Figure 4). In Figure 7a, we show an example of the written Xtend code to transform the E_x entities into subjects, the Cypher statement is expressed in line 13 to create users (U) nodes. Cypher statements are expressed and added to the array list of 'rules'. The same concept is also applied to transform the AU, PU, and S_t entities. Moreover, the sample code in Figure 7a generates the sample Java code in Figure 7b to define subject entities. A Cypher statement to create U nodes is generated in line 10.

In this case study we use three types of assignment relationships. The first is 'assigned_to' where some Us are assigned to some UA containers, Os are assigned to some OA containers, and some OAs are assigned to some some other OA containers. The second is 'include' to represent the hierarchy of objects (Os and OAs). The third is 'has_child_content' to represent the hierarchy of roles (UAs). In Figure 8a, an example of Xtend notation is illustrated expressing the Cypher statements to create the 'assigned_to' relationship for object O at root level with OA container (lines 2–4), and 'include' relationship to represent the hierarchy of objects, for example, O–OA or OA–OA containers (lines 6–8). In line 2, of t he generated Java expressions in Figure 8b, r_object1 refers to objects at the root level and h_object2 in line 7 refers to the second level of object hierarchy.



```
//Explicit Entities
«FOR j : 0 ..< explicitlist.size»

System.out.println("«explicitlist.get(j).name.toUpperCase»(s)-----");
«FOR a : explicitlist.get(j).attributes»
System.out.print("\t«explicitlist.get(j).name».«a.name»: ");
temp = sc.nextLine().split(" ");

for (int i=0; i <temp.length; i++) {
    «explicitlist.get(j).name.toString.toLowerCase».set«a.name.toFirstUpper»(temp[i]);
    e_«explicitlist.get(j).name.toString.toLowerCase»
    .add(«explicitlist.get(j).name.toString.toLowerCase».get«a.name.toFirstUpper»());
    rules.add("CREATE (:U {«a.name»:'"+temp[i]+"'})");
}
«ENDFOR»
«ENDFOR»
```

(a)

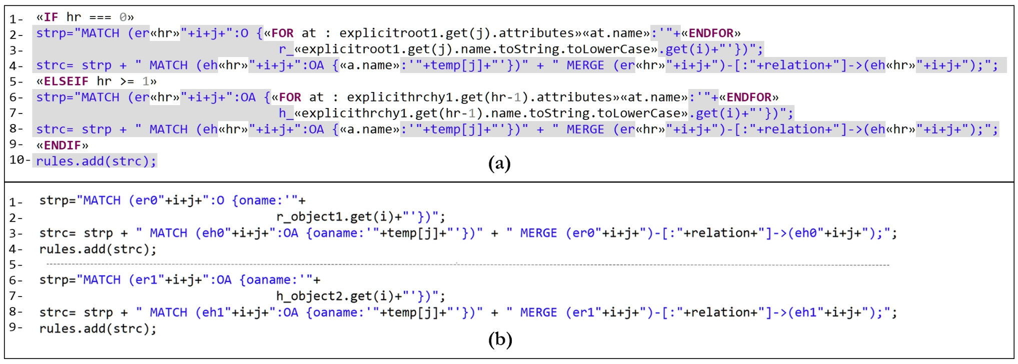
```
//Explicit Entities

System.out.println("SUBJECT(s)-----");
System.out.print("\tsubject.sname: ");
temp = sc.nextLine().split(" ");

for (int i=0; i <temp.length; i++) {
    subject.setSname(temp[i]);
    e_subject.add(subject.getSname());
    rules.add("CREATE (:U {sname:'"+temp[i]+"'})");
}
```

(b)

Figure 7. Case Study 1: (**a**) A sample of Xtend notation for Explicit entities; (**b**) the generated Java code for the subject entities.



```
«IF hr === 0»
strp="MATCH (er«hr»"+i+j+":O {«FOR at : explicitroot1.get(j).attributes»«at.name»:'"+«ENDFOR»
                                 r_«explicitroot1.get(j).name.toString.toLowerCase».get(i)+"'})";
strc= strp + " MATCH (eh«hr»"+i+j+":OA {«a.name»:'"+temp[j]+"'})" + " MERGE (er«hr»"+i+j+")-[:"+relation+"]->(eh«hr»"+i+j+");";
«ELSEIF hr >= 1»
strp="MATCH (er«hr»"+i+j+":OA {«FOR at : explicithrchy1.get(hr-1).attributes»«at.name»:'"+«ENDFOR»
                                 h_«explicithrchy1.get(hr-1).name.toString.toLowerCase».get(i)+"'})";
strc= strp + " MATCH (eh«hr»"+i+j+":OA {«a.name»:'"+temp[j]+"'})" + " MERGE (er«hr»"+i+j+")-[:"+relation+"]->(eh«hr»"+i+j+");";
«ENDIF»
rules.add(strc);
```

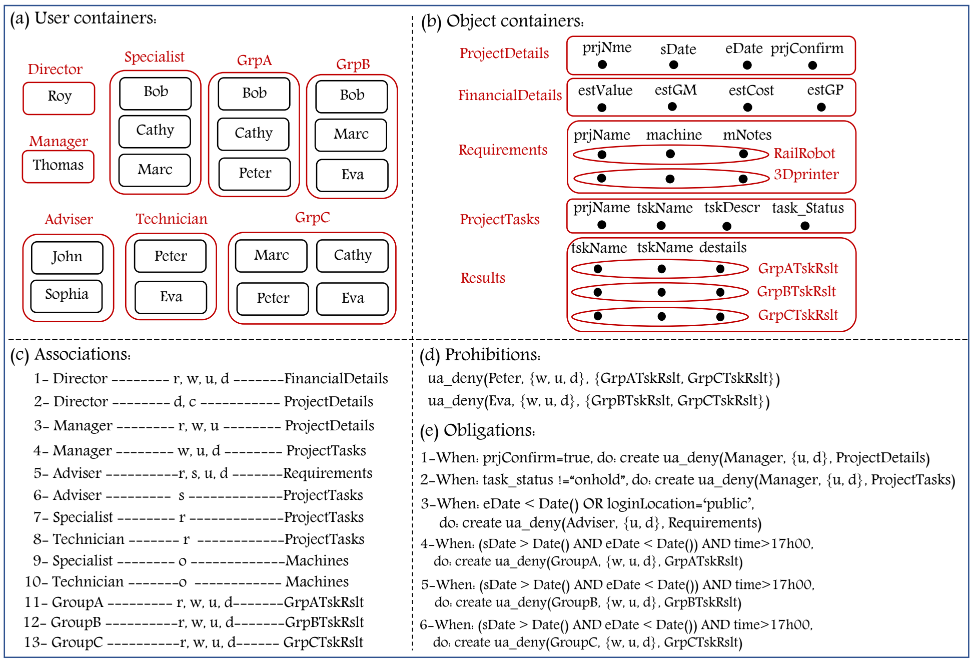
(a)

```
strp="MATCH (er0"+i+j+":O {oname:'"+
                                 r_object1.get(i)+"'})";
strc= strp + " MATCH (eh0"+i+j+":OA {oaname:'"+temp[j]+"'})" + " MERGE (er0"+i+j+")-[:"+relation+"]->(eh0"+i+j+");";
rules.add(strc);
 ---------------------------------------------------------------------------------------------------------------------
strp="MATCH (er1"+i+j+":OA {oaname:'"+
                                 h_object2.get(i)+"'})";
strc= strp + " MATCH (eh1"+i+j+":OA {oaname:'"+temp[j]+"'})" + " MERGE (er1"+i+j+")-[:"+relation+"]->(eh1"+i+j+");";
rules.add(strc);
```

(b)

Figure 8. Case Study 1: A sample of (**a**) Xtend notation to express the assignment relationships between O–OA nodes; (**b**) The generate the Java code expressing the Cypher statements.

As mentioned earlier, to illustrate the NGAC policy graph, we need to generate the needed Cypher statements as Java output. To obtain the required Cypher statements, the system administrator (or the authorized user) must be aware of how to configure the NGAC policy by specifying the users, resources, relationships, conditions, etc., based on the computing environment of the case study. Figure 9 illustrates the schema of NGAC policy configuration of this case study. In Figure 9a, we show user containers that represent the assignment of U–UA. For example, John (U) and Sophia (U) are assigned to Adviser (UA), and Roy (U) is assigned to Director (UA), etc. In (b), we show object containers with the representation of relational database tables with some distinguished rows/columns where they are represented as containers of data objects corresponding to the row/column fields. For example, a container named ProjectDetails includes the fields prjName, startDate, endDate, and prjConfirmation. Furthermore, (c) Lists thirteen association relations in terms of the user and object attributes/containers expressed in (a,b). In (c), users' access rights to perform operations are formulated through associations. For example, in (1) and (2) the director is allowed to {r:read, w:write/ insert, u:update, d:delete} the data in FinancialDetails container. He is also allowed to {d: delete, c: confirm} data of ProjectDetails container; in (3) the manager is allowed to {r, w, u} data of ProjectDetails; etc. Note that the operations 's' and 'o' in 6 and 9 mean select and operate. In (d), we have two prohibition relations which express user attribute-deny (ua_deny) to deny the technicians 'Peter' and 'Eva' from performing {w, u, d} on GrpATskRslt/GrpCTskRslt and GrpBTskRslt/GrpCTskRslt respectively. In (e), we represent six obligations defined as event–response relations to define constraints under which policy state data are obligated to change. For example, in (1) when the director confirms the ProjectDetails information, the manager would not be able to {u, d} this information; another example in (3) expresses an obligation due to some context where the adviser is not allowed to {u, d} information in Requirements container if he logged into the system via a public network or if the current date is greater than the end date of NQR project; also in (4, 5, and 6) users of Groups A, B, and C are not allowed to {w, u, d} information of containers GrpATskRslt, GrpBTskRslt, and GrpCTskRslt during the non-business hours and before and after the start and end of project dates.



(c) Associations:

```
 1- Director -------- r, w, u, d -------FinancialDetails
 2- Director -------- d, c ----------- ProjectDetails
 3- Manager -------- r, w, u -------- ProjectDetails
 4- Manager -------- w, u, d -------- ProjectTasks
 5- Adviser ----------r, s, u, d -------Requirements
 6- Adviser ---------  s -------------ProjectTasks
 7- Specialist -------- r -------------ProjectTasks
 8- Technician ------- r  ------------ProjectTasks
 9- Specialist -------- o -------------Machines
10- Technician -------o  ------------ Machines
11- GroupA --------- r, w, u, d-------GrpATskRslt
12- GroupB ----------r, w, u, d-------GrpBTskRslt
13- GroupC ----------r, w, u, d------ GrpCTskRslt
```

(d) Prohibitions:

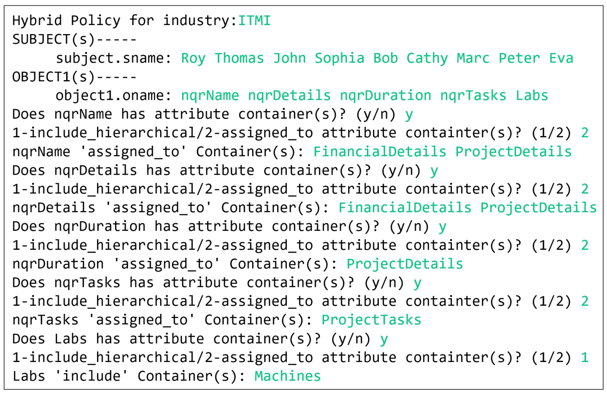
ua_deny(Peter, {w, u, d}, {GrpATskRslt, GrpCTskRslt})
ua_deny(Eva, {w, u, d}, {GrpBTskRslt, GrpCTskRslt})

(e) Obligations:

1-When: prjConfirm=true, do: create ua_deny(Manager, {u, d}, ProjectDetails)
2-When: task_status !="onhold", do: create ua_deny(Manager, {u, d}, ProjectTasks)
3-When: eDate < Date() OR loginLocation='public',
do: create ua_deny(Adviser, {u, d}, Requirements)
4-When: (sDate > Date() AND eDate < Date()) AND time>17h00,
do: create ua_deny(GroupA, {w, u, d}, GrpATskRslt)
5-When: (sDate > Date() AND eDate < Date()) AND time>17h00,
do: create ua_deny(GroupB, {w, u, d}, GrpBTskRslt)
6-When: (sDate > Date() AND eDate < Date()) AND time>17h00,
do: create ua_deny(GroupC, {w, u, d}, GrpCTskRslt)

Figure 9. Case Study 1: NGAC Policy Configuration.

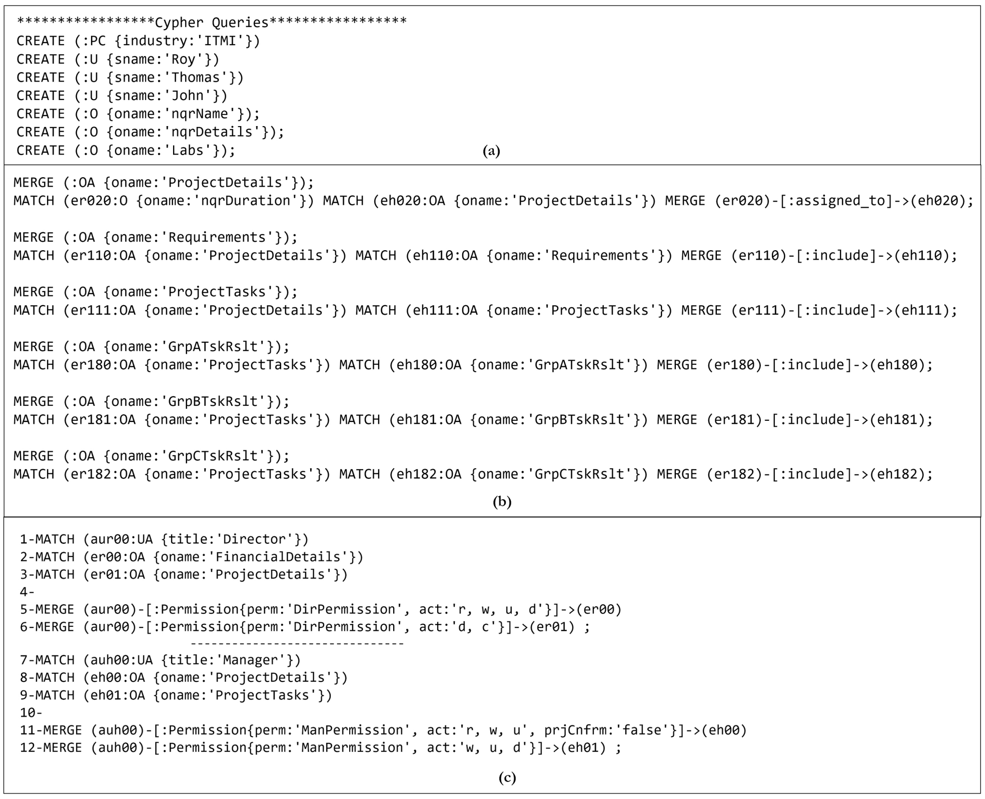
Figure 10 shows a sample of Java output to configure the required AC policy. The green text demonstrates the configuration of the subject and object entities and attributes that is performed by a system administrator based on the schema from Figure 9, indicating the hierarchical relationship between the object containers. For example, the object 'nqrDuration' is assigned_to 'ProjectDetails' container, the object 'Labs' has a hierarchical relation with (or includes) the 'Machines' container, and so on. After configuring the policy elements, Figure 11a shows a sample of the generated Cypher statements to create policy class (PC), e.g., create (:PC {industry: 'ITMI'}); users (U), e.g., create (:U {sname: 'Roy'}); objects (O), e.g., create (:O {oname: 'Labs'}), etc. Figure 11b shows examples of Cypher statements representing O–OA and OA–OA assignments, and OA hierarchies. For example, the first statement is to create 'ProjectDetails' node, then assign 'nqrDuration' node to the created node. The second and third statements means that the object 'ProjectDetails' at the root level 'include' the 'Requirements' and 'ProjectTasks' nodes which are at the second level of object hierarchy. The remaining statements mean that 'ProjectTasks' node at level 2 'include' the nodes 'GrpATskRslt', 'GrpBTskRslt', and 'GrpCTskRslt' which are at the third level. Note that the semicolon ';' indicates the end of a statement.



```
Hybrid Policy for industry:ITMI
SUBJECT(s)-----
     subject.sname: Roy Thomas John Sophia Bob Cathy Marc Peter Eva
OBJECT1(s)-----
     object1.oname: nqrName nqrDetails nqrDuration nqrTasks Labs
Does nqrName has attribute container(s)? (y/n) y
1-include_hierarchical/2-assigned_to attribute containter(s)? (1/2) 2
nqrName 'assigned_to' Container(s): FinancialDetails ProjectDetails
Does nqrDetails has attribute container(s)? (y/n) y
1-include_hierarchical/2-assigned_to attribute containter(s)? (1/2) 2
nqrDetails 'assigned_to' Container(s): FinancialDetails ProjectDetails
Does nqrDuration has attribute container(s)? (y/n) y
1-include_hierarchical/2-assigned_to attribute containter(s)? (1/2) 2
nqrDuration 'assigned_to' Container(s): ProjectDetails
Does nqrTasks has attribute container(s)? (y/n) y
1-include_hierarchical/2-assigned_to attribute containter(s)? (1/2) 2
nqrTasks 'assigned_to' Container(s): ProjectTasks
Does Labs has attribute container(s)? (y/n) y
1-include_hierarchical/2-assigned_to attribute containter(s)? (1/2) 1
Labs 'include' Container(s): Machines
```

Figure 10. Case Study 1: A sample of Java output with the configuration of subjects, objects, etc.

The Cypher statements in Figure 11c expresses the association relationships that define the access rights for some UAs (e.g., Director, Manager . . .) over some OAs (e.g., FinancialDetails, ProjectDetails . . .). For example, lines 1–3 and lines 7–9 match the UAs 'Director' and 'Manager', and the OAs 'FinancialDetails', 'ProjectDetails', and 'ProjectTasks'. Then, lines 5–6 express that the director with DirPermission is able to perform {r, w, u, d} operations on 'FinancialDetails' and {d, c} on 'ProjectDetails', and lines 11–12 express that the manager with ManPermission is able to perform {r, w, u} operations on 'ProjectDetails' as long as the information is not confirmed by the director (prjConfirm = 'false'). Since the basic elements of NGAC are U, O, UA, OA, and assignment/association relationships, we configure the constraints and contextual constraints as properties in the association relationship. As a result of configuring the NGAC policy and injecting the generated Cypher statements into Neo4j, we can display the NGAC policy graph in Figure 12.



```
*****************Cypher Queries*****************
CREATE (:PC {industry:'ITMI'})
CREATE (:U {sname:'Roy'})
CREATE (:U {sname:'Thomas'})
CREATE (:U {sname:'John'})
CREATE (:O {oname:'nqrName'});
CREATE (:O {oname:'nqrDetails'});
CREATE (:O {oname:'Labs'});
```
(a)

```
MERGE (:OA {oname:'ProjectDetails'});
MATCH (er020:O {oname:'nqrDuration'}) MATCH (eh020:OA {oname:'ProjectDetails'}) MERGE (er020)-[:assigned_to]->(eh020);

MERGE (:OA {oname:'Requirements'});
MATCH (er110:OA {oname:'ProjectDetails'}) MATCH (eh110:OA {oname:'Requirements'}) MERGE (er110)-[:include]->(eh110);

MERGE (:OA {oname:'ProjectTasks'});
MATCH (er111:OA {oname:'ProjectDetails'}) MATCH (eh111:OA {oname:'ProjectTasks'}) MERGE (er111)-[:include]->(eh111);

MERGE (:OA {oname:'GrpATskRslt'});
MATCH (er180:OA {oname:'ProjectTasks'}) MATCH (eh180:OA {oname:'GrpATskRslt'}) MERGE (er180)-[:include]->(eh180);

MERGE (:OA {oname:'GrpBTskRslt'});
MATCH (er181:OA {oname:'ProjectTasks'}) MATCH (eh181:OA {oname:'GrpBTskRslt'}) MERGE (er181)-[:include]->(eh181);

MERGE (:OA {oname:'GrpCTskRslt'});
MATCH (er182:OA {oname:'ProjectTasks'}) MATCH (eh182:OA {oname:'GrpCTskRslt'}) MERGE (er182)-[:include]->(eh182);
```
(b)

```
1-MATCH (aur00:UA {title:'Director'})
2-MATCH (er00:OA {oname:'FinancialDetails'})
3-MATCH (er01:OA {oname:'ProjectDetails'})
4-
5-MERGE (aur00)-[:Permission{perm:'DirPermission', act:'r, w, u, d'}]->(er00)
6-MERGE (aur00)-[:Permission{perm:'DirPermission', act:'d, c'}]->(er01) ;
                      ------------------------------
7-MATCH (auh00:UA {title:'Manager'})
8-MATCH (eh00:OA {oname:'ProjectDetails'})
9-MATCH (eh01:OA {oname:'ProjectTasks'})
10-
11-MERGE (auh00)-[:Permission{perm:'ManPermission', act:'r, w, u', prjCnfrm:'false'}]->(eh00)
12-MERGE (auh00)-[:Permission{perm:'ManPermission', act:'w, u, d'}]->(eh01) ;
```
(c)

Figure 11. Case Study 1: sample of Cypher statements for (**a**) PC, U, and O nodes; (**b**) O–OA assignments and object hierarchies; (**c**) access rights as AU–OA associations.

5.2.3. Phase 3: NGAC—The Policy Enforcement Point

In this section, we explain how the AC policy is enforced over Cypher queries issued from a system that uses Cypher statements as NGAC inputs (we use Neo4j) and can generate NGAC authorization responses based on those queries. In Figure 12, in the left section the nodes shown in light and dark green color illustrate the assignment of users (Us) to their roles/groups (UAs), in addition to the role hierarchy which is indicated by the red arrows and 'has_child_content' relationship. The nodes indicated in light and dark blue color in the right section of the graph represent the assignment of objects (Os) to object containers (OAs) (and OAs-OAs), in addition to object hierarchy where the hierarchy of object containers is represented by the red arrows and 'include' relationship. The association relationships, the yellow arrows, represent users' permissions based on their roles/groups. Note that in this paper we use the term 'has_child_content' to indicate the hierarchy of roles, and the term 'include' to represent the hierarchy of objects.

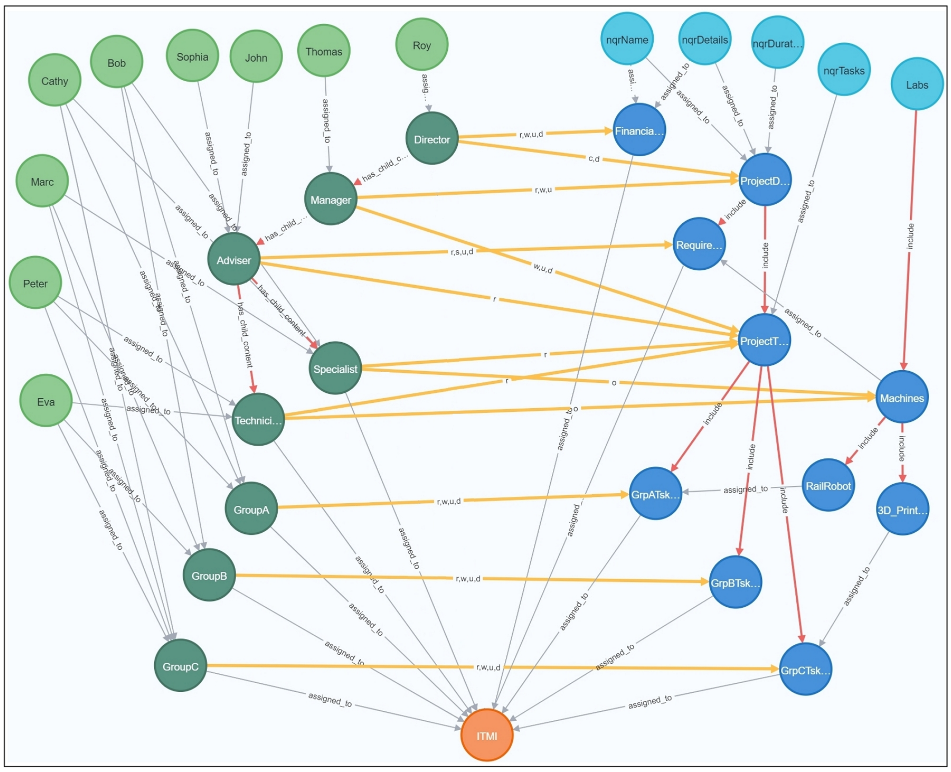
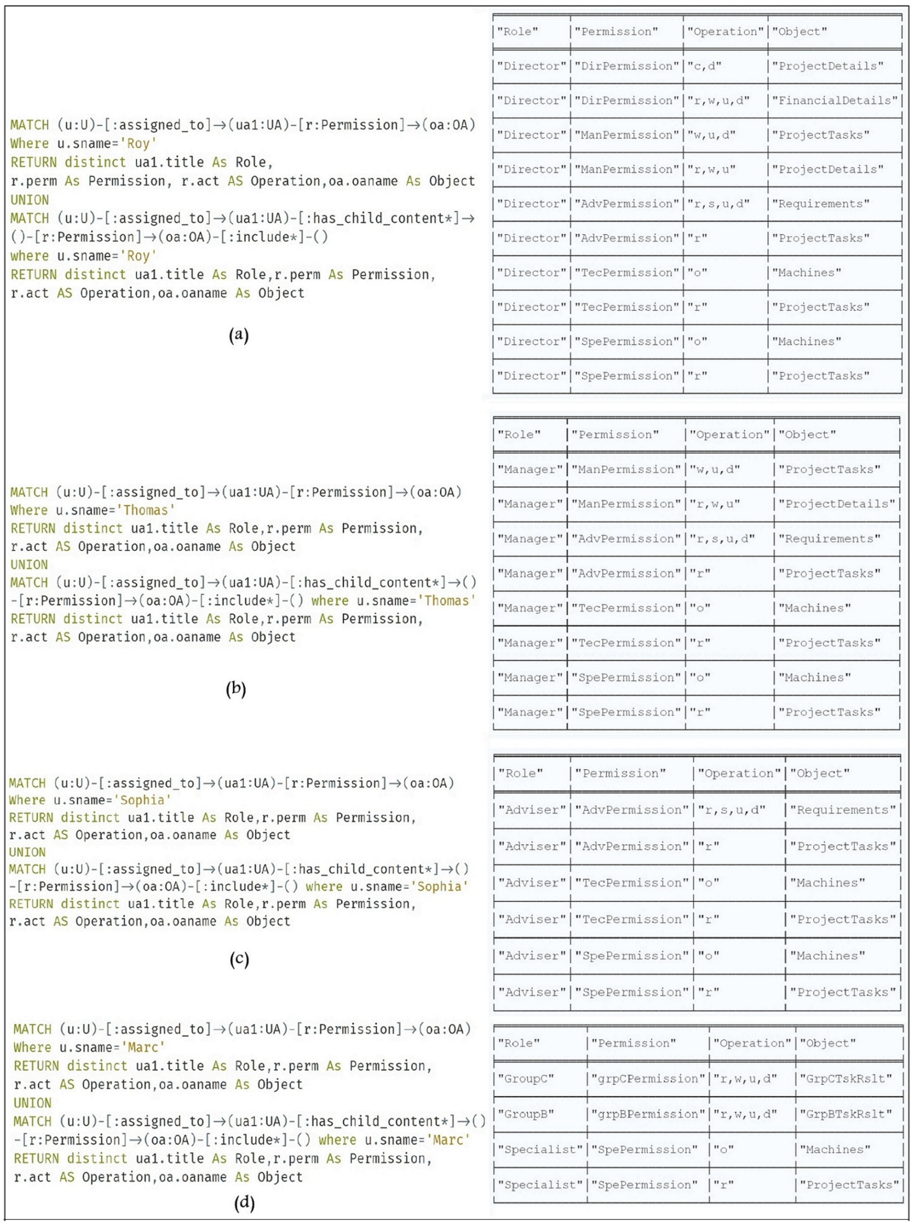


Figure 12. Case Study 1: The NGAC policy graph.

By assigning a user to a role (with the inheritance relation between roles), the user is indirectly associated with the access rights of that role's lower roles. Figure 13 illustrates permissions associated for some users on some objects such as in (a) Roy is assigned to the role of director and also has the permission of the lower roles (manager, adviser …). In (b) Thomas is assigned to role manager also has the permission of the lower roles (adviser, technician …). In (c), Sophia is assigned to role adviser and also has the permission of the lower roles. In (d), Marc is assigned to role specialist in addition to the permissions of groups B and C. For example, the user Roy has the permission DirPermission to confirm and delete {c, d} ProjectDetails data through the Roy–Director assignment, Roy also has the permission ManPermission to {r, w, u} ProjectDetails through the Director–Manager assignment which is expressed as 'has_child_content' relationship in Figure 12 to represent role hierarchy. Consequently, a manager with ManPermission also has AdvPermission, SpePermission, TecPermission, and so on. Note that, to simplify and minimize the number of association relationships between UAs and OAs in the graph of Figure 12, we use single association relationship instead of two or three. For example, in the association between

Manager(UA)–ProjectDetails(OA) which represents ManPermission with {r,w,u} operations, the operation {r} on ProjectDetails can unconditionally be performed by the manager, and the manager cannot perform {w,u} if the (non-contextual constraint) PrjConfirm value is true. In our illustration, we use a single association relationship, instead of two, and apply the constraints on all operations. In Figure 14 we illustrate an example for AdvPermission with its properties which include actions/operations and the contextual constraints. The adviser is able to perform {r, s, u, d} on project Requirements when the context LoginLocation='local' and the current date is before the endDate of the project.



```
MATCH (u:U)-[:assigned_to]→(ua1:UA)-[r:Permission]→(oa:OA)
Where u.sname='Roy'
RETURN distinct ua1.title As Role,
r.perm As Permission, r.act AS Operation,oa.oaname As Object
UNION
MATCH (u:U)-[:assigned_to]→(ua1:UA)-[:has_child_content*]→
()-[r:Permission]→(oa:OA)-[:include*]-()
where u.sname='Roy'
RETURN distinct ua1.title As Role,r.perm As Permission,
r.act AS Operation,oa.oaname As Object
```

(a)

"Role"	"Permission"	"Operation"	"Object"
"Director"	"DirPermission"	"c,d"	"ProjectDetails"
"Director"	"DirPermission"	"r,w,u,d"	"FinancialDetails"
"Director"	"ManPermission"	"w,u,d"	"ProjectTasks"
"Director"	"ManPermission"	"r,w,u"	"ProjectDetails"
"Director"	"AdvPermission"	"r,s,u,d"	"Requirements"
"Director"	"AdvPermission"	"r"	"ProjectTasks"
"Director"	"TecPermission"	"o"	"Machines"
"Director"	"TecPermission"	"r"	"ProjectTasks"
"Director"	"SpePermission"	"o"	"Machines"
"Director"	"SpePermission"	"r"	"ProjectTasks"

```
MATCH (u:U)-[:assigned_to]→(ua1:UA)-[r:Permission]→(oa:OA)
Where u.sname='Thomas'
RETURN distinct ua1.title As Role,r.perm As Permission,
r.act AS Operation,oa.oaname As Object
UNION
MATCH (u:U)-[:assigned_to]→(ua1:UA)-[:has_child_content*]→()
-[r:Permission]→(oa:OA)-[:include*]-() where u.sname='Thomas'
RETURN distinct ua1.title As Role,r.perm As Permission,
r.act AS Operation,oa.oaname As Object
```

(b)

"Role"	"Permission"	"Operation"	"Object"
"Manager"	"ManPermission"	"w,u,d"	"ProjectTasks"
"Manager"	"ManPermission"	"r,w,u"	"ProjectDetails"
"Manager"	"AdvPermission"	"r,s,u,d"	"Requirements"
"Manager"	"AdvPermission"	"r"	"ProjectTasks"
"Manager"	"TecPermission"	"o"	"Machines"
"Manager"	"TecPermission"	"r"	"ProjectTasks"
"Manager"	"SpePermission"	"o"	"Machines"
"Manager"	"SpePermission"	"r"	"ProjectTasks"

```
MATCH (u:U)-[:assigned_to]→(ua1:UA)-[r:Permission]→(oa:OA)
Where u.sname='Sophia'
RETURN distinct ua1.title As Role,r.perm As Permission,
r.act AS Operation,oa.oaname As Object
UNION
MATCH (u:U)-[:assigned_to]→(ua1:UA)-[:has_child_content*]→()
-[r:Permission]→(oa:OA)-[:include*]-() where u.sname='Sophia'
RETURN distinct ua1.title As Role,r.perm As Permission,
r.act AS Operation,oa.oaname As Object
```

(c)

"Role"	"Permission"	"Operation"	"Object"
"Adviser"	"AdvPermission"	"r,s,u,d"	"Requirements"
"Adviser"	"AdvPermission"	"r"	"ProjectTasks"
"Adviser"	"TecPermission"	"o"	"Machines"
"Adviser"	"TecPermission"	"r"	"ProjectTasks"
"Adviser"	"SpePermission"	"o"	"Machines"
"Adviser"	"SpePermission"	"r"	"ProjectTasks"

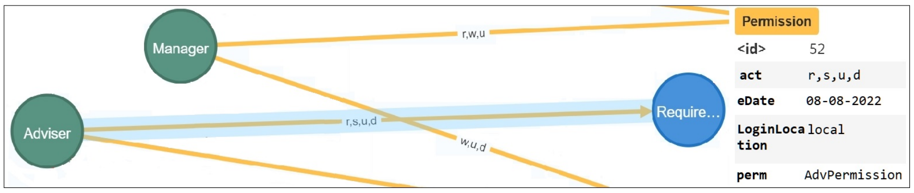
```
MATCH (u:U)-[:assigned_to]→(ua1:UA)-[r:Permission]→(oa:OA)
Where u.sname='Marc'
RETURN distinct ua1.title As Role,r.perm As Permission,
r.act AS Operation,oa.oaname As Object
UNION
MATCH (u:U)-[:assigned_to]→(ua1:UA)-[:has_child_content*]→()
-[r:Permission]→(oa:OA)-[:include*]-() where u.sname='Marc'
RETURN distinct ua1.title As Role,r.perm As Permission,
r.act AS Operation,oa.oaname As Object
```

(d)

"Role"	"Permission"	"Operation"	"Object"
"GroupC"	"grpCPermission"	"r,w,u,d"	"GrpCTskRslt"
"GroupB"	"grpBPermission"	"r,w,u,d"	"GrpBTskRslt"
"Specialist"	"SpePermission"	"o"	"Machines"
"Specialist"	"SpePermission"	"r"	"ProjectTasks"

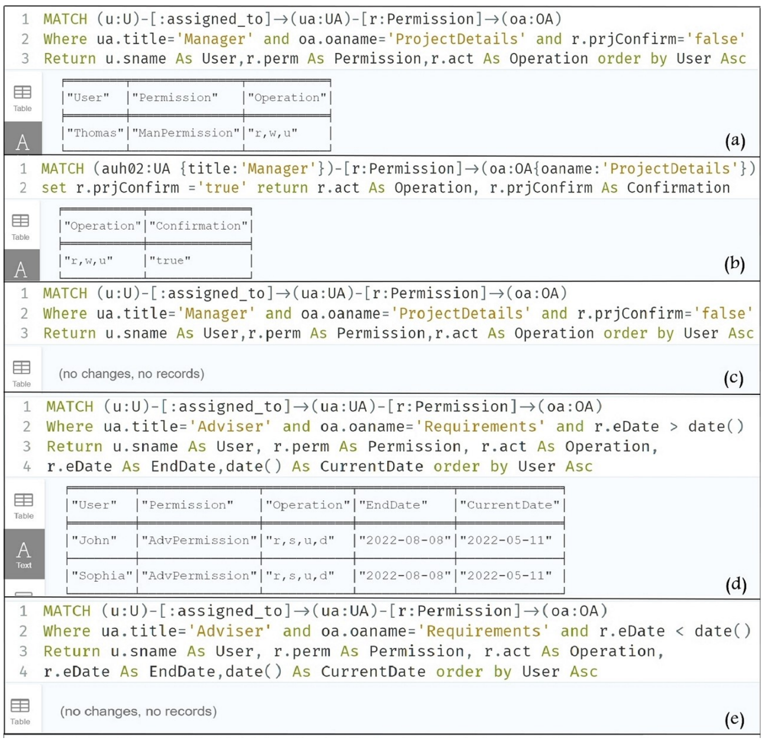
Figure 13. Case Study 1: Examples of some Cypher query results to show users access right based on the Permissions of (**a**) Director; (**b**) Manager; (**c**) Adviser; and (**d**) Specialist and groups B & C.

In Figure 15 we run some Cypher queries with some constraints using context, user, and object attributes to show NGAC authorization responses to Cypher statements. In (a) we show that the manager is able to perform the needed operations on ProjectDetails, (b) when the value of prjConfirm is updated (by the director) to 'true', (c) he would not be able to perform any operation on ProjectDetails. In (d) and (e) we show that an adviser is able to {r, w, u, d} project Requirements if the current date is less than the end project date.



Permission	
<id>	52
act	r,s,u,d
eDate	08-08-2022
LoginLocation	local
perm	AdvPermission

Figure 14. Case Study 1: Examples of association relationship properties for role permissions.



```
MATCH (u:U)-[:assigned_to]→(ua:UA)-[r:Permission]→(oa:OA)
Where ua.title='Manager' and oa.oaname='ProjectDetails' and r.prjConfirm='false'
Return u.sname As User,r.perm As Permission,r.act As Operation order by User Asc
```

```
╒════════╤═══════════════╤═══════════╕
│"User"  │"Permission"   │"Operation"│
╞════════╪═══════════════╪═══════════╡
│"Thomas"│"ManPermission"│"r,w,u"    │
└────────┴───────────────┴───────────┘
```

```
MATCH (auh02:UA {title:'Manager'})-[r:Permission]→(oa:OA{oaname:'ProjectDetails'})
set r.prjConfirm ='true' return r.act As Operation, r.prjConfirm As Confirmation
```

```
╒═══════════╤══════════════╕
│"Operation"│"Confirmation"│
╞═══════════╪══════════════╡
│"r,w,u"    │"true"        │
└───────────┴──────────────┘
```

```
MATCH (u:U)-[:assigned_to]→(ua:UA)-[r:Permission]→(oa:OA)
Where ua.title='Manager' and oa.oaname='ProjectDetails' and r.prjConfirm='false'
Return u.sname As User,r.perm As Permission,r.act As Operation order by User Asc
```

```
(no changes, no records)
```

```
MATCH (u:U)-[:assigned_to]→(ua:UA)-[r:Permission]→(oa:OA)
Where ua.title='Adviser' and oa.oaname='Requirements' and r.eDate > date()
Return u.sname As User, r.perm As Permission, r.act As Operation,
r.eDate As EndDate,date() As CurrentDate order by User Asc
```

```
╒════════╤═══════════════╤═══════════╤════════════╤═════════════╕
│"User"  │"Permission"   │"Operation"│"EndDate"   │"CurrentDate"│
╞════════╪═══════════════╪═══════════╪════════════╪═════════════╡
│"John"  │"AdvPermission"│"r,s,u,d"  │"2022-08-08"│"2022-05-11" │
├────────┼───────────────┼───────────┼────────────┼─────────────┤
│"Sophia"│"AdvPermission"│"r,s,u,d"  │"2022-08-08"│"2022-05-11" │
└────────┴───────────────┴───────────┴────────────┴─────────────┘
```

```
MATCH (u:U)-[:assigned_to]→(ua:UA)-[r:Permission]→(oa:OA)
Where ua.title='Adviser' and oa.oaname='Requirements' and r.eDate < date()
Return u.sname As User, r.perm As Permission, r.act As Operation,
r.eDate As EndDate,date() As CurrentDate order by User Asc
```

```
(no changes, no records)
```

Figure 15. Case Study 1: Examples of NGAC authorization responses to Cypher statements.

Controlling access to resources in industrial organizations is often managed at the application level, which is sufficient in static computing environments where changes in the data sources or system are not expected. On the other hand, industry 4.0 applications are generally dynamic, which means that new machines, devices, sensors, users, etc., are frequently added or changed, and the AC policies need to be frequently added and updated. Accordingly, industry 4.0 environment, and other highly dynamic environments, need flexible and frequently upgradable AC models to answer the frequent changes in AC requirements. In the following section, we propose a simplified case study for the industry 4.0 environment.

6. Case Study 2— ITMI: IoT

Due to the immense value IoT brings to every organization, ITMI refers to the use of IoT in improving existing systems and processes. This enables it to increase operational efficiency, create better experiences, and unlock additional value for the running projects. For ITMI, the IoT and industry 4.0 reflect a growing focus on driving results using sensor-based data and creating analytically rich data sets.

Figure 16 illustrates ITMI's IoT system architecture. Note that this case study continues the previous case study. One of the significant IoT projects maintained by ITMI, is the Inspection of the Railways of Quebec (IRQ) in Canada to detect any unrevealed cracks in railway tracks to avoid accidents. To achieve this, a rail robot (which is accomplished in case study 1) and a drone are connected or paired together so that they are synchronized and well geolocated. Both are connected to the internet via cellular or satellite networks depending on where the inspection is taking place. The sensor nodes attached to the rail robot are used to inspect the railways and take images of cracks and technical problems. Synchronously, the drone also captures images of locations where cracks in railway tracks exist. All data and captured images are sent to ITMI's cloud server. In this case study, AC is based on users' roles, in addition to other attributes of subjects, objects, and the environment. Those who have access to the labs 24/7 can take out the Rail Robot and the Drone and operate them on site. This is when the status of this task is in progress and within the start and end date of the running project. Onsite, users must log into the machines using PIN codes, they have three possible attempts to use the correct PIN code to operate and control the machines. Otherwise, the machines, using a GPS tracking system, send alert messages with the geographical coordinates of the location to the manager, specialists, and technicians. In the following we describe the guidelines to perform the project tasks:

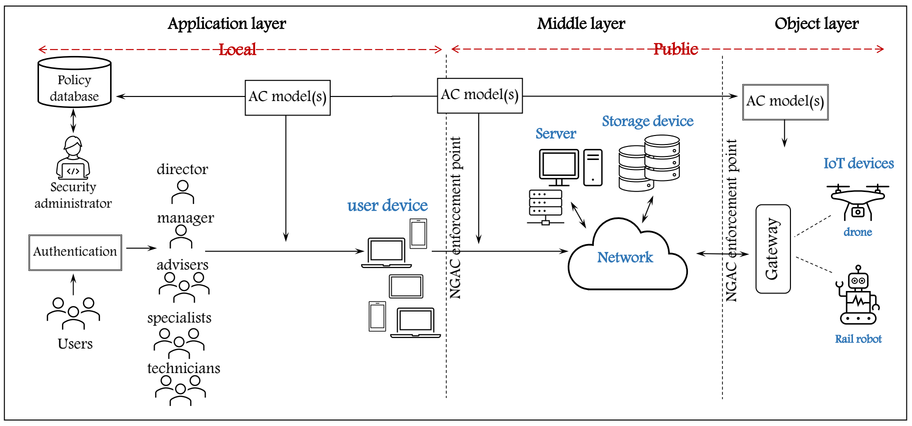


Figure 16. The system architecture of ITMI: IoT environment.

- The specialists Bob and Cathy with the technician Peter can operate the Rail Robot and the Drone using the IoT device controller to control the machines and read the collected data and images.
- The adviser John has full access to database tables at the ITMI cloud server where the collected data and images from the site are received from the Rail Robot and the Drone. He analyzes the obtained information, then writes/inserts the needed report information to the table Results.
- Before emailing the reports to the company maintaining the railways, the manager Thomas must read (and modify if needed), and confirm the report(s).

All users are allowed to perform the specified actions if the task status is in progress and within the project start and end dates.

6.1. Challenge

The rail robot and the drone could generate privacy-sensitive information, for example, for some logistic locations. Hence, it is critical to provide an efficient AC model considers all the needed factors to avoid operating the devices for any illegal use by any illegal user. With a set of physical identities (ITMI workers, company employees, and others) at some of the sites where the railway inspection process is assumed to take place, operating the rail robot and the drone must be carried out by trusted users. Moreover, the machines are objects that need to be controlled by authorized users, but they are also users when they send and insert data into the public database on ITMI's cloud server. In other words, they are objects in some context and subjects in another context.

6.2. The Solution: HEAD Metamodel

As shown in Figure 16, to protect the resources there are various AC models need to be implemented. To provide the desired solution for this case study, we will also follow the same steps explained in Figure 4. The DSL grammar of HEAD metamodel is used to derive the model(s) needed to protect the resources of ITMI's public environment. The solution should consider that the physical objects are also subjects (users)—for example, although the RailRobot and the Drone are objects need to be protected from any illegal access, they are also subjects when they connect to ITMI's public database to insert/write the collected data. Hence, we have to derive a hybrid model with two PCs, PC1 considers the machines as objects and PC2 considers them as subjects. However, the DSL of HEAD metamodel allows instantiating different PCs with E_x, I_m, and S_t entities.

- PC1: E_x = {subject (sname, . . .); object (oname, . . .)}.
  I_m: AUs = {role (type . . .)} and PUs = {permission (perm . . .); action (type . . .)}
  S_t = {contextual-constraint (loginLocation, time, pw-attempts . . .); constraint (inspectionState, confirmation . . .)}.
- PC2: E_x = {subject (sname, . . .); object (oname, . . .)}.
  I_m: PU = {action (type, . . .)}
  S_t = {contextual-constraint (Location, pwAttmpts . . .)}.

6.2.1. Phase 1: Deriving Models

In this section, we investigate the best AC model that fits the AC requirements of this case study. Due to the above Ex, Im, and St entities, we have the possibilities:

- Hybrid RBAC/ABAC model: due to the above entities for PC1, the notion of role reflects the importance of considering the RBAC model, and the need to express static and dynamic AC rules based on subject, object, and context attributes reflect the importance of considering the ABAC model.
- ABAC model: the above entities for PC2 match the entities of ABAC.

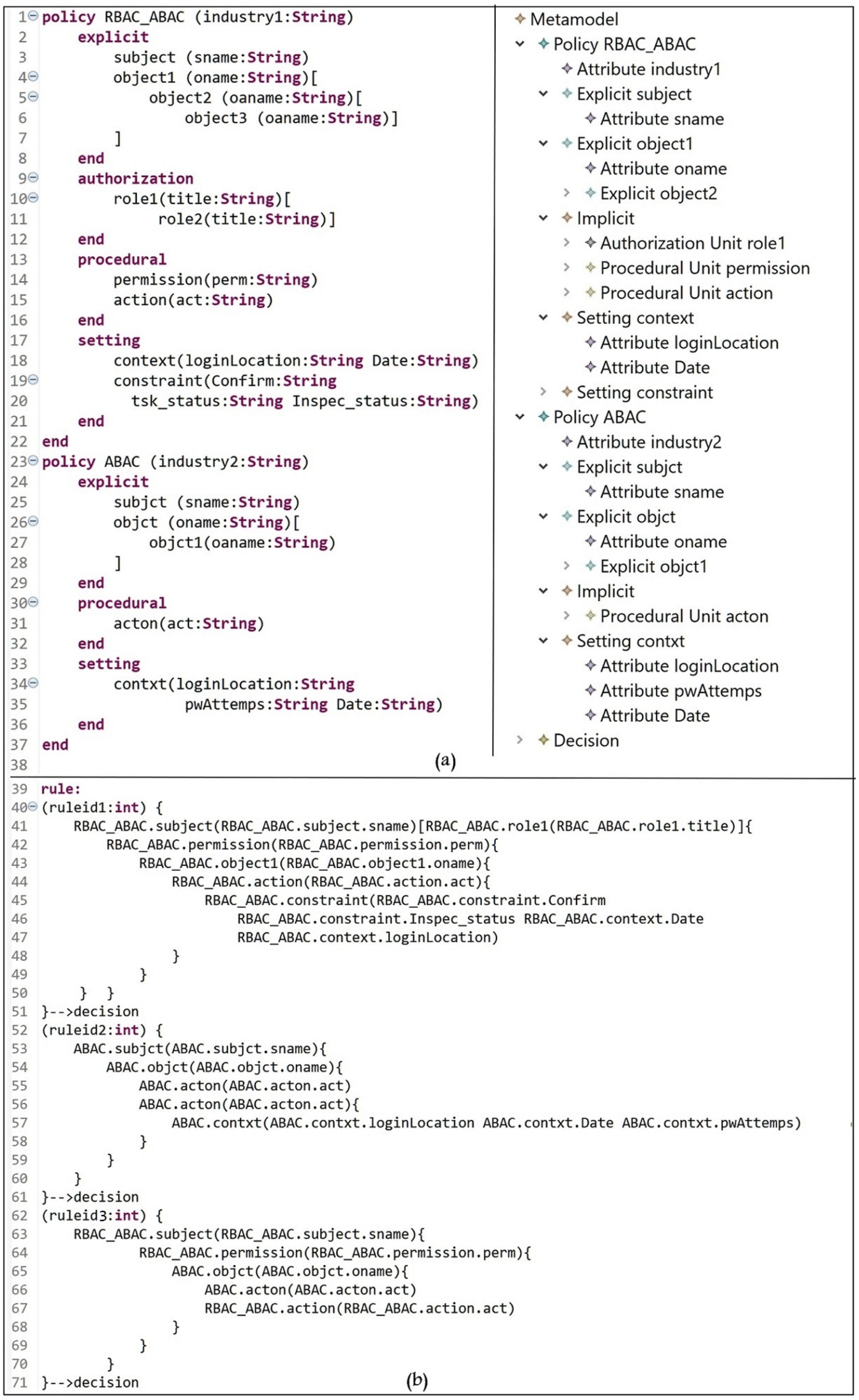
Accordingly, our solution is to derive a hybrid model with two PCs (the first is hybrid RBAC/ABAC, and the second is ABAC). Using the DSL grammar of HEAD metamodel, we derive the required model in Figure 17. In Figure 17a we define the following:

- Lines 1 to 22: the block of specifying PC1.
- Lines 2 to 8: the block of creating E_x entities (subject and object), it starts and ends with explicit and end keywords. Three levels of object hierarchy are created to represent objects. We actually have two levels of object hierarchy, but since we use NGAC we add an additional level. In object1 level, objects are described. In the other two levels, the hierarchy of objects containers is represented.
- Lines 9 to 12: the block of AU entities is created with the keywords authorization and end. Managers and advertisers have two levels of role hierarchy.
- Lines 13 to 16: the block of PU entities is created with procedural and end keywords.
- Lines 17 to 21: the block of creating S_t entities (context and constraint).
- Lines 23 to 37: the block of specifying PC2.
- Lines 24 to 29: the block of creating E_x entities (subject and object).
- Lines 30 to 32: the block of creating PU entity (action entity and its attributes).
- Lines 33 to 36: the block of creating S_t entities (context and constraint).

In Figure 17b we express some AC rules based on the defined entities/attributes:

- lines 40 to 51: the rule expression of the hybrid RBAC/ABAC model. It expresses access rights for users using PC1 components/attributes.
- lines 52 to 61: the rule expression of ABAC model. It expresses access rights for users using PC2 components/attributes.
- lines 62 to 71: hybrid rule expression using PC1 and PC2 components/attributes.

For each PC the names of the components must be unique. For example, we create two different names for the subject (for PC1 we create 'subject' entity and for PC2 we create 'subject' entity). Among the many advantages of the DSL for the HEAD metamodel is that it allows for the specification of an unlimited number of PCs, as well as the definition of entities and attributes for each PC.



```
policy RBAC_ABAC (industry1:String)
    explicit
        subject (sname:String)
        object1 (oname:String)[
            object2 (oaname:String)[
                object3 (oaname:String)]
        ]
    end
    authorization
        role1(title:String)[
             role2(title:String)]
    end
    procedural
        permission(perm:String)
        action(act:String)
    end
    setting
        context(loginLocation:String Date:String)
        constraint(Confirm:String
          tsk_status:String Inspec_status:String)
    end
end
policy ABAC (industry2:String)
    explicit
        subjct (sname:String)
        objct (oname:String)[
            objct1(oaname:String)
        ]
    end
    procedural
        acton(act:String)
    end
    setting
        contxt(loginLocation:String
                pwAttemps:String Date:String)
    end
end
```

```
Metamodel
  Policy RBAC_ABAC
    Attribute industry1
    Explicit subject
      Attribute sname
    Explicit object1
      Attribute oname
      Explicit object2
    Implicit
      Authorization Unit role1
      Procedural Unit permission
      Procedural Unit action
    Setting context
      Attribute loginLocation
      Attribute Date
    Setting constraint
  Policy ABAC
    Attribute industry2
    Explicit subjct
      Attribute sname
    Explicit objct
      Attribute oname
      Explicit objct1
    Implicit
      Procedural Unit acton
    Setting contxt
      Attribute loginLocation
      Attribute pwAttemps
      Attribute Date
  Decision
```

(a)

```
rule:
(ruleid1:int) {
    RBAC_ABAC.subject(RBAC_ABAC.subject.sname)[RBAC_ABAC.role1(RBAC_ABAC.role1.title)]{
        RBAC_ABAC.permission(RBAC_ABAC.permission.perm){
            RBAC_ABAC.object1(RBAC_ABAC.object1.oname){
                RBAC_ABAC.action(RBAC_ABAC.action.act){
                    RBAC_ABAC.constraint(RBAC_ABAC.constraint.Confirm
                        RBAC_ABAC.constraint.Inspec_status RBAC_ABAC.context.Date
                        RBAC_ABAC.context.loginLocation)
                }
            }
     }  }
}-->decision
(ruleid2:int) {
    ABAC.subjct(ABAC.subjct.sname){
        ABAC.objct(ABAC.objct.oname){
            ABAC.acton(ABAC.acton.act)
            ABAC.acton(ABAC.acton.act){
                ABAC.contxt(ABAC.contxt.loginLocation ABAC.contxt.Date ABAC.contxt.pwAttemps)
            }
        }
    }
}-->decision
(ruleid3:int) {
    RBAC_ABAC.subject(RBAC_ABAC.subject.sname){
            RBAC_ABAC.permission(RBAC_ABAC.permission.perm){
                ABAC.objct(ABAC.objct.oname){
                    ABAC.acton(ABAC.acton.act)
                    RBAC_ABAC.action(RBAC_ABAC.action.act)
                }
            }
        }
}-->decision
```

(b)

Figure 17. Case Study 2: (**a**) A hybrid model with two PCs; (**b**) rule expressions.

6.2.2. Phase 2: Generating Policies

In this section we describe the policy elements for each PC:

- PC1: Hybrid RBAC/ABAC
    - Users = {Thomas, John, Bob, Cathy, Peter}

- User Attributes: each user is associated with a set of attributes (e.g., name . . .).
- Role = {manager, adviser}.
- Objects: set of logical (e.g., tables rows/columns at the cloud server) and physical (e.g., rail robot and drone) objects need protection.
- Object attributes: object attributes include inspection_status, confirmation, etc.
- Actions: a set of actions on logical objects in the database include read, write, update, delete, etc.; and on physical objects include control, operate, etc.
- Permissions: are granted based on users' roles to perform actions on objects.
- Context attributes: for a given context—for example, log in via private/public network, machine failure, etc.,—context attributes include the date, time, location, password attempts, etc. Contextual constraint expressions are significant for authorization decisions.
- Constraints: expressions include the other various types of attributes, for example, subject attributes, object attributes, etc.

The policy expression for PC1 can be expressed as follows:

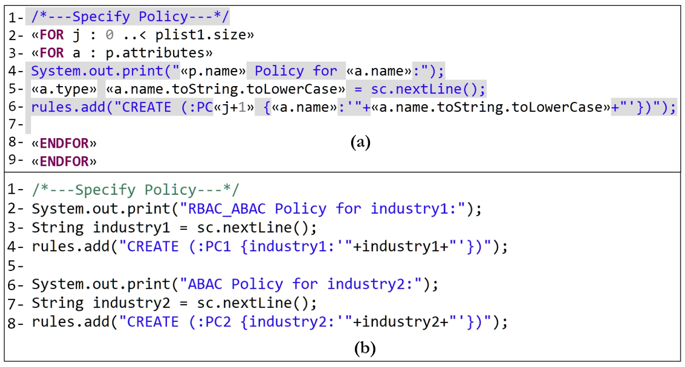
$$Policy = \langle subject(name \ldots); object(title \ldots); role(type \ldots); permission(ptype \ldots); \\ action(type \ldots); context(date \ldots); constraints(confirm \ldots)\rangle$$

- PC2: ABAC
  - Users = {RailRobot, Drone}, and users' attributes (e.g., machineName).
  - Objects: the logical resources (e.g., tables rows/columns at the cloud server), with their attributes, e.g., coordinates, image, etc.
  - Actions: set of actions on logical objects in the database, e.g., write.
  - Contextual and non-contextual constraint expressions with the attributes of context, subject, and object.

  The policy expression for PC2 can be expressed as follows:

$$Policy = \langle subject(name \ldots); object(title \ldots); action(type \ldots); context(time \ldots)\rangle$$

At the system level, Xtend notation is used to generate the Java code for the concrete instance of the hybrid model. Then, Java output (Cypher statements) is injected into the Neo4j database to represent the NGAC policy graph. In this case study we have two PCs (as defined in Figure 17 RBAC/ABAC for PC1, and ABAC for PC2). To specify the PCs, the sample of Xtend code in Figure 18a generates the Java code in Figure 18b based on the derived model. In Figure 18a, plist1 in line 2 is an ArrayList where PCs are added (in the Xtend notation code). Since in this case study we have two PCs, in Figure 18b two sets of Java code (lines 2–4 and lines 6–8) are generated to define the attribute name of each PC, with the Cypher expression to create each PC node.



```
1- /*---Specify Policy---*/
2- «FOR j : 0 ..< plist1.size»
3- «FOR a : p.attributes»
4- System.out.print("«p.name» Policy for «a.name»:");
5- «a.type» «a.name.toString.toLowerCase» = sc.nextLine();
6- rules.add("CREATE (:PC«j+1» {«a.name»:'"+«a.name.toString.toLowerCase»+"'})");
7-
8- «ENDFOR»
9- «ENDFOR»
```

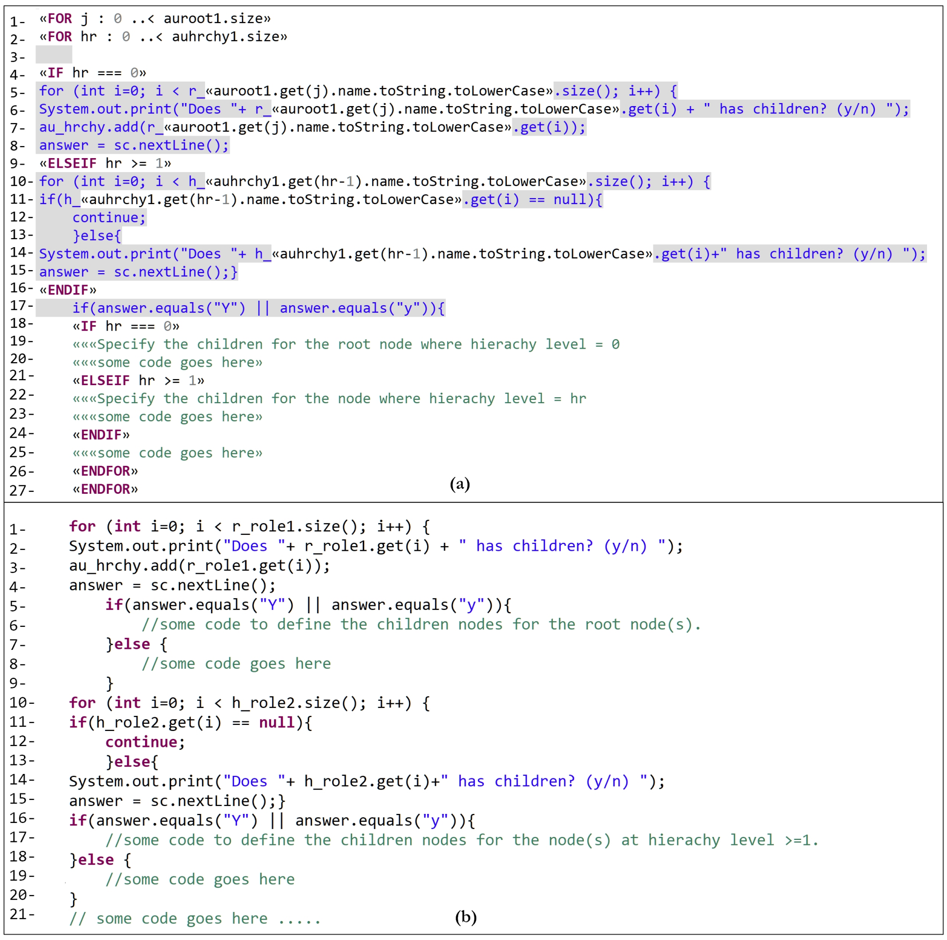
(a)

```
1- /*---Specify Policy---*/
2- System.out.print("RBAC_ABAC Policy for industry1:");
3- String industry1 = sc.nextLine();
4- rules.add("CREATE (:PC1 {industry1:'"+industry1+"'})");
5-
6- System.out.print("ABAC Policy for industry2:");
7- String industry2 = sc.nextLine();
8- rules.add("CREATE (:PC2 {industry2:'"+industry2+"'})");
```

(b)

Figure 18. Case Study 2: (**a**) Xtend notation to specify PCs; (**b**) the generated Java code.

For each PC, sets of E_x, AU, PU, and S_t entities/attributes are created. In Figure 19a we show a sample of Xtend notation to define the hierarchy of AUs (e.g., roles). The ArrayList 'auroot1' for the root entities, and 'auhrchy' for the lower level entities (child nodes). In lines 4–8, if the node is at the root level and has children, then specify its children (lines 18–20). In lines 9–15, if the node level is ≥1 and it has children, then specify its children (lines 21–23). Figure 19a generates the Java code in Figure 19b which is used to configure the hierarchy of roles. In line 1, 'r_role1' refers to the ArrayList where root nodes of roles are defined, and in line 10 'h_role2' refers to the ArrayList where nodes at level 2 of the role hierarchy are defined.



```
1-  «FOR j : 0 ..< auroot1.size»
2-  «FOR hr : 0 ..< auhrchy1.size»
3-
4-  «IF hr === 0»
5-  for (int i=0; i < r_«auroot1.get(j).name.toString.toLowerCase».size(); i++) {
6-  System.out.print("Does "+ r_«auroot1.get(j).name.toString.toLowerCase».get(i) + " has children? (y/n) ");
7-  au_hrchy.add(r_«auroot1.get(j).name.toString.toLowerCase».get(i));
8-  answer = sc.nextLine();
9-  «ELSEIF hr >= 1»
10- for (int i=0; i < h_«auhrchy1.get(hr-1).name.toString.toLowerCase».size(); i++) {
11- if(h_«auhrchy1.get(hr-1).name.toString.toLowerCase».get(i) == null){
12-     continue;
13-     }else{
14- System.out.print("Does "+ h_«auhrchy1.get(hr-1).name.toString.toLowerCase».get(i)+" has children? (y/n) ");
15- answer = sc.nextLine();}
16- «ENDIF»
17-     if(answer.equals("Y") || answer.equals("y")){
18-     «IF hr === 0»
19-     «««Specify the children for the root node where hierachy level = 0
20-     «««some code goes here»
21-     «ELSEIF hr >= 1»
22-     «««Specify the children for the node where hierachy level = hr
23-     «««some code goes here»
24-     «ENDIF»
25-     «««some code goes here»
26-     «ENDFOR»
27-     «ENDFOR»
```

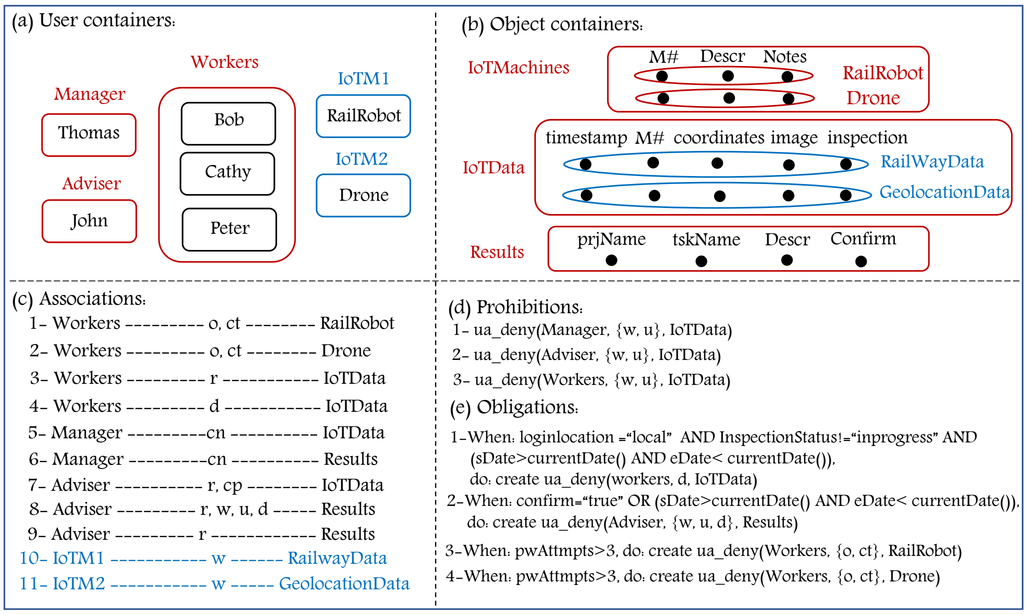
(a)

```
1-      for (int i=0; i < r_role1.size(); i++) {
2-      System.out.print("Does "+ r_role1.get(i) + " has children? (y/n) ");
3-      au_hrchy.add(r_role1.get(i));
4-      answer = sc.nextLine();
5-          if(answer.equals("Y") || answer.equals("y")){
6-              //some code to define the children nodes for the root node(s).
7-          }else {
8-              //some code goes here
9-          }
10-     for (int i=0; i < h_role2.size(); i++) {
11-     if(h_role2.get(i) == null){
12-         continue;
13-         }else{
14-     System.out.print("Does "+ h_role2.get(i)+" has children? (y/n) ");
15-     answer = sc.nextLine();}
16-     if(answer.equals("Y") || answer.equals("y")){
17-         //some code to define the children nodes for the node(s) at hierachy level >=1.
18-     }else {
19-         //some code goes here
20-     }
21-     // some code goes here .....
```

(b)

Figure 19. Case Study 2: Xtend notation to generate Java code to define (**a**) hierarchy AUs; (**b**) the generated Java code to configure hierarchy of roles.

To output the required Cypher statements and represent the NGAC policy graph, the system administrator must be aware how to configure the policy by specifying the users, resources, relationships, conditions, etc., based on the computing environment of this case study. However, in Figure 20a, we show the configuration of user containers. Red containers represent the assignment of Us to UAs of the first PC1, and blue ones represent the assignment of Us to UAs of the second PC2. In PC1, U stands for Thomas, John, etc., and UA for the role (Manager, Adviser, etc.). In PC2, U stands for Rail Robot and Drone, and UA for the first and second IoT devices. In Figure 20b, we show object containers for physical resources (Rail Robot and Drone) are indicated in red for PC1, and logical resources (database tables at ITMI's cloud server) with the representation of tables with some distinguished rows/columns are indicated in red for PC1, and in blue for PC2. In Figure 20c, access rights of users to perform operations are determined through associations. For example, in (1) and (2) the workers are allowed to {o: operate, ct: control} the Rail Robot and Drone objects, in (5) the manager should confirm that the inspection process is carried out before allowing the adviser in (7) to {r:read, cp:copy} the collected data in order to analyze them. Moreover, in (10) and (11) the subjects of PC2, the Rail Robot and the Drone which are assigned to IoTM1 and IoTM2 as UAs, need permission to connect the cloud server to {w:write/insert} data and images into RailwayData and GeolocationData containers. In Figure 20d we present three prohibition relations which express ua_deny to deny all users performing {w, u} on IoTData. In Figure 20e, four obligations are defined as event–response relations to define constraints under which policy state data are obligated to change. For example, workers onsite are allowed to access the cloud server and delete the collected IoTData due to unexpected events, e.g., machine malfunction or system error, to repeat the inspection process. In (1), they are not allowed to {d} IoTData if they are logged into the system locally, if the InspectionStatus= "in progress", and if they are not within the start and end dates of the IRQ project.



(c) Associations:
1- Workers --------- o, ct -------- RailRobot
2- Workers --------- o, ct -------- Drone
3- Workers --------- r ----------- IoTData
4- Workers --------- d ----------- IoTData
5- Manager ---------cn ---------- IoTData
6- Manager ---------cn ---------- Results
7- Adviser ---------- r, cp -------- IoTData
8- Adviser --------- r, w, u, d ----- Results
9- Adviser --------- r ------------ Results
10- IoTM1 ----------- w ------ RailwayData
11- IoTM2 ----------- w ----- GeolocationData

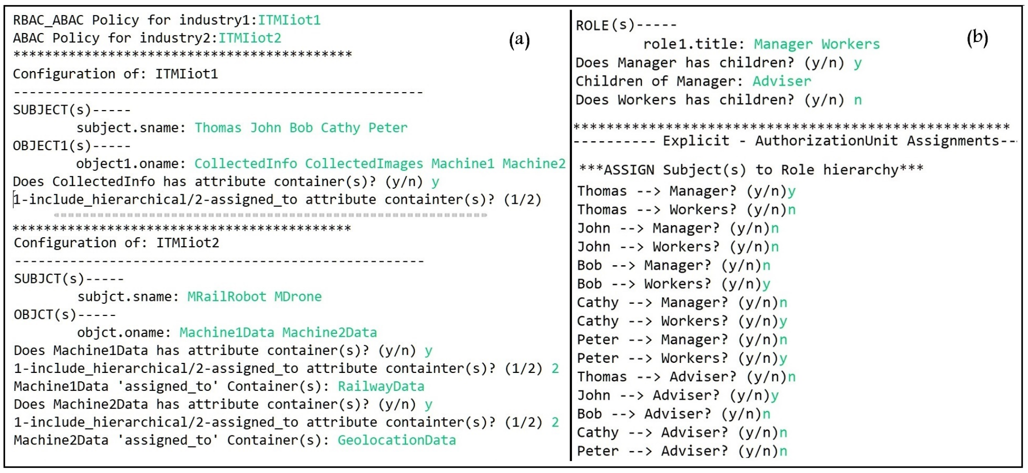
(d) Prohibitions:
1- ua_deny(Manager, {w, u}, IoTData)
2- ua_deny(Adviser, {w, u}, IoTData)
3- ua_deny(Workers, {w, u}, IoTData)

(e) Obligations:
1-When: loginlocation ="local" AND InspectionStatus!="inprogress" AND (sDate>currentDate() AND eDate< currentDate()), do: create ua_deny(workers, d, IoTData)
2-When: confirm="true" OR (sDate>currentDate() AND eDate< currentDate()), do: create ua_deny(Adviser, {w, u, d}, Results)
3-When: pwAttmpts>3, do: create ua_deny(Workers, {o, ct}, RailRobot)
4-When: pwAttmpts>3, do: create ua_deny(Workers, {o, ct}, Drone)

Figure 20. Case Study 2: NGAC Policy Configuration.

At the system level, a sample of the Java output with the policy configuration is presented in Figure 21. In (a), the system administrator specifies the PCs (ITMIiot1 and ITMIiot2) and configures the entities/attributes for each PC, along with setting up relationships between objects. In (b), the creation of instances of role entities and their hierarchies is presented, in addition to the assignment relationship of E_x-AUs which represent in this case user-role relationship. For example, the name of PC2 is ITMIiot2, the subject names are MRailRobot and MDrone, the object names are Machine1Data and Machine2Data, Machine1Data has OA and it is 'assigned_to' RailwayData, and Machine2Data has OA and it is 'assigned_to' GeolocationData.

After policy configuration by the system administrator, Cypher statements are generated as Java output that matches the NGAC policy graph. Different output samples of Cypher statements are presented in Figure 22a to create PCs, Us, and Os. Figure 22b show some examples of Cypher statements for PC1 to create UAs with the needed assignment and hierarchies. Also, Figure 22c shows some statements to create UA nodes (e.g., IoTM1) of PC2 and assign the appropriate Us (e.g., MRailRobot) to them. For example, the 'MRailRobot' (U node) is assigned to 'IoTM1' (UA container).



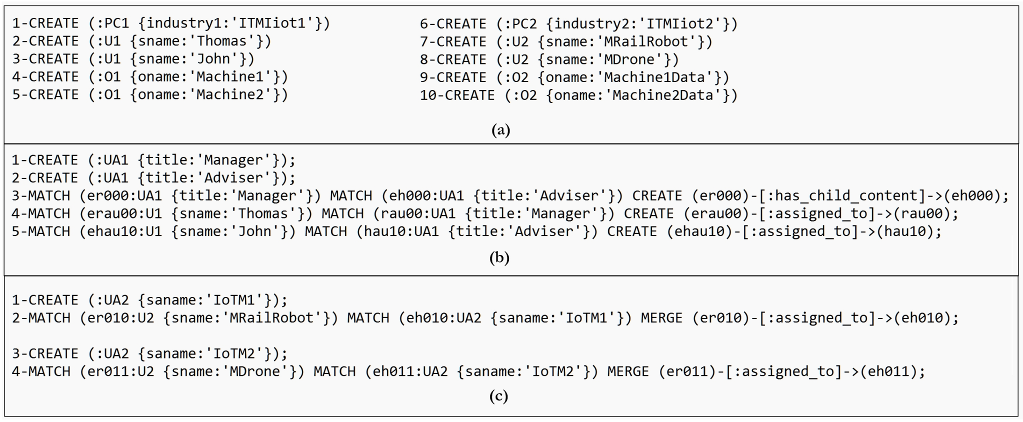
(a)

```
RBAC_ABAC Policy for industry1:ITMIiot1
ABAC Policy for industry2:ITMIiot2
*******************************************
Configuration of: ITMIiot1
----------------------------------------------------
SUBJECT(s)-----
        subject.sname: Thomas John Bob Cathy Peter
OBJECT1(s)-----
        object1.oname: CollectedInfo CollectedImages Machine1 Machine2
Does CollectedInfo has attribute container(s)? (y/n) y
1-include_hierarchical/2-assigned_to attribute containter(s)? (1/2)
*******************************************
Configuration of: ITMIiot2
----------------------------------------------------
SUBJCT(s)-----
        subjct.sname: MRailRobot MDrone
OBJCT(s)-----
        objct.oname: Machine1Data Machine2Data
Does Machine1Data has attribute container(s)? (y/n) y
1-include_hierarchical/2-assigned_to attribute containter(s)? (1/2) 2
Machine1Data 'assigned_to' Container(s): RailwayData
Does Machine2Data has attribute container(s)? (y/n) y
1-include_hierarchical/2-assigned_to attribute containter(s)? (1/2) 2
Machine2Data 'assigned_to' Container(s): GeolocationData
```

(b)

```
ROLE(s)-----
        role1.title: Manager Workers
Does Manager has children? (y/n) y
Children of Manager: Adviser
Does Workers has children? (y/n) n
****************************************************
---------- Explicit - AuthorizationUnit Assignments--
 ***ASSIGN Subject(s) to Role hierarchy***
Thomas --> Manager? (y/n)y
Thomas --> Workers? (y/n)n
John --> Manager? (y/n)n
John --> Workers? (y/n)n
Bob --> Manager? (y/n)n
Bob --> Workers? (y/n)y
Cathy --> Manager? (y/n)n
Cathy --> Workers? (y/n)y
Peter --> Manager? (y/n)n
Peter --> Workers? (y/n)y
Thomas --> Adviser? (y/n)n
John --> Adviser? (y/n)y
Bob --> Adviser? (y/n)n
Cathy --> Adviser? (y/n)n
Peter --> Adviser? (y/n)n
```

Figure 21. Case Study 2: Java output sample (**a**) to configure PC1/PC2; (**b**) user-role assignment.



```
1-CREATE (:PC1 {industry1:'ITMIiot1'})          6-CREATE (:PC2 {industry2:'ITMIiot2'})
2-CREATE (:U1 {sname:'Thomas'})                 7-CREATE (:U2 {sname:'MRailRobot'})
3-CREATE (:U1 {sname:'John'})                   8-CREATE (:U2 {sname:'MDrone'})
4-CREATE (:O1 {oname:'Machine1'})               9-CREATE (:O2 {oname:'Machine1Data'})
5-CREATE (:O1 {oname:'Machine2'})               10-CREATE (:O2 {oname:'Machine2Data'})
```

(a)

```
1-CREATE (:UA1 {title:'Manager'});
2-CREATE (:UA1 {title:'Adviser'});
3-MATCH (er000:UA1 {title:'Manager'}) MATCH (eh000:UA1 {title:'Adviser'}) CREATE (er000)-[:has_child_content]->(eh000);
4-MATCH (erau00:U1 {sname:'Thomas'}) MATCH (rau00:UA1 {title:'Manager'}) CREATE (erau00)-[:assigned_to]->(rau00);
5-MATCH (ehau10:U1 {sname:'John'}) MATCH (hau10:UA1 {title:'Adviser'}) CREATE (ehau10)-[:assigned_to]->(hau10);
```

(b)

```
1-CREATE (:UA2 {saname:'IoTM1'});
2-MATCH (er010:U2 {sname:'MRailRobot'}) MATCH (eh010:UA2 {saname:'IoTM1'}) MERGE (er010)-[:assigned_to]->(eh010);

3-CREATE (:UA2 {saname:'IoTM2'});
4-MATCH (er011:U2 {sname:'MDrone'}) MATCH (eh011:UA2 {saname:'IoTM2'}) MERGE (er011)-[:assigned_to]->(eh011);
```

(c)

Figure 22. Case Study 2: A sample of Cypher statements (**a**) PC, U, and O nodes; (**b**) UAs of roles with U–UA assignment; and (**c**) a sample U–UA assignment of PC2.

6.2.3. Phase 3: NGAC—The Policy Enforcement Point

The generated Cypher statements, which are Java outputs, are used as inputs for NGAC authorization responses. However, they are injected into the Neo4j database to represent the AC policy as an NGAC policy graph in Figure 23. The dark brown, green, and blue nodes refer to PC1, and the light brown, green, and blue nodes refer to PC2. As shown in the graph on the left, the dark/light brown nodes represent the assignment of users (Us) to their UA containers, in addition to the role hierarchy, which is indicated by the red arrow, and 'has_child_content' relationship. In the left section of the graph, the dark/light brown with the dark/light blue nodes show the assignment of objects (Os) to their OA containers, in addition to object hierarchy which is depicted by the red arrow and 'include' relationship. The association relationships, the yellow arrows for PC1 and the dark brown arrows for PC2, indicate users' access rights.

Due to the inheritance relationship between the manager and the adviser roles, in Figure 24 we show some access rights examples associated with the manager Thomas and the users who are assigned to the Workers (UA) container. In Figure 24a, Thomas has ManPermission to {cn: confirm} data in Results and IoTData containers. He also has the permission AdvPermission to {w, u, d} Results and {r, cp} IoTData through the Manager–Adviser assignment which is expressed as 'has_child_content' relationship to represent role hierarchy. In Figure 24b, we show the access rights associated to the workers Bob, Cathy, and Peter to {d} IoTData in a certain context and {o:operate, ct:control} the RailRobot and the Drone. In Figure 24c, we show the set of permissions associated with users to perform operations on the IoTData object container.

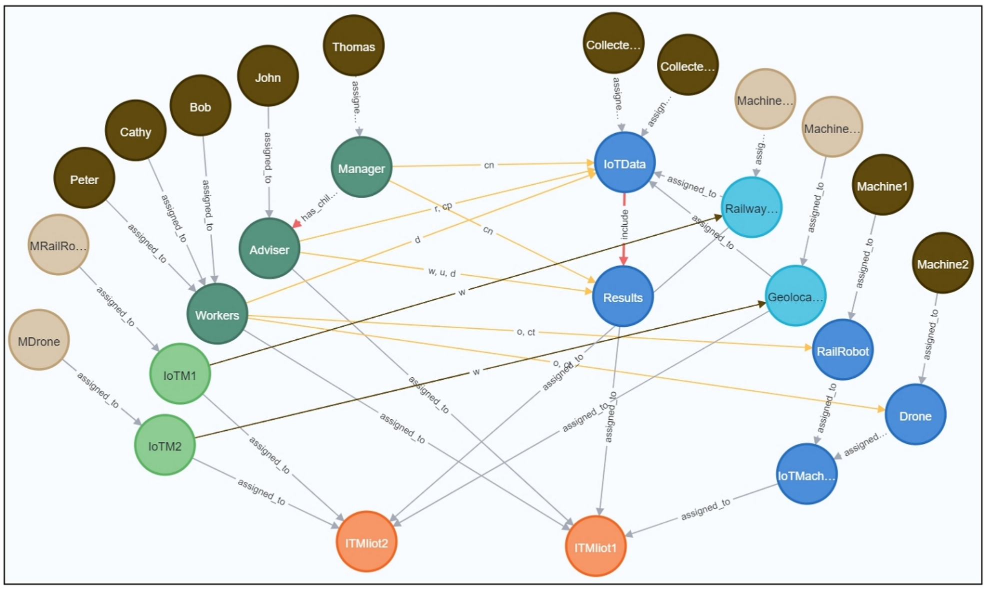
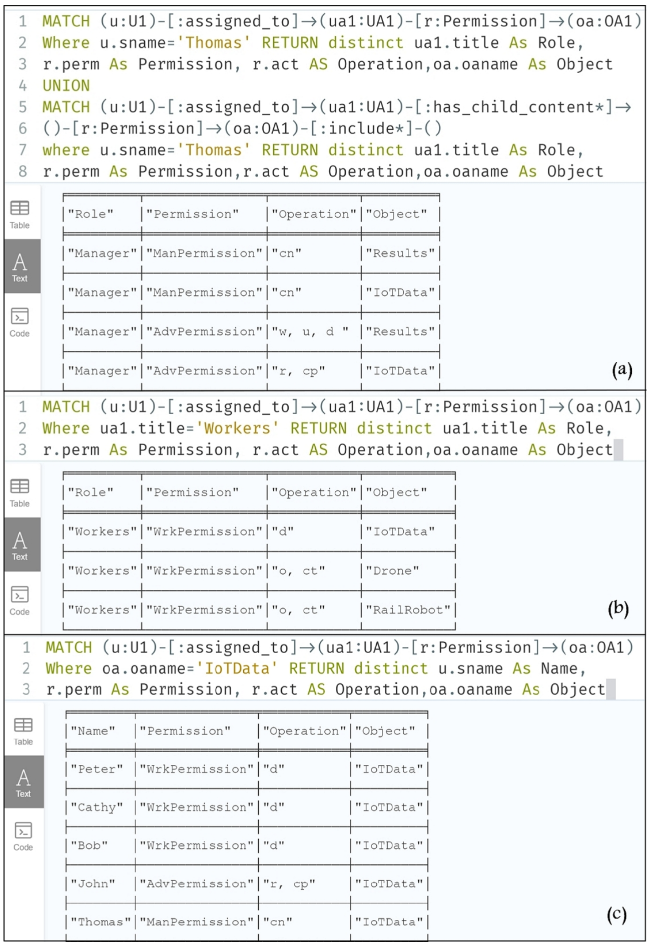


Figure 23. Case Study 2: NGAC graph.



```
MATCH (u:U1)-[:assigned_to]→(ua1:UA1)-[r:Permission]→(oa:OA1)
Where u.sname='Thomas' RETURN distinct ua1.title As Role,
r.perm As Permission, r.act AS Operation,oa.oaname As Object
UNION
MATCH (u:U1)-[:assigned_to]→(ua1:UA1)-[:has_child_content*]→
()-[r:Permission]→(oa:OA1)-[:include*]-()
where u.sname='Thomas' RETURN distinct ua1.title As Role,
r.perm As Permission,r.act AS Operation,oa.oaname As Object
```

```
│"Role"    │"Permission"   │"Operation"│"Object" │
│"Manager" │"ManPermission"│"cn"       │"Results"│
│"Manager" │"ManPermission"│"cn"       │"IoTData"│
│"Manager" │"AdvPermission"│"w, u, d " │"Results"│
│"Manager" │"AdvPermission"│"r, cp"    │"IoTData"│
```

(a)

```
MATCH (u:U1)-[:assigned_to]→(ua1:UA1)-[r:Permission]→(oa:OA1)
Where ua1.title='Workers' RETURN distinct ua1.title As Role,
r.perm As Permission, r.act AS Operation,oa.oaname As Object
```

```
│"Role"    │"Permission"   │"Operation"│"Object"   │
│"Workers" │"WrkPermission"│"d"        │"IoTData"  │
│"Workers" │"WrkPermission"│"o, ct"    │"Drone"    │
│"Workers" │"WrkPermission"│"o, ct"    │"RailRobot"│
```

(b)

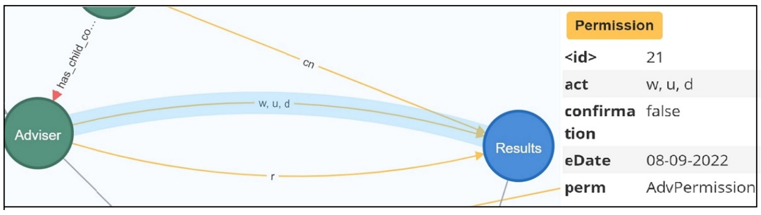
```
MATCH (u:U1)-[:assigned_to]→(ua1:UA1)-[r:Permission]→(oa:OA1)
Where oa.oaname='IoTData' RETURN distinct u.sname As Name,
r.perm As Permission, r.act AS Operation,oa.oaname As Object
```

```
│"Name"    │"Permission"   │"Operation"│"Object" │
│"Peter"   │"WrkPermission"│"d"        │"IoTData"│
│"Cathy"   │"WrkPermission"│"d"        │"IoTData"│
│"Bob"     │"WrkPermission"│"d"        │"IoTData"│
│"John"    │"AdvPermission"│"r, cp"    │"IoTData"│
│"Thomas"  │"ManPermission"│"cn"       │"IoTData"│
```

(c)

Figure 24. Case Study 2: Examples of Users' access rights (**a**) for Manager; (**b**) for Workers; (**c**) for users to access IoTData object.

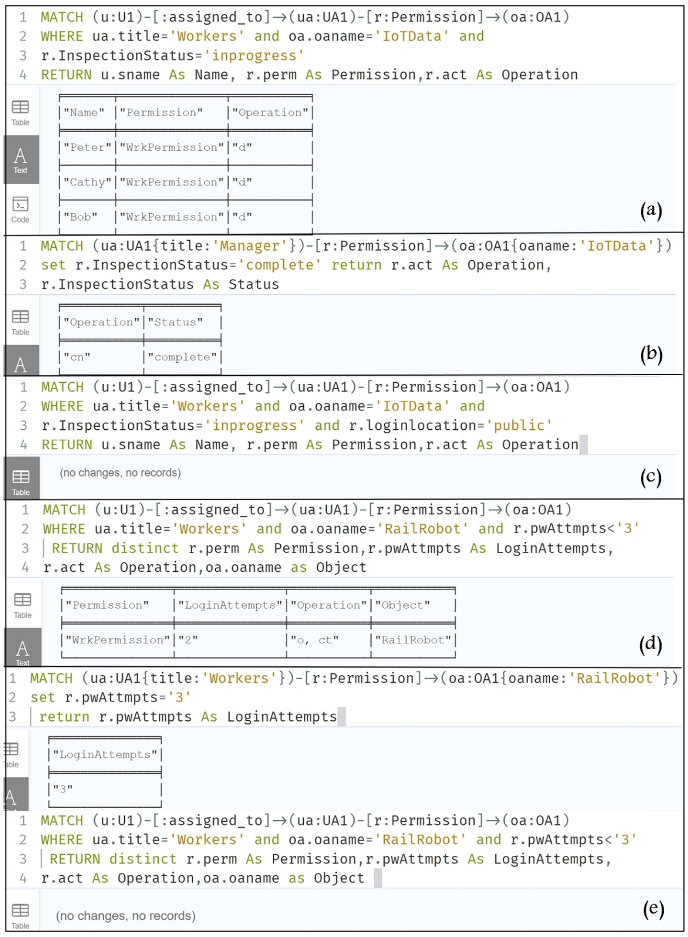
Similar to case study 1, to avoid multiple association relationships between UA and OA containers we use single association relationship, for example between Adviser and Results. In Figure 25 we show an example with double association relationships between Adviser and Results. The first association is to express the {w, u, d} operations and the second association is to express the {r} operation and express the constraints with each association.



Permission	
<id>	21
act	w, u, d
confirmation	false
eDate	08-09-2022
perm	AdvPermission

Figure 25. Case Study 2: Example of association relationship properties for Advisers permission.

The Neo4j platform is used to run some Cypher queries with some contextual and non-contextual constraints to present NGAC authorization responses. In Figure 26a, the Workers are able to {d} IoTData when the value of InspectionStatus = 'inprogress', which means the inspection process is not completed and more data need to be collected, but when the value of InspectionStatus is updated (by the manager) to 'complete' as illustrated in (b), the workers would not be allowed to delete the IoTData even if they are logged into the system via the public network as shown in (c). In Figure 26d we show that a Worker can {o, ct} the Rail Robot or the Drone if the pwAttpmts are less than or equal to three attempts, otherwise (e) he would not.



```
MATCH (u:U1)-[:assigned_to]→(ua:UA1)-[r:Permission]→(oa:OA1)
WHERE ua.title='Workers' and oa.oaname='IoTData' and
r.InspectionStatus='inprogress'
RETURN u.sname As Name, r.perm As Permission,r.act As Operation
```

```
╒═══════╤═══════════════╤═══════════╕
│"Name" │"Permission"   │"Operation"│
╞═══════╪═══════════════╪═══════════╡
│"Peter"│"WrkPermission"│"d"        │
├───────┼───────────────┼───────────┤
│"Cathy"│"WrkPermission"│"d"        │
├───────┼───────────────┼───────────┤
│"Bob"  │"WrkPermission"│"d"        │
└───────┴───────────────┴───────────┘
```

(a)

```
MATCH (ua:UA1{title:'Manager'})-[r:Permission]→(oa:OA1{oaname:'IoTData'})
set r.InspectionStatus='complete' return r.act As Operation,
r.InspectionStatus As Status
```

```
╒═══════════╤══════════╕
│"Operation"│"Status"  │
╞═══════════╪══════════╡
│"cn"       │"complete"│
```

(b)

```
MATCH (u:U1)-[:assigned_to]→(ua:UA1)-[r:Permission]→(oa:OA1)
WHERE ua.title='Workers' and oa.oaname='IoTData' and
r.InspectionStatus='inprogress' and r.loginlocation='public'
RETURN u.sname As Name, r.perm As Permission,r.act As Operation
```

```
(no changes, no records)
```

(c)

```
MATCH (u:U1)-[:assigned_to]→(ua:UA1)-[r:Permission]→(oa:OA1)
WHERE ua.title='Workers' and oa.oaname='RailRobot' and r.pwAttmpts<'3'
RETURN distinct r.perm As Permission,r.pwAttmpts As LoginAttempts,
r.act As Operation,oa.oaname as Object
```

```
╒═══════════════╤═══════════════╤═══════════╤═══════════╕
│"Permission"   │"LoginAttempts"│"Operation"│"Object"   │
╞═══════════════╪═══════════════╪═══════════╪═══════════╡
│"WrkPermission"│"2"            │"o, ct"    │"RailRobot"│
└───────────────┴───────────────┴───────────┴───────────┘
```

(d)

```
MATCH (ua:UA1{title:'Workers'})-[r:Permission]→(oa:OA1{oaname:'RailRobot'})
set r.pwAttmpts='3'
return r.pwAttmpts As LoginAttempts
```

```
╒═══════════════╕
│"LoginAttempts"│
╞═══════════════╡
│"3"            │
└───────────────┘
```

```
MATCH (u:U1)-[:assigned_to]→(ua:UA1)-[r:Permission]→(oa:OA1)
WHERE ua.title='Workers' and oa.oaname='RailRobot' and r.pwAttmpts<'3'
RETURN distinct r.perm As Permission,r.pwAttmpts As LoginAttempts,
r.act As Operation,oa.oaname as Object
```

```
(no changes, no records)
```

(e)

Figure 26. Case Study 2: Examples of NGAC authorization responses to Cypher statements.

7. HEAD Administrative Panel

Based on the case study in Section 5, we provide another example of the HEAD metamodel implementation using VB.net and SQL. As well, the AC decision is based on the user's role/group, contextual information (location, time, etc.), and subject/object attributes. An example of an administrative panel can be found in Figure 27, which shows how to create the AC model using entities and attributes such as E_x, AU, PU, and S_t. In Figure 28 we show the steps to create AC model and its components (in this case we show the Hybrid model of case study 1). As shown in Figure 28a, after defining the needed model (1), the administrator is able to edit the model to update, delete, or (2) create/define and (3) add model entities (e.g., object). In Figure 28b, the needed entity (e.g., object) is (1) selected to (2) define the number of hierarchy levels, and the administrator (3) sets and (4) confirms the hierarchical levels of an entity. In Figure 28c, the root node object (1) has three levels of the object hierarchy, also the nodes can be deleted and their names can be updated (Figure 28d). The same steps can be applied to create the other entities.

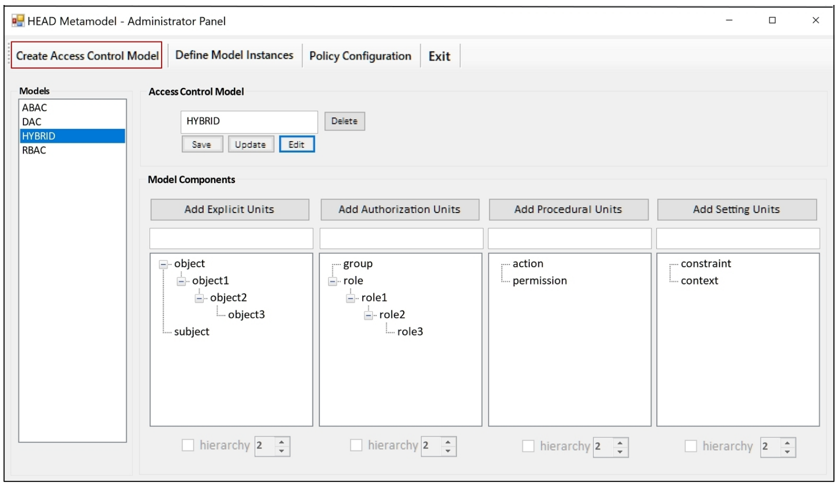
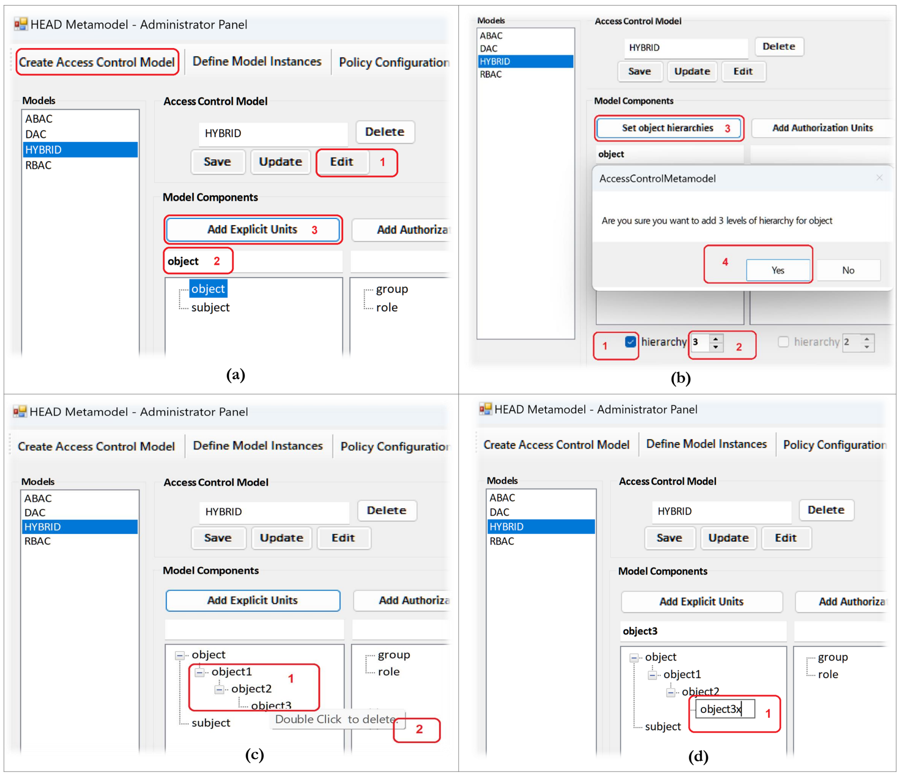


Figure 27. HEAD metamodel: Administrative Panel example.



Are you sure you want to add 3 levels of hierarchy for object

Figure 28. HEAD Administrative Panel: Instantiation of AC model.

Thereafter, in Figure 29 we show how the model entities/attributes can be instantiated. In step (1), we show the defined entities (and hierarchies), of Figure 28. In step (2), we configure a hybrid model for ITMI. To create the required instances of each entity, for example, (3) object entity (at the root level) we define objects names (4) separated by ';' instead of defining each object individually, then (5) we confirm the creation. In step (6), all other elements of subjects, roles, etc., are created in the same way.

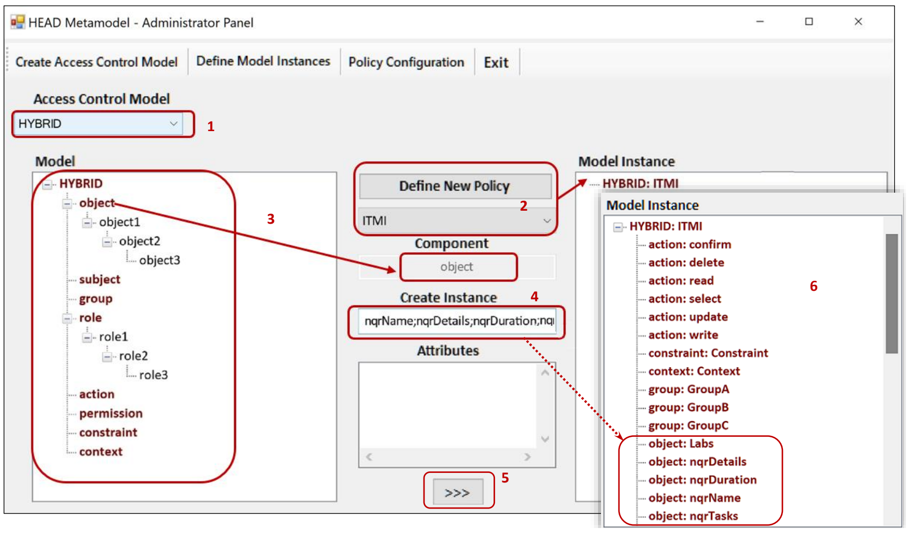


Figure 29. HEAD Administrative Panel: Instantiation of model entities.

In Figure 30, (1) we show how a child node can be created for each a specific node. For example, (2) a child node Manager is created for the root node Director, and (3) shows the hierarchy of roles of case study 1.

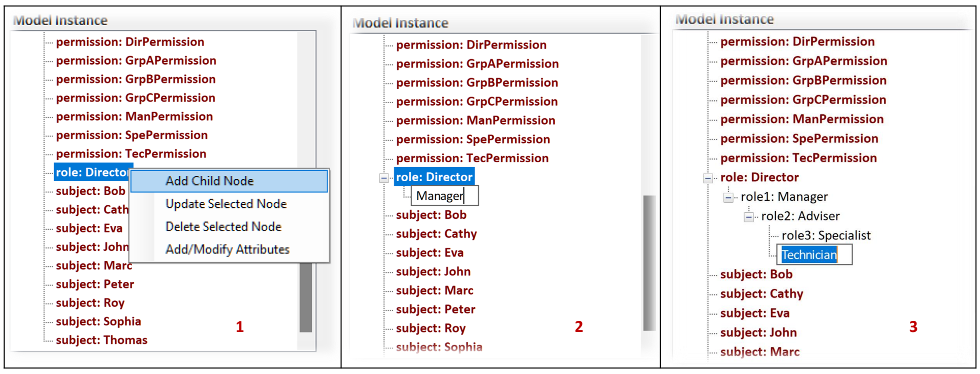
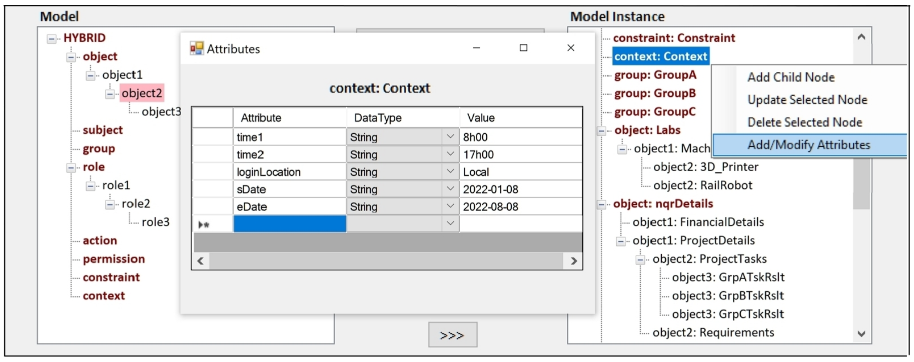


Figure 30. HEAD Administrative Panel: hierarchy of entities.

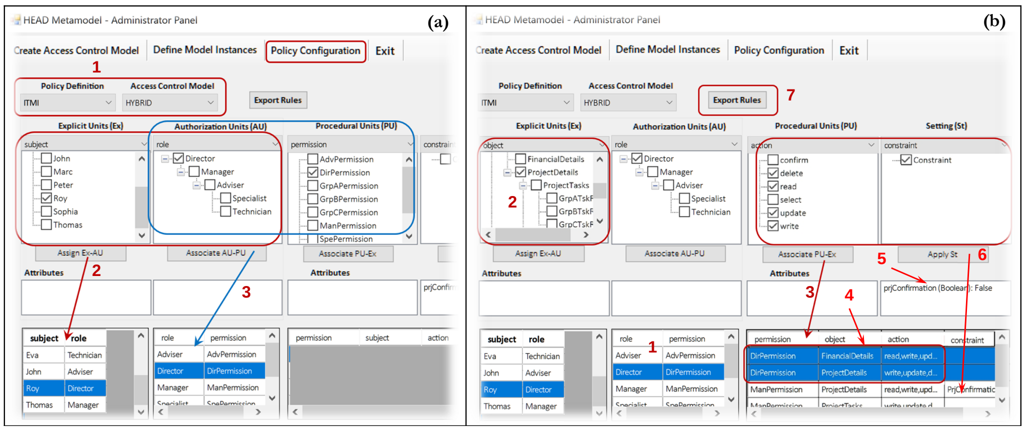
Note that, by default, each entity has a name attribute. In Figure 31, we show how additional attributes can be created. By selecting and right-clicking on a specific element, then choosing 'Add/Modify Attributes' where a popup window opens to define and create various types of attributes (with their values). In our example, we define context attributes to specify if the user needs to access resources within business hours. We also specify the user's login location, and if the current date is between the start/end dates of the project. Moreover, the selected nodes can be updated or deleted.



Attribute	DataType	Value
time1	String	8h00
time2	String	17h00
loginLocation	String	Local
sDate	String	2022-01-08
eDate	String	2022-08-08

Figure 31. HEAD Administrative Panel: adding attributes to entities.

In Figure 32 we show the steps of defining AC rules. In Figure 32a, (1) the system security administrator selects the policy model, which is in case 'Hybrid', to configure the AC rules. Then, he assigns E_x to AUs, in our example, (2) subjects to roles/groups. The administrator (3) associates AUs with PUs, in our example, the role permission. After assigning subjects to roles/groups and specifying their permission(s), in Figure 32b, (1) based on the permission type, (2) the administrator selects which objects may be accessed, then (3) associates the actions that can be performed on each of the selected objects (FinancialDetails and ProjectDetails). To apply constraints, (4) the administrator should select the row (permission, object, and action), then (5) selects the needed attributes to (6) include them with the rule definition. Finally, (7) the rules can be exported as Cypher, json, or any other format of code. In this example, we also generate Cypher statements as explained in Section 5.



subject	role
Eva	Technician
John	Adviser
Roy	Director
Thomas	Manager

role	permission
Adviser	AdvPermission
Director	DirPermission
Manager	ManPermission
Specialist	SpePermission

permission	object	action	constraint
DirPermission	FinancialDetails	read,write,upd...	
DirPermission	ProjectDetails	write,update,d...	
ManPermission	ProjectDetails	read,write,upd...	PrjConfirmatic
ManPermission	ProjectTasks	write,update,d...	

Figure 32. HEAD Administrative Panel: Definition of AC Rules.

8. Evaluation and Validation of HEAD Metamodel

After presenting the above case studies and showing how the HEAD metamodel can derive different AC models, and how it can be adapted to different computing environments. In this section, we present the comparison and evaluation and validation of the HEAD metamodel with the proposed AC metamodels in the literature.

8.1. Comparison

In Table 1 we summarize the features of the HEAD metamodel compared to the other proposed metamodels in the literature [17,18,20].

Table 1. Comparison between HEAD Metamodel and the other AC metamodels.

Metamodel	**Access Control Metamodels**	
Features	**HEAD Metamodel**	**Other Metamodels**
Unify components	Unify all heterogeneous components of different AC models.	Some metamodels unify some heterogeneous components under the notion of 'category' which includes roles, groups, security levels, etc.
Generality	Include all features and components of common AC models and allow deriving various instances of various models.	Hybrid structures to derive some AC models rather than generic metamodels
Dynamism	Allows defining and adding any type of components and attributes for existing models and non-existing ones.	None of the existing metamodels support this feature, and they are not dynamic enough to define static and dynamic AC policies.
Extensibility	New components can be defined and integrated with already-derived models to support new AC features in addition to the previous ones.	Some metamodels are extended but not extensible, and none of the existing metamodels support this feature.
Hierarchical	Allows defining multi-levels of all components (e.g., role, context) to conform to hierarchical organizational structures.	Some metamodels support hierarchy for some components, but none of them consider context hierarchy which is crucial feature in complex and dynamic environments.
Flexibility	Allows creating existing and non existing AC models (new models) and expressing various static and dynamic rules	Only derive some common models. They are limited to the features employed in their structures.
Upgradability	Able to follow technology upgrades and update any policy.	None of the existing metamodels support this feature.
Unified framework	Allows the creation of any model, in addition to any hybrid model with different policy classes (e.g., case study 2), and hybrid models with hybrid components (e.g., case study 1)	Allows the creation of some models based on features employed in their hybrid structures.
Adaptability	Can be adapted to different centralized and distributed environments, especially IoT.	They provide solutions for specific cases and scenarios.
Novelty	A new development in the domain with advanced features.	The last AC metamodel was proposed in 2015 [21].

Overall, the derived models of the HEAD metamodel are flexible and can be easily extended, updated, and defined to follow the technology upgrades and enforce larger sets of static/dynamic policies. Compared to the existing metamodels, system designers and security experts need to redesign a model and rethink how it could be enhanced. This is instead of extending/upgrading it, which means additional time and cost.

8.2. Evaluation and Validation

Comparatively to the proposed metamodels in the literature, HEAD contributes to several essential advancements in the domain. HEAD metamodel allows the specification, formalization, generation, and verification of AC policies. In contrast to tackling each issue separately, the HEAD metamodel draws up a complete strategy. In our experience so far, AC research and practice focus on one phase and confuse issues that cut across multiple phases. However, (1) as for the specification, HEAD metamodel is flexible enough to allow identifying the E_x, I_m, and S_t entities (with their attributes) based on the AC requirements of a system. Using the HEAD metamodel, system administrators and security experts are not restricted to defining some entities for some models. Instead, they can create any entity (and attribute) for any model whether it is an existing AC model or a non-existent model. (2) For formalization, after specifying the needed AC entities, the DSL language of HEAD metamodel allows formalizing the specified AC models (common AC models, hybrid models, and other models) with the ability to follow technology upgrades by allowing the definition of new entities and attributes. The power of the DSL language of the HEAD metamodel is simple and flexible to appropriately express any AC policy requirements, it overcomes the complication of existing language expressions, and also it is independent of specific AC models. (3) For the generation of AC policies, the nature of meta-components of HEAD metamodel which allows deriving any model, can also be adapted to represent the concrete instance of any AC model and generate the needed AC policies. In this paper, we use Eclipse Xtend notation to depict the concrete instances of the derived models. The AC rules are expressed and generated in a format of Cypher queries. Another generator could be used, for example, to represent the needed models and then generate AC rules in as 'json' format. (4) For the formal verification of the generated AC policies, it is necessary to formally verify the accuracy and coherence of the concrete instance of the AC policy before policy enforcement. For example, using Cypher queries as NGAC inputs to represent the AC rules of a system in a graph, then verifying the objects, the relationships between them, and the subjects that interact with the system in a way that adheres to an organization's semantics.

Likewise, the HEAD metamodel serves as a unifying framework and it is the only metamodel that provides to the literature all the design phases and steps starting from the conception [6,15,41–43] to policy enforcement which is addressed in this paper, through the implementation of two case studies, in addition to the new research opportunities this metamodel opens in the domain [20]. The effectiveness of HEAD metamodel is reflected in the proof of concept since we show that the metamodel idea with theoretical foundations can be implemented and applied using different tools. Additionally, if the above case studies are implemented using one of the proposed metamodels, the solution would not meet ITMI's AC requirements, as the existing metamodels only incorporate DAC, MAC, and RBAC features, so attributes cannot be defined nor new components added. Moreover, they are not flexible enough to define different PCs. Moreover, the HEAD metamodel is an essential development in the field since the last proposed metamodel was in 2015 [20], and it is a hybrid metamodel that only includes DAC, MAC, and RBAC, not dynamic, does not support the hierarchy of all components, and the derived models cannot be extended.

It is evident from the above that the HEAD metamodel represents a significant advancement over the other proposed metamodels in the literature, and thus should be viewed as a centerpiece for enhancing other essential characteristics, as well as establishing different research directions in the domain (see [20]).

9. Conclusions and Future Perspectives

Although over the past decade security researchers have proposed a variety of policy models and metamodels to address real-world security problems, the limited ability of the existing AC methods to generically specify, upgrade, and enforce policy persists. With the IoT and industry 4.0, along with the digital transformation, where resources are widely distributed and must be accessed from anywhere, at any time, the need for secure information sharing has increased dramatically. Moreover, the evolution of pervasive ISs and intelligent manufacturing has had a significant impact on different directions, such as the industry's future. Smart industries combine various physical and cyber technologies to enhance productivity, performance, quality, and management. In this context, controlling access to protect resources from unauthorized use is a complicated and challenging task, especially with cybercriminals and cyberattacks. All this has motivated us to design and implement a new and advanced AC metamodel, named HEAD metamodel, that addresses the limitations of the existing metamodels—for example, generality, the hierarchy of components, dynamism, and extensibility of AC models—and works as a base to develop other essential features—for example policy migration, and collaboration and interoperability between AC models.

The AC policy concerns what AC rules need to be enforced, while the AC mechanism concerns how AC rules are enforced. In this paper, we present two case studies inspired by ITMI's local (non-IoT) and IoT computing environments to show that our metamodel can be adapted to different computing environments, and various AC models can be instantiated with the needed components/attributes to fit the AC needs of an organization (or industry sector). For the AC policy, we apply the DSL grammar of HEAD metamodel to derive the needed model for each case study. We use Xtend notation to represent the concert instance of the derived model. The AC rules are generated as Cypher statements which are then injected into Neo4j to represent the NGAC policy as a graph. NGAC framework is used as an enforcement point for the rules generated for each case study. The results show that the HEAD metamodel:

- Overcomes the limitations of the existing AC metamodels and can serve as a basis for other essential features.
- Is a unified framework, and encompasses heterogeneous models.
- Can be adapted into various local and distributed computing environments.
- Can meet current AC requirements and follow policy upgrades.
- Can generate different AC rules formats (e.g., Cypher statements, json. . .).

Consequently, by applying the HEAD metamodel to the presented case studies, we demonstrate that it can be applied to more complex computing environments that include more users, machines, contexts, dynamic devices, and other variables. For example, highly dynamic IIoT environment with various static and dynamic AC rules.

Moreover, the HEAD metamodel reveals some limitations of the NGAC framework. In theory, the NGAC graph includes obligations and prohibitions that express context and constraint elements, but in practice, they are not included. Hence, during implementation, this would make it difficult to make accurate access control decisions. As described in the case studies, we have to create multiple association relationships between UA nodes and OA nodes if some operations need to be performed without constraints by UA, and some other operations can be performed if some contextual (or non-contextual) constraints are true/false depending on the defined rule.

Policies with minimum quality may lead to unacceptable situations and decisions, for example preventing users from accessing resources they are allowed to, or allowing some users to access resources they are not allowed to. As future perspectives, we aim to develop the needed methods to analyze and assess the obtained policies at run-time before enforcing them to avoid uncertainties concerning the obtained AC decision. Policy analysis and assessment are intended to ensure the quality of the generated AC policies. They are for ensuring they are consistent, relevant, minimal, complete, and correct with respect to the required actions by subjects on some objects. This process is of major importance while implementing AC policies in highly dynamic and heterogeneous environments, especially IoT. Moreover, we also aim to support the HEAD metamodel with additional features and services. For example, developing the needed algorithms (with theoretical foundations) and methods to migrate AC policies from one model to another. Providing packages of predefined AC models (e.g., common models) as support for the metamodel. This would lessen technical implementation efforts and facilitate administrative efforts to specify models and define policies. Further, we plan to extend the NGAC framework to adapt context and constraint in addition to policy.

Author Contributions: Conceptualization, N.K. and M.A.; methodology, N.K., M.A. and H.I.; software, N.K.; validation, N.K., M.A., H.I., J.-F.M. and T.D.; formal analysis, N.K. and M.A.; investigation, N.K., M.A., H.I., J.-F.M. and T.D.; resources, N.K., M.A., H.I., J.-F.M. and T.D.; data curation, N.K. and J.-F.M.; writing—original draft preparation, N.K.; writing—review and editing, N.K., M.A. and H.I.; visualization, N.K., M.A. and H.I.; supervision, M.A. and H.I.; project administration, M.A. and H.I. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Natural Sciences and Engineering Research Council of Canada (NSERC), grant number 06351.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The study did not report any data.

Acknowledgments: We acknowledge the support of Fonds Québécois de la Recherche sur la Nature et les Technologies (FRQNT), Réseau Québécois sur l'Énergie Intelligente (RQEI), and Centre d'Entrepreneuriat et de Valorisation des Innovations (CEVI).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ravidas, S.; Lekidis, A.; Paci, F.; Zannone, N. Access control in Internet-of-Things: A survey. *J. Netw. Comput. Appl.* **2019**, *144*, 79–101. [CrossRef]
2. Zhang, Y.; Li, B.; Liu, B.; Wu, J.; Wang, Y.; Yang, X. An Attribute-Based Collaborative Access Control Scheme Using Blockchain for IoT Devices. *Electronics* **2020**, *9*, 285. [CrossRef]
3. Ndibanje, B.; Lee, H.J.; Lee, S.G. Security Analysis and Improvements of Authentication and Access Control in the Internet of Things. *Sensors* **2014**, *14*, 14786–14805. [CrossRef] [PubMed]
4. Antunes, M.; Maximiano, M.; Gomes, R.; Pinto, D. Information Security and Cybersecurity Management: A Case Study with SMEs in Portugal. *J. Cybersecur. Priv.* **2021**, *1*, 219–238. [CrossRef]
5. Jaïdi, F.; Labbene Ayachi, F.; Bouhoula, A. A methodology and toolkit for deploying reliable security policies in critical infrastructures. *Secur. Commun. Netw.* **2018**, *2018*, 7142170. [CrossRef]
6. Kashmar, N.; Adda, M.; Atieh, M.; Ibrahim, H. Access control metamodel for policy specification and enforcement: From conception to formalization. *Procedia Comput. Sci.* **2021**, *184*, 887–892. [CrossRef]
7. Mishra, A.; Alzoubi, Y.I.; Gill, A.Q.; Anwar, M.J. Cybersecurity Enterprises Policies: A Comparative Study. *Sensors* **2022**, *22*, 538. [CrossRef]
8. Narouei, M.; Khanpour, H.; Takabi, H. Identification of access control policy sentences from natural language policy documents. In Proceedings of the IFIP Annual Conference on Data and Applications Security and Privacy, Philadelphia, PA, USA, 19–21 July 2017; Springer: Cham, Switzerland, 2017; pp. 82–100.
9. Chaudhry, S.A.; Yahya, K.; Al-Turjman, F.; Yang, M.H. A secure and reliable device access control scheme for IoT based sensor cloud systems. *IEEE Access* **2020**, *8*, 139244–139254. [CrossRef]
10. Neisse, R.; Steri, G.; Fovino, I.N.; Baldini, G. SecKit: A model-based security toolkit for the internet of things. *Comput. Secur.* **2015**, *54*, 60–76. [CrossRef]
11. Cruz-Piris, L.; Rivera, D.; Marsa-Maestre, I.; De La Hoz, E.; Velasco, J.R. Access control mechanism for IoT environments based on modelling communication procedures as resources. *Sensors* **2018**, *18*, 917. [CrossRef]
12. Kukhun, D.A. Steps towards Adaptive Situation and Context-Aware Access: A Contribution to the Extension of Access Control Mechanisms within Pervasive Information Systems. Ph.D. Thesis, Toulouse 3, Toulouse, France, 2012.
13. Ulltveit-Moe, N.; Nergaard, H.; Erdödi, L.; Gjøsæter, T.; Kolstad, E.; Berg, P. Secure information sharing in an industrial Internet of Things. *arXiv* **2016**, arXiv:1601.04301.
14. Salonikias, S.; Gouglidis, A.; Mavridis, I.; Gritzalis, D. Access control in the industrial internet of things. In *Security and Privacy Trends in the Industrial Internet of Things*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 95–114. [CrossRef]
15. Kashmar, N.; Adda, M.; Ibrahim, H. HEAD Metamodel: Hierarchical, Extensible, Advanced, and Dynamic Access Control Metamodel for Dynamic and Heterogeneous Structures. *Sensors* **2021**, *21*, 6507. [CrossRef]
16. Wang, H.; Sun, L.; Bertino, E. Building access control policy model for privacy preserving and testing policy conflicting problems. *J. Comput. Syst. Sci.* **2014**, *80*, 1493–1503. [CrossRef]
17. Kashmar, N.; Adda, M.; Ibrahim, H. Access Control Metamodels: Review, Critical Analysis, and Research Issues. *J. Ubiquitous Syst. Pervasive Netw.* **2021**, *3*. [CrossRef]
18. Kashmar, N.; Adda, M.; Atieh, M.; Ibrahim, H. A review of access control metamodels. *Procedia Comput. Sci.* **2021**, *184*, 445–452. [CrossRef]
19. Yang, Q.; Zhang, M.; Zhou, Y.; Wang, T.; Xia, Z.; Yang, B. A Non-Interactive Attribute-Based Access Control Scheme by Blockchain for IoT. *Electronics* **2021**, *10*, 1855. [CrossRef]
20. Kashmar, N.; Adda, M.; Ibrahim, H. HEAD Access Control Metamodel: Distinct Design, Advanced Features, and New Opportunities. *J. Cybersecur. Priv.* **2022**, *2*, 42–64. [CrossRef]
21. Abd-Ali, J.; El Guemhioui, K.; Logrippo, L. A Metamodel for Hybrid Access Control Policies. *J. Softw.* **2015**, *10*, 784–797. [CrossRef]
22. Slimani, N.; Khambhammettu, H.; Adi, K.; Logrippo, L. UACML: Unified access control modeling language. In Proceedings of the 2011 4th IFIP International Conference on New Technologies, Mobility and Security, Paris, France, 7–10 February 2011; pp. 1–8. [CrossRef]
23. Korman, M.; Lagerström, R.; Ekstedt, M. Modeling Enterprise Authorization: A Unified Metamodel and Initial Validation. *Complex Syst. Inform. Model. Q.* **2016**, *7*, 1–24. [CrossRef]
24. Paige, R.F.; Kolovos, D.S.; Polack, F.A. A tutorial on metamodelling for grammar researchers. *Sci. Comput. Program.* **2014**, *96*, 396–416. [CrossRef]
25. Bettini, L. *Implementing Domain-Specific Languages with Xtext and Xtend*; Packt Publishing Ltd.: Birmingham, UK, 2016.
26. Kovacevic, D.; Krunic, M.; Cetic, N.; Kovacevic, J. Xtext-based eclipse editor for linker configuration file. In Proceedings of the 2015 23rd Telecommunications Forum Telfor (TELFOR), Belgrade, Serbia, 24–26 November 2015; pp. 862–865.
27. Ferraiolo, D.; Chandramouli, R.; Kuhn, R.; Hu, V. Extensible access control markup language (XACML) and next generation access control (NGAC). In Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control, New Orleans, LA, USA, 11 March 2016; pp. 13–24.

28. Basnet, R.; Mukherjee, S.; Pagadala, V.M.; Ray, I. An efficient implementation of next generation access control for the mobile health cloud. In Proceedings of the 2018 Third International Conference on Fog and Mobile Edge Computing (FMEC), Barcelona, Spain, 23–26 April 2018; pp. 131–138.
29. Ray, I.; Alangot, B.; Nair, S.; Achuthan, K. Using attribute-based access control for remote healthcare monitoring. In Proceedings of the 2017 Fourth International Conference on Software Defined Systems (SDS), Valencia, Spain, 8–11 May 2017; pp. 137–142.
30. Abdullahi, M.; Baashar, Y.; Alhussian, H.; Alwadain, A.; Aziz, N.; Capretz, L.F.; Abdulkadir, S.J. Detecting Cybersecurity Attacks in Internet of Things Using Artificial Intelligence Methods: A Systematic Literature Review. *Electronics* **2022**, *11*, 198. [CrossRef]
31. Quader, F.; Janeja, V.P. Insights into Organizational Security Readiness: Lessons Learned from Cyber-Attack Case Studies. *J. Cybersecur. Priv.* **2021**, *1*, 638–659. [CrossRef]
32. Leander, B.; Čaušević, A.; Hansson, H.; Lindström, T. Toward an ideal access control strategy for industry 4.0 manufacturing systems. *IEEE Access* **2021**, *9*, 114037–114050. [CrossRef]
33. Andaloussi, Y.; El Ouadghiri, M.D.; Maurel, Y.; Bonnin, J.M.; Chaoui, H. Access control in IoT environments: Feasible scenarios. *Procedia Comput. Sci.* **2018**, *130*, 1031–1036. [CrossRef]
34. Kayes, A.S.M.; Kalaria, R.; Sarker, I.H.; Islam, M.S.; Watters, P.A.; Ng, A.; Hammoudeh, M.; Badsha, S.; Kumara, I. A Survey of Context-Aware Access Control Mechanisms for Cloud and Fog Networks: Taxonomy and Open Research Issues. *Sensors* **2020**, *20*, 2464. [CrossRef]
35. Desmedt, Y.; Shaghaghi, A. Function-Based Access Control (FBAC): Towards Preventing Insider Threats in Organizations. In *From Database to Cyber Security*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 143–165.
36. Qi, S.; Zheng, Y.; Li, M.; Liu, Y.; Qiu, J. Scalable industry data access control in RFID-enabled supply chain. *IEEE/ACM Trans. Netw.* **2016**, *24*, 3551–3564. [CrossRef]
37. Ruland, C.; Sassmannshausen, J. Access control in safety critical environments. In Proceedings of the 2018 12th International Conference on Reliability, Maintainability, and Safety (ICRMS), Shanghai, China, 17–19 October 2018; pp. 223–229.
38. Alagar, V.; Alsaig, A.; Ormandjiva, O.; Wan, K. Context-based security and privacy for healthcare IoT. In Proceedings of the 2018 IEEE International Conference on Smart Internet of Things (SmartIoT), Xi'an, China, 17–19 August 2018; pp. 122–128.
39. Ahamed, J.; Khan, F. An enhanced context-aware capability-based access control model for the internet of things in healthcare. In Proceedings of the 2019 Sixth HCT Information Technology Trends (ITT), Ras Al Khaimah, United Arab Emirates, 20–21 November 2019; pp. 126–131.
40. Mrabet, H.; Alhomoud, A.; Jemai, A.; Trentesaux, D. A Secured Industrial Internet-of-Things Architecture Based on Blockchain Technology and Machine Learning for Sensor Access Control Systems in Smart Manufacturing. *Appl. Sci.* **2022**, *12*, 4641. [CrossRef]
41. Kashmar, N.; Adda, M.; Atieh, M.; Ibrahim, H. A new dynamic smart-AC model methodology to enforce access control policy in IoT layers. In Proceedings of the 2019 IEEE/ACM 1st International Workshop on Software Engineering Research & Practices for the Internet of Things (SERP4IoT), Montreal, QC, Canada, 27 May 2019; pp. 21–24.
42. Kashmar, N.; Adda, M.; Atieh, M.; Ibrahim, H. Smart-ac: A new framework concept for modeling access control policy. *Procedia Comput. Sci.* **2019**, *155*, 417–424. [CrossRef]
43. Kashmar, N.; Adda, M.; Atieh, M.; Ibrahim, H. Deriving access control models based on generic and dynamic metamodel architecture: Industrial use case. *Procedia Comput. Sci.* **2020**, *177*, 162–169. [CrossRef]