Article

Context-Aware Policy Analysis for Distributed Usage Control

Gonzalo Gil [1,*], Aitor Arnaiz [1], Mariví Higuero [2], Francisco Javier Diez [1] and Eduardo Jacob [2]

1 Tekniker, Basque Research and Technology Alliance (BRTA), Iñaki Goenaga 5, 20600 Eibar, Spain
2 Escuela de Ingeniería de Bilbao, Plaza Ingeniero Torres Quevedo 1, 48013 Bilbao, Spain
* Correspondence: gonzalo.gil@tekniker.es

Abstract: To boost data spaces and benefit from the great opportunities that they present, data sovereignty must be provided by Distributed Usage Control (DUC). Assuming that DUC will be managed by implementing and enforcing policies, notable efforts have already been undertaken in the context of Access Control (AC) regarding policy analysis due to the impact of low-quality policies on security. In this regard, this paper proposes that policy analysis in the DUC context should be understood as an extension of the AC, which is further affected by other challenging features, chief among which are context-aware control and extended control through action requirements. This paper presents a novel Context-Aware Policy Analysis (CAPA) algorithm for detecting inconsistencies and redundancies for DUC policies by supporting a large set of heterogeneous conditions. In this regard, the dependent relationship of conditions is formulated which will lead to more efficient conflict detection. By implementing this concept, a novel tree structure that combines a resource and a policy structure is presented to search for and compare relevant rules from policies. Built on the tree structure and through the formalization of rule conflicts, CAPA is developed and the security and performance it provides is tested in a wind energy use case.

Keywords: data sovereignty; distributed usage control; policy quality; energy data; conditions

check for updates

Citation: Gil, G.; Arnaiz, A.; Higuero, M.; Diez, F.J.; Jacob, E. Context-Aware Policy Analysis for Distributed Usage Control. *Energies* **2022**, *15*, 7113. https://doi.org/10.3390/en15197113

Academic Editor: Islam Md Rizwanul Fattah

Received: 22 July 2022
Accepted: 22 September 2022
Published: 27 September 2022

1. Introduction

In a study conducted by the European Commission [1] in which many companies of different sizes and sectors were interviewed, companies recognized the importance of Business-to-Business (B2B) data sharing. Among others, the following benefits stand out: improvements in the design of products and services, the generation of new business models, and the establishment of new partnerships. As a result, the European Commission is promoting the development of European Data Spaces [2] where companies can participate in distributed collaborative ecosystems as data providers by making their data available under certain requirements, and as data consumers by exploiting the data available. This serves to boost the data economy, competitiveness, and innovation at the European level.

In the energy sector, being able to process data acquired from different stakeholders in the course of power generation, transmission, transformation, distribution, and consumption would significantly advance knowledge to improve the performance of the value chain [3]. However, stakeholders are reluctant to share data if they lose control over its use once it reaches the distributed consumer to be exploited. As a result, knowledge extraction from data becomes a challenge. In this regard, ensuring data sovereignty [4], defined as the self-determination of individuals and organizations regarding the use of their data, becomes the key enabler to boost data sharing, enable data processing, and gain knowledge to take a big step in the energy sector. This is the case in areas such as wind energy generation, where recent analyses have highlighted data sharing as an important drawback [5–7]. For instance, access to data from e.g., turbine performance, usually retained by wind farm operators and OEMs (original equipment manufacturers), by different stakeholders (e.g., component OEMs) would optimize the maintenance & operation of the value chain [8]. Data sovereignty calls for future control over the usage of the data in distributed infrastructures, also denoted as Distributed Usage Control (DUC) [9]. In this sense, Access Control (AC) models can be considered a good starting point from which to address the development of DUC approaches.

Assuming that DUC will be managed through the implementation and enforcement of sticky policies [10], AC research literature [11] has already demonstrated that expressing policies without conflicts, i.e., providing good quality, is not trivial. Furthermore, low-quality policies have a great impact on:

- Security: leading to unauthorized data disclosure and/or denial of legal access, thus affecting data sovereignty.
- Performance: requiring longer policy enforcement times, which affects data sharing performance.

Thus, ensuring security in DUC through policy quality becomes crucial to ensure data sovereignty and to encourage data sharing.

The DUC model can be considered as an extension of AC models [12], but it introduces some significant features that further affect policy quality and render the adoption of existing AC policy analysis methods [13,14] non-feasible. These are the following:

- Context-aware control: due to the importance of context information for policy enforcement in distributed infrastructures, permission and prohibition rules are refined through much more numerous and diverse conditions than in AC. That is, while some conditions may apply to different domains (e.g., time or location), as they are non-dependent, dependent conditions such as those related to time can be heterogeneously described (e.g., time intervals or an event in time).
- Extended control by supporting action requirements: to further control distributed data usage, permission rules are refined by supporting duty rules that define the actions that must be executed before and after data usage is granted.

This paper addresses data sovereignty by providing security in DUC. Addressing both new features of DUC is of great importance, but we focus on analyzing the rules within policies that support context-aware control. In this regard, we take the policy quality requirements already defined in AC [11], ensure policy correctness by implementing semantic technologies, and we address security through a policy-analysis-based approach that ensures policy consistency and minimality. Consistency ensures that two policies do not express contradictory rights that may lead to unauthorized data disclosure, while minimality makes sure that there are no redundant policies, so security risks are avoided in policy management. Policy completeness and relevance may certainly affect security but providing them requires a data usage transaction-based analysis which lies outside the scope of this paper.

Following a logical approach, we face the challenge of supporting DUC-specific features based on existing AC policy analysis approaches. In this regard, we consider tree-based modeling to be the most suitable approach, because it enables all consistency and minimality conflicts to be detected. This means that security is always ensured. Thus, it guarantees data sovereignty, which is of utmost importance for DUC to reduce the reluctance to share data. This approach can also be adapted to the complexity introduced by conditions on rules. On this basis, this paper makes the following contributions:

- A novel tree structure combining a Policy Tree (PT) and a Resource Tree (RT) is presented. Using this tree structure, security can be assessed and provided by efficiently analyzing consistency and minimality for DUC with the support of context-aware control. On the one hand, by identifying, formulating, and implementing the dependent relationship of conditions, relevant rules are efficiently searched and irrelevant analyses are thus avoided. On the other hand, dependent conditions are stored in reference formats. In this way, efficient comparisons can be made between heterogeneous conditions.
- A method for assessing rule conflicts in the form of inconsistencies and redundancies is formalized. In this regard, depending on whether rules are of permission or pro-

hibition, the inferences of their refinement with conditions (e.g., time intervals) for dependent but non-overlapping conditions (the rest of the time) are considered.

- A Context-Aware Policy Analysis (CAPA) algorithm is presented. It is built on the tree structures presented and implements the proposed assessment method for rule conflicts.

Lastly, to assess the feasibility of this approach, CAPA is formalized and implemented in a real wind energy use case. It is demonstrated that security can be provided. Thus, ensuring data sovereignty, leading to the sharing of data from wind turbine performance along the value chain, which will allow wind farm maintenance and operation to be optimized. Its own performance is also analyzed.

The rest of the paper is structured as follows: Section 2 reviews existing work on policy quality in DUC. Due to the lack of research in this domain, related work on AC is also reviewed with special focus on the tree-based modeling approach. Section 3 defines and formulates a DUC model encompassing existing models, thus generalizing the contributions presented throughout the paper. Section 4 introduces several definitions, which are used to present the tree structure and the rule conflict assessment method that enable policy analysis algorithms to be developed. To enable an efficient policy analysis to be conducted, Section 5 describes the tree structure composed of an RT and a PT. Section 6 sets out the assessment method for conflicts between rules. Section 7 presents CAPA, based on the tree structure explained in Section 5, and implements the rule conflict assessment method defined in Section 6. Section 8 confirms the feasibility of CAPA in a wind energy use case by presenting its ability to ensure security and by analyzing its performance. Finally, conclusions about this work and future lines of research are highlighted.

2. Related Work

Below, we provide an overview of the related work published so far about policy quality in DUC. The related work on policy quality in AC is considered of great interest as a starting point for developing DUC solutions, so it is also analyzed, with particular focus on the tree-based modeling approach.

2.1. DUC Policy Quality

Lazouski et al. [15] conducted a survey on the pioneering Usage Control (UCON) model [16]. This paper identifies security analysis, policy completeness, and consistency as important fields of research within usage control that have not yet been studied but that need to be addressed as fundamental points.

Kelbert et al. [17] present the Distributed Usage Control Framework. For the implementation of DUC policies, the Obligation Specification Language (OSL) [18] was followed. However, policy quality analysis is not addressed.

The International Data Spaces Association (IDSA) Reference Architecture Model [19], based on the reference Open Digital Rights Language (ODRL) information model (IM) (https://www.w3.org/TR/odrl-model/ accessed on 26 September 2022), formalizes the IDSA usage policy language (UPL) within the IDSA IM (https://w3id.org/idsa/core accessed on 26 September 2022) for the implementation of DUC policies [20]. Furthermore, built on the IDSA UPL, Bader et al. [21] present an overview of the challenges posed by DUC policies if they are to be enforceable. However, this work is incomplete and policy quality is not mentioned.

2.2. AC Policy Quality

In AC research literature, major efforts have been dedicated to policy quality assessment. In this regard, Aqib et al. [13] report the first survey on policy analysis mechanisms. This paper finds that policy can be analyzed through different approaches, denoted as formal methods, model checking, matrix-based approaches, mining techniques, mutation testing techniques, and others. Policy quality requirements are also limited to consistency and completeness. In this regard, based on existing work, each mechanism is reviewed and classified according to the policy analysis strategy used and the policy quality requirements addressed. Later, Bertino et al. [11] analyze policy quality in more depth. As a result, policy quality requirements are extended by including minimality, relevance, and correctness.

Based on these defined policy quality requirements and the increased number of papers on policy analysis in AC, Jabal et al. [14] conducted a more extensive and mature survey. Formal methods, model checking, data mining, graph/tree-based modeling, and mutation testing were identified as the most widely used approaches for policy analysis. Additionally, the large body of existing research was reviewed and classified based on the approach used and the policy quality requirements addressed. As a result, a comparison was drawn between the most widely used approaches and the policy quality requirements most widely addressed.

Because of the extensive research on policy analysis for AC, gathered in the previous survey, and due to some common features considered of interest for our approach for analyzing the quality of policies in the DUC context, we will deepen and narrow the research on specific tree-based modeling policy analysis approaches, with a special focus on papers that address context-aware control.

Chao et al. [22] formally define inconsistency for the role-based AC model [23] and address it through a checking algorithm that analyzes a set of rules defined for hierarchical classified roles. However, consistency is highly oriented to ensure the concepts of separation of duty and the cardinality constraint.

Sun et al. [24] present a conflict detection algorithm to ensure consistency in the purpose AC model by supporting hierarchically classified usage purposes. Specifically, inconsistencies are defined in a novel way. It is defined that if a usage purpose is already permitted for a resource, dependent usage purposes are denied by inference for the same resource and cannot be permitted. Otherwise, an inconsistency is detected. In this approach, it is assumed that rules are always defined as permissions.

Aqib et al. [25] present an algorithm for detecting inconsistencies for XACML [26], addressing rule refinement through Boolean expressions. However, it is limited to time-related Boolean expressions, which are always expressed as time ranges. Inconsistency is defined as contrary decisions for the same resource in overlapping time intervals. Thus, conditions are oversimplified because their inferences for dependent but non overlapping conditions are not considered.

Finally, although the concept of dependent relationship of resources is presented by Mohan et al. [27], Deng et al. [28] address it in depth and the transmission relationship of access authority is presented. In this regard, a conflict algorithm is presented for XACML by supporting time-based conditions expressed through time ranges. This paper focuses on detecting inconsistencies not only for common resources but also for dependent resources. However, inconsistencies are limited to overlapping time spans. Therefore, the concept of condition is also oversimplified, as inferences are not considered.

In short, from the AC research literature, the algorithms presented for policy analysis following the tree-based modeling approach are oriented towards providing policy consistency. Furthermore, only one condition is addressed in each algorithm. This means that roles are supported for role-based AC, usage purpose is addressed for purpose AC, and time is considered for XACML. Moreover, conditions are always expressed homogeneously. Roles and usage purposes are hierarchically classified, and time is expressed through time ranges. In addition, the concept of condition is oversimplified. Depending on whether rules are of permission or prohibition, their refinement with conditions has inferences for dependent but non-overlapping conditions. However, this issue is not considered for conflict detection. Finally, the dependent relationship of resources is introduced for XACML through the transmission relationship of access authority. However, it has all the above-mentioned limitations.

That said, although these papers can be considered a good starting point for building DUC solutions and providing support for DUC specific features, the importance of context-aware control for DUC means that policy analysis must be addressed by managing a broad set of heterogeneous conditions. Otherwise, policy quality will not be completely ensured.

In this regard, we present CAPA in this paper: a policy analysis algorithm which seeks to provide security in DUC through policy consistency and minimality for context-aware policies. CAPA is built on a tree-based structure that implements the dependent relationship of resources and conditions and thus enables conflicts between rules refined through multiple heterogeneous conditions to be detected efficiently. It also detects inconsistencies and redundancies by considering the inferences of rule refinement. Thus, security is ensured through an algorithm that also provides good performance.

3. Distributed Usage Control Model

The DUC model has followed an incremental approach in which the pioneering UCON model has evolved into more refined and expressive models. Cases in point are the reference ODRL IM and, later, the IDSA UPL which, formalized within the IDSA IM, is the most mature DUC model to date. On that basis, we define a generic DUC model below that encompasses all the above.

Formula (1) formalizes the set of policies $p \in P$ that are composed of the subset of rules $r \in R$. These rules will be analyzed for policy consistency and minimality.

$$p = \{r_i\},\ i = 1,\ \ldots,\ N \tag{1}$$

where N is the number of rules.

A rule r_i describes the usage control statement related to permission for or prohibition of data usage and the duty that DUC may be required to perform under a permission. It is described through the $type_i = \{permission,\ prohibition,\ duty\}$, $sub_i\ \in S$, $act_i\ \in A$, $res_i\ \in Res$ and $C_i\ \subset C$ which refer respectively to the rule type, subject, action, resource, and the set of conditions so that Formula (2) is satisfied.

$$r_i = (type_i,\ sub_i,\ act_i,\ res_i,\ C_i),\ i = 1, \ldots,\ N \tag{2}$$

$$C_i = \{con_j\},\ j = 1,\ \ldots,\ M$$

where M is the number of conditions.

The rule type $type_i$ refers to whether the usage control statement is a permission, prohibition, or duty. Action requirements specified as duty rules in the DUC model are an additional feature that further affects policy quality. Addressing them in policy analysis is a very interesting line of research, but the scope of this paper is focused on context-aware control, so only permission and prohibition rules are considered in the rest of the paper.

A subject sub_i represents a participant within a data usage relationship. These may have the role of assignee or assigner. The assigner is responsible for defining r_i, while the assignee is responsible for its enforcement. Thus, the assigner has no impact on policy quality. Therefore, in the rest of the paper sub_i always refers to the assignee.

An action act_i describes the activity that a sub_i is permitted or prohibited from performing.

A resource res_i represents the target digital content to which a r_i applies.

Conditions C_i describe the specifications under which r_i applies. Each $con_j\ \in C_i$ is in the form of a Boolean expression, and thus composed of two operands ($leftOperand_j$ and $rightOperand_j$) compared by an operator $(operator_j)$ which results in either true or false. Thus, we define condition con_j satisfying Formula (3).

$$con_j = \{leftoperand_j,\ operator_j,\ rightOperand_j\} \tag{3}$$

The rest of our work is presented below, based on this general DUC model.

4. Definitions

This section presents some definitions and axioms which constitute basic aspects and statements assumed in our approach.

Definition 1. ***Atomic Condition Rule:*** *if a rule r_1 is refined by only one condition such that $M = 1$, we define that r_1 is an atomic condition rule.*

Definition 2. ***Composite Condition Rule:*** *if a rule r_1 is refined by more than one condition such that $M > 1$, we define that r_1 is a composite condition rule.*

Axiom 1. ***Rule Condition Atomization:*** *given a composite condition rule $r_1 = (type_1,\ sub_1,\ res_1,\ act_1,\ C_1 = \{con_{11},\ con_{12}\})$, we can split it into atomic condition rules $r_{11} = (type_1,\ sub_1,\ res_1,\ act_1,\ \{con_{11}\})$ and $r_{12} = (type_1,\ sub_1,\ res_1,\ act_1,\ \{con_{12}\})$ maintaining the entire scope.*

Thus, a set of rules R_1, which can be refined by atomic conditions and/or composite conditions can be split into a set of atomic condition rules R_2. From R_2, the tree structure is built, and the rule conflict assessment method is applied, enabling policy analysis algorithms to be developed. Thus, hereinafter we refer to atomic condition rules.

Definition 3. ***Dependent Relationship of Conditions:*** *for conditions con_1 and con_2, we define them as satisfying the dependent relationship of conditions $con_1 \leftrightarrow con_2$ if they apply to the same domain, such as time, so that they may overlap. Furthermore, we define them as applying to the same domain and thus, that the dependent relationship of conditions is satisfied if their corresponding $leftoperand_1$ and $leftoperand_2$ are related to the same application domain L_{ad} so that Formula (4) is met.*

$$con_1 \leftrightarrow con_2 = (leftoperand_1 \in L_{ad} \wedge leftoperand_2 \in L_{ad}) \tag{4}$$

For example, two conditions con_1 and con_2 which refine r_1 and r_2 apply to the same domain if their corresponding $leftoperand_1 = current\ time\ \in L_{time}$ and $leftoperand_2 = duration\ \in L_{time}$ are related to time. However, con_1 and con_3 which refine r_1 and r_3 are not dependent, for example, if $leftoperand_1 = current\ time\ \in L_{time}$ and $leftoperand_3 = connector \in L_{connector}$ are respectively related to time and the connector.

Definition 4. ***Overlapping Relationship of Conditions:*** *for conditions con_1 and con_2, if $con_1 \cap con_2 \neq \varnothing$, we consider that con_1 and con_2 satisfy the overlapping relationship of conditions.*

$$con_1 \cap con_2 \neq \varnothing \tag{5}$$

Definition 5. ***Conditions Whitelisting Approach:*** *in this approach, policies are described only through permission rules that explicitly define all the conditions that must be met to allow data usage. This leads to Axiom 2, which we define as follows.*

Axiom 2. *If a rule r_1 of $type_1 = permission$ is set for subject sub_1 on resource res_1 with action act_1 given a condition con_1 related to a specific application domain such that $leftoperand_1 \in L_{ad_1}$ as shown in Formula (6), the enforcement of r_1 for dependent but non-overlapping conditions with respect to con_1 will always lead to a prohibition. Furthermore, complementary permissions such as for con_2 can be defined on dependent and overlapping conditions so that permission is refined at $con_1 \cap con_2$.*

$$r_1 = (type_1 = permission,\ sub_1,\ res_1,\ act_1,\ con_1 = (leftOperand_1 \in L_{ad1},\ operator_1,\ rightOperand_1)) \tag{6}$$

For example, the enforcement of a rule r_1 of the permission type leads to permission from 1 January 2022 to 1 January 2023, but condition enforcement at any other time will result in a prohibition. Moreover, a permission rule r_2 can be defined as complementary at maintenance time so that permission is only granted at maintenance time between 1 January 2022 and 1 January 2023.

Definition 6. ***Conditions Blacklisting Approach:*** *following this approach data usage is granted by default, and only prohibition rules are described, so that if only one prohibition is met, i.e., if only one condition is met, data usage is blocked. From this, we define Axiom 3.*

Axiom 3. *If a rule r_1 of $type_1 = prohibition$ is set for subject sub_1 on resource res_1 with action act_1 given a condition con_1 related to a specific application domain such that $leftoperand_1 \in L_{ad_1}$ as shown in Formula (7), the enforcement of r_1 for non-overlapping but dependent conditions with respect to con_1 will lead to permission by default. However, complementary prohibitions such as for con_2 can be defined on dependent but non-overlapping conditions, so that prohibition is extended to $con_1 + con_2$.*

$$r_1 = (type_1 = prohibition,\ sub_1,\ res_1,\ act_1,\ con_1 = (leftOperand_1 \in L_{ad1},\ operator_1,\ rightOperand_1)) \tag{7}$$

For example, the enforcement of a rule r_1 of type prohibition leads to a prohibition for a Connector A, while a rule r_2 also of the prohibition type will complement the previous one, describing a prohibition also for Connector B. However, for all other connectors data usage is permitted.

Definition 7. ***Dependent Relationship of Resources:*** *for resources res_i and res_j, if res_j is part of res_i so that $res_i \rightarrow res_j$, we say that they satisfy the dependent relationship of resources.*

$$res_i \rightarrow res_j \tag{8}$$

Definition 8. ***Time Complexity:*** *This measures the time required to execute an algorithm. It is measured as the number of times that the statements of an algorithm are executed. It is expressed using the big O annotation.*

5. Tree Structures for Efficient Policy Analysis

Our approach is based on building tree structures for resources and the rules defined on them. This section presents some initial concerns related to efficient policy analysis and the most significant features of the trees.

For N atomic condition rules, conflicts can be detected by brute force by making a total of $N * (N - 1)$ comparisons between each of the rules. However, that results in poor performance for the following two reasons:

1. **Irrelevant analysis of rules that never lead to conflict:** For two rules, r_1 and r_2, inconsistencies and redundancies appear only for the same subject and action, so $sub_1 = sub_2 \wedge act_1 = act_2$. Furthermore, they may appear for common resources $res_1 = res_2$ as well as for dependent resources $res_1 \rightarrow res_2$ As a result, only rules satisfying Formulas (9) and (10) must be analyzed for common and dependent resource conflicts respectively.

$$sub_i = sub_j \wedge act_i = act_j \wedge res_i = res_j \tag{9}$$

$$sub_i = sub_j \wedge act_i = act_j \wedge res_i \rightarrow res_j \tag{10}$$

These rules that must always be analyzed because they may lead to conflicts that are denoted below as relevant rules.

In DUC, with the refinement of rules through a wide set of conditions, not only does the number of rules increase considerably but rules for a subject and action also apply to different condition application domains which are non-dependent and thus never lead to a conflict, so their analysis is irrelevant. From this, we define Axiom 4.

Axiom 4. ***Efficient analysis of DUC relevant rules.*** *To efficiently detect common and dependent resource conflicts in DUC, only relevant rules that satisfy Formulas (11) and (12) respectively, which introduce the dependent relationship of conditions, must be analyzed.*

$$sub_i = sub_j \wedge act_i = act_j \wedge res_i = res_j \wedge con_i \rightarrow con_j \tag{11}$$

$$sub_i = sub_j \wedge act_i = act_j \wedge res_i \rightarrow res_j \wedge con_i \rightarrow con_j \tag{12}$$

These relevant rules must also be efficiently detected through the rule set.

2. **Inefficient comparison between relevant rules:** In the AC context, conditions are always homogeneously defined. For example, time is always expressed as a range. Therefore, no conversions are needed for conditions to be made comparable. In DUC, by contrast, the conditions that apply to the same domain may be heterogeneously expressed. Time conditions are a case in point. Time may be expressed as a range or as an event in time. As a result, direct comparisons cannot be made, and condition conversions are required. On this basis, we define Axiom 5.

Axiom 5. *Efficient comparison of DUC relevant rules. Based on the dependent relationship of conditions, reference formats must be established for each condition application domain so that heterogeneous conditions are converted once and then efficiently compared as many times as necessary.*

The following subsections present a novel tree structure based on Axiom 4 and Axiom 5 that combines a Resource Tree (RT) and a Policy Tree (PT). This enables relevant rules for common and dependent resources to be detected and compared efficiently.

5.1. Resource Tree

In the RT class diagram, shown in Figure 1, all the resources available in a use case are stored and structured according to their dependent relationships.

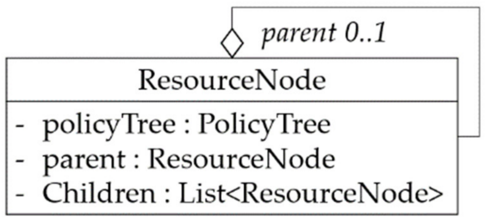


Figure 1. Resource Tree Class Diagram.

Specifically, each resource is represented via a RT node denoted as *resourceNode*. Each *resourceNode* contains its parent *resourceNode* and child *resourceNodes* satisfying the dependent relationship. Each *resourceNode* also contains a PT that stores and structures all the rules defined on it. Thus, relevant rules not only for common resources but also for dependent resources can be efficiently searched and compared.

5.2. Policy Tree

As shown in Figure 2, the PT of each *resourceNode* is composed of 5 types of nodes: the resource res_i, the action act_i, the subject sub_i, the condition application domain L_{adi}, and the conditions con_i.

The process of building the PT for each *resourceNode* is as follows. In the first instance all the rules defined within policies are sequentially analyzed in search of common resource conflicts. In this process, each rule is analyzed in relation to every relevant rule already stored in the PT of the corresponding *resourceNode*. Then the rule is stored in that PT. In this way, the following rules can be compared with that rule to check for common resource conflicts in further analysis and once all the rules have been analyzed, dependent resource conflicts can be checked for throughout the tree structure. To enable policy analysis to take place efficiently, the rules are structured in the PT following a set of guidelines.

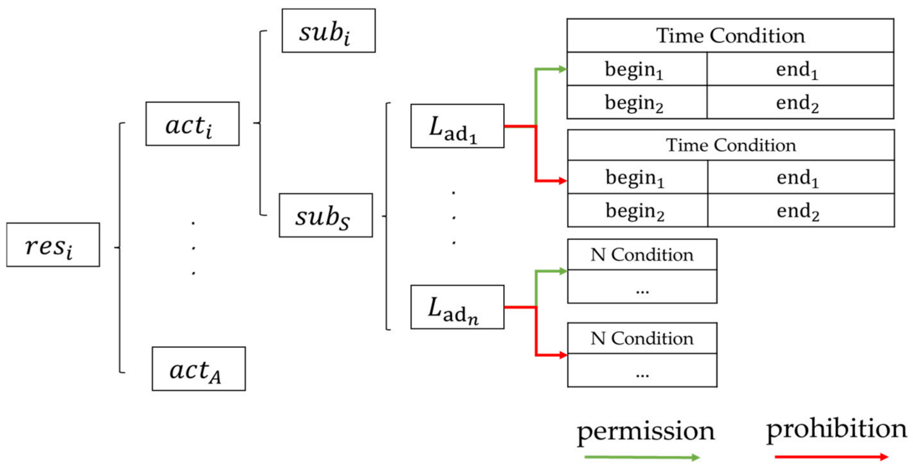


Figure 2. Tree Structure of Policy Tree.

First, to detect dependent resource conflicts, when common resource conflicts are detected and all the rules are stored in the PT of each *resourceNode*, searches are made from the root *resourceNode* through the RT branches sequentially comparing the resulting permissions or prohibitions defined for each subject sub_i on each action act_i with those defined for dependent resources. Therefore, to run this search efficiently and avoid redundancies, starting from the root node res_i, we built the PT by classifying the rules based on the actions act_i and subjects sub_i to which they refer. Given that there are typically fewer actions than subjects, and to reduce the number of branches and improve the search efficiency, the resource node res_i has one branch for each node related to the action act_i involved in data usage, but each action node act_i has one branch for each node related to the subject sub_i to which the rule refers.

Second, we classify the rules according to the dependent relationship of conditions in application domains L_{ad_i}. Specifically, in the PT each sub_i has a set of branches each of which contains a child node with one condition application domain L_{ad_i} where the rule applies. These condition application domains are identified from each $leftoperand_i$ that may describe a condition. Furthermore each L_{adi} node has two branches depending on the type of the rule defined. Finally, each of these branches has the list of conditions under which each type of rule applies for the corresponding application domain L_{adi}. The PT is represented by a hash table where the key represents the node itself and the value, the child node. Thus, for common resource conflicts it takes $O(1)$ to search for relevant rules on a specific application domain L_{adi}. For dependent resource conflicts, the analysis is performed individually comparing the resulting relevant permissions and prohibitions for two dependent resources for each subject sub_i, action act_i, and condition application domain L_{adi}. Therefore, taking at most $O\left(SxAxL_{ad}\right)$.

Finally, we store the lists of conditions for each application domain through reference formats. Thus, time-related conditions are stored as time intervals regardless of whether they are expressed in that term or, for example, as an event in time, while user-related conditions are stored as a list of users regardless of whether they are explicitly expressed as such or defined through membership. Consequently, every time a rule is analyzed, the condition that refines it is only converted once and many unnecessary conversions between stored heterogeneous conditions are avoided. This improves performance in comparing conditions.

6. Conflict Detection

In the following subsections we propose an assessment method for detecting conflicts between relevant rules in DUC for common and dependent resources in the form of inconsistencies and redundancies. The method is individualized depending on the approach used for rule definition (whitelisting or blacklisting).

6.1. Common Resource Conflicts

Below, we define how inconsistencies and redundancies appear between two relevant rules defined for a common resource depending on the approach used for rule definition.

6.1.1. Whitelisting

For two relevant rules r_i and r_j described through the whitelisting approach, we define that two non-overlapping conditions such that $con_i \cap con_j = \varnothing$ lead to an inconsistency when Formula (13) is satisfied, but if one condition is encompassed by another such that $con_i \cap con_j = con_i$ or $con_i \cap con_j = con_j$ a redundancy appears when Formula (14) is satisfied. Regarding the former, two non-overlapping conditions will never be satisfied at the same time. Therefore, a prohibition will always result. For the latter, one condition encompasses the other one, so the first one is unnecessary because it is less restrictive.

$$type_i = type_j = PERMISSION \wedge sub_i = sub_j \wedge act_i = act_j \wedge res_i = res_j \wedge con_i \cap con_j = \varnothing \quad (13)$$

$$type_i = type_j = PERMISSION \wedge sub_i = sub_j \wedge act_i = act_j \wedge res_i = res_j \wedge \left(con_i \cap con_j = con_i \vee con_i \cap con_j = con_j\right) \quad (14)$$

As shown in Figure 3, for the relevant rules r_1, r_2, r_3, and r_4 that satisfy the dependent relationship of conditions such that $leftoperand_{1,2,3,4} \in L_{time}$, con_2 refines con_1. However, while con_1 and con_3 lead to an inconsistency as $con_1 \cap con_3 = \varnothing$ because permission would never be granted, con_1 and con_4 lead to a redundancy because $con_1 \cap con_4 = con_4$ and permission would only be granted for con_4.

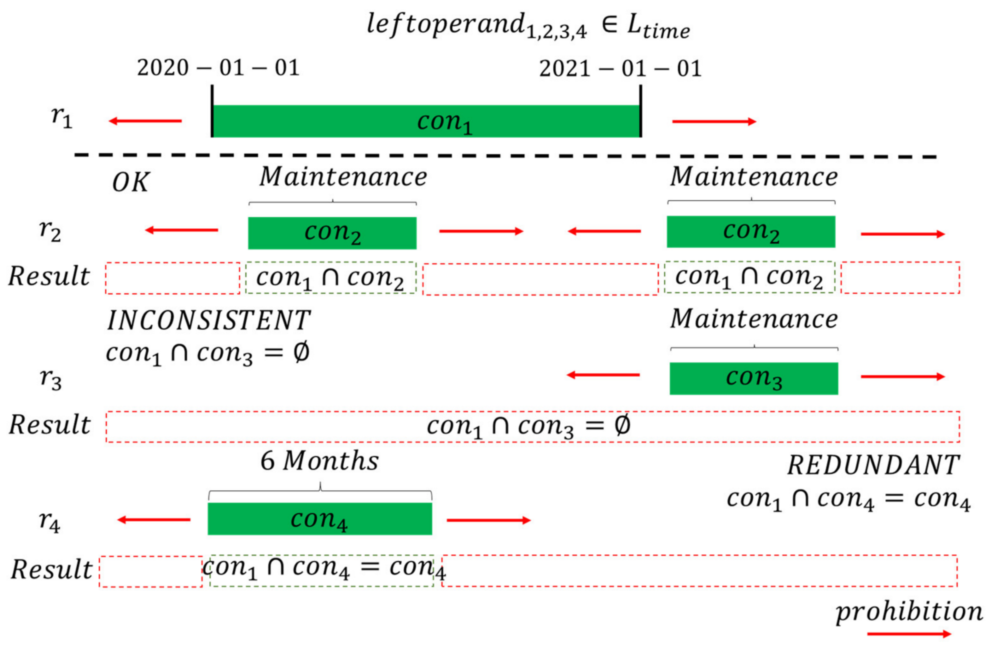


Figure 3. Common Resource Conflict Detection. Whitelisting Approach.

6.1.2. Blacklisting

In this case, for two relevant rules r_i and r_j described through the blacklisting approach, we say that a common resource conflict appears (always as a redundancy) if a condition such as con_i encompasses the other condition con_j, or vice versa so that Formula (15) is satisfied.

$$type_i = type_j = PROHIBITION \wedge sub_i = sub_j \wedge act_i = act_j \wedge res_i = res_j \wedge \left(con_i \cap con_j = con_i \vee con_i \cap con_j = con_j\right) \quad (15)$$

As shown in Figure 4, for the relevant rules r_1, r_2, and r$_3$, which satisfy the dependent relationship of conditions such that $leftoperand_{1,2,3} \in L_{con}$, while con$_1$ complements con_2, a conflict arises as a redundancy when a condition con_2 encompasses con_3. This is because con_3 is not necessary as it is already defined within con_2.

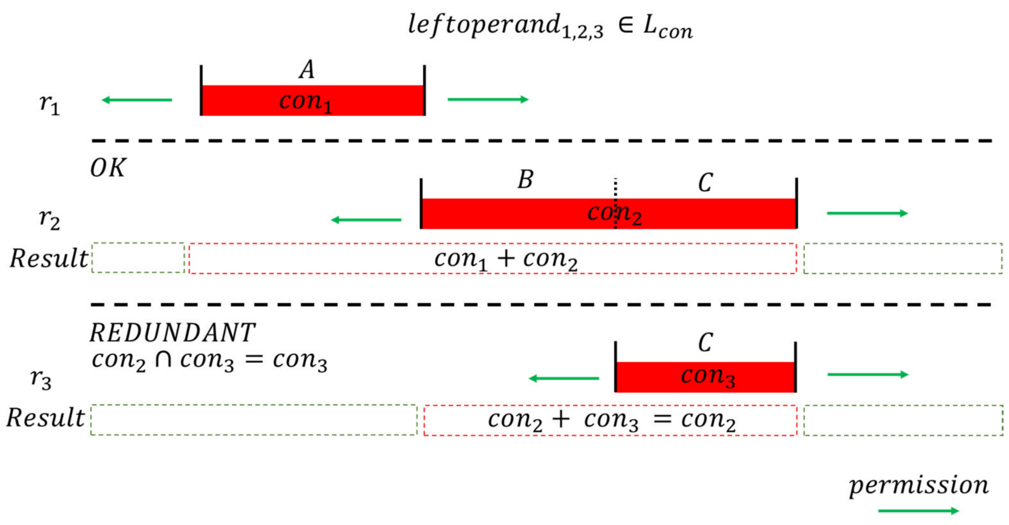


Figure 4. Common Resource Conflict Detection. Blacklisting Approach.

Within this approach, we consider that prohibition rules can be combined with permission rules so that exceptions are introduced to the permissions granted. Thus, relevant rules r_i and r_j described through the whitelisting and blacklisting approaches respectively do not lead to a conflict if the prohibition refines the permission to some extent so that $con_i \cap con_j \subset con_i$.

In other words, on the one hand we define that a redundancy appears that satisfies Formula (16) when the two rules do not overlap so that $con_i \cap con_j = \varnothing$. This is because the prohibition rule is redundant with respect to the prohibition already set by the permission rule on dependent but non-overlapping conditions. On the other hand, we define that an inconsistency appears that satisfies Formula (17) if the prohibition rule makes an exception to the entire permission so that $con_i \cap con_j = con_i$ and the permission becomes invalid.

$$type_i = PERMISSION \wedge type_j = PROHIBITION \wedge sub_i = sub_j \wedge act_i = act_j \wedge res_i = res_j \wedge con_i \cap con_j = \varnothing \quad (16)$$

$$type_i = PERMISSION \wedge type_j = PROHIBITION \wedge sub_i = sub_j \wedge act_i = act_j \wedge res_i = res_j \wedge con_i \cap con_j = con_i \quad (17)$$

As shown in Figure 5, for the relevant rules r_1, r_2, r_3, r_4 refined through dependent conditions, an exception is correctly made for r_1 when r_2 is defined. Whereas r_4 leads to a redundancy as the prohibition is already made by r_1, and r_3 results in an inconsistency because it makes the permission already defined invalid regardless of the condition.

6.2. Dependent Resource Conflicts

Dependent resource conflicts may arise between the resulting relevant permissions or prohibitions defined for a subject to perform an action on a resource for each condition application domain.

Deng et al. [28] define the transmission relationship of access authority, based on the idea that if a prohibition is defined on an upper resource, it is deduced that access for child resources is also denied. However, the oversimplification with which conditions are supported means that this axiom is not correctly translated. Based on the real inferences of refining rules through conditions, we fully translate the axiom proposed in what we call the transmission relationship of usage authority. Depending on how rules are defined, the transmission relationship of usage authority changes. However, the idea is the same: for a set of rules defined on a resource, the resulting permissions must be more restrictive through dependent resources.

The following subsections present the concept of the transmission relationship of usage authority for each approach used in rule definition. On that basis, we also define how inconsistencies appear depending on the conditions.

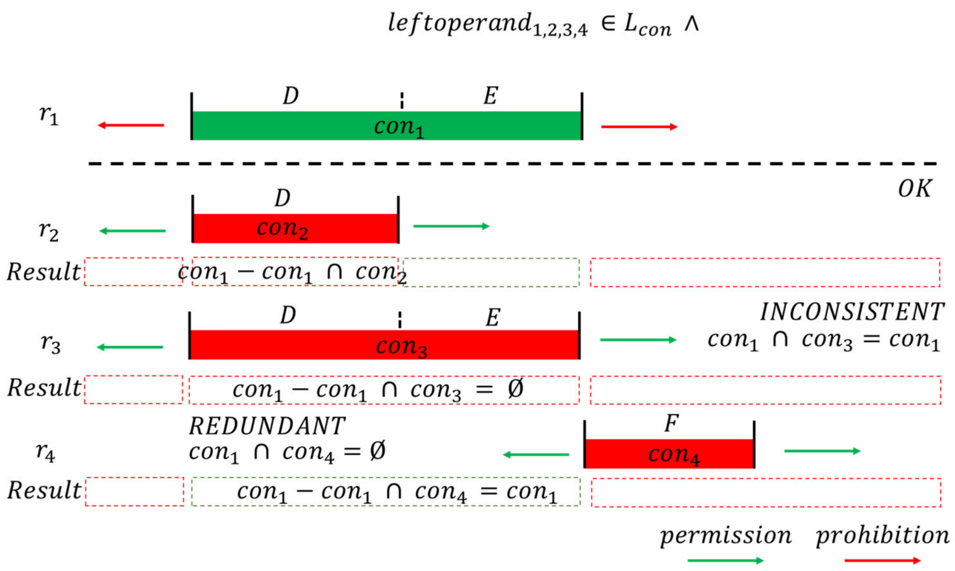

$leftoperand_{1,2,3,4} \in L_{con} \wedge$
r_1
D
E
con_1
OK
r_2
D
con_2
Result
$con_1 - con_1 \cap con_2$
r_3
D
E
con_3
INCONSISTENT
$con_1 \cap con_3 = con_1$
Result
$con_1 - con_1 \cap con_3 = \emptyset$
REDUNDANT
$con_1 \cap con_4 = \emptyset$
r_4
F
con_4
Result
$con_1 - con_1 \cap con_4 = con_1$
permission
prohibition

Figure 5. Common Resource Conflict Detection. Whitelisting + Blacklisting Approach.

6.2.1. Whitelisting

For the whitelisting approach, we define the transmission relationship of usage authority as per Figure 6.

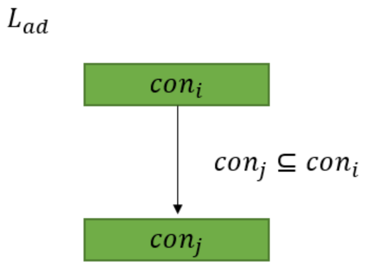


Figure 6. Transmission Relationship of Usage Authority. Whitelisting Approach.

Based on the resulting permissions from all the relevant rules defined for two dependent resources on a specific application domain L_{ad}, it can be said that if a permission is defined on an upper resource, the transmission relationship of usage authority is satisfied if a more restrictive permission is defined on child resources such that $con_j \subseteq con_i$. But if a less restrictive permission $con_j \supseteq con_i$ satisfying Formula (18) or a prohibition is defined, an inconsistency appears.

$$type_i = PERMISSION \wedge type_i = PERMISSION \wedge sub_i = sub_j \wedge act_i = act_j \wedge res_i = res_j \wedge con_j \supseteq con_i = 0 \quad (18)$$

6.2.2. Blacklisting

We define the transmission relationship of usage authority for the blacklisting approach as per Figure 7.

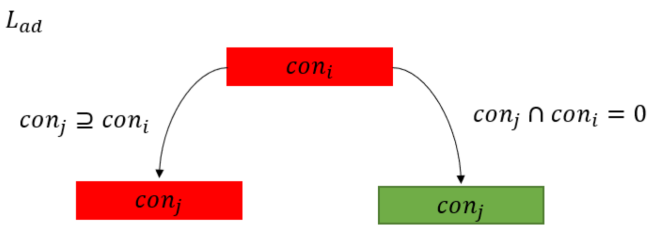


Figure 7. Transmission Relationship of Usage Authority. Blacklisting Approach.

In particular, also based on the results obtained for two dependent resources from all the relevant rules on an specific application domain L_{ad}, if a prohibition is defined on an upper resource, it can be said that the transmission relationship of usage authority is satisfied if either a more restrictive prohibition is defined on child resources such that at least the upper resource conditions are also included satisfying Formula (19), or a permission is explicitly defined on a condition where a prohibition has not been explicitly defined that satisfies Formula (20).

$$type_i = PROHIBITION \wedge type_i = PROHIBITION \wedge sub_i = sub_j \wedge act_i = act_j \wedge res_i = res_j \wedge con_j \supseteq con_i \quad (19)$$

$$type_i = PROHIBITION \wedge type_i = PERMISSION \wedge sub_i = sub_j \wedge act_i = act_j \wedge res_i = res_j \wedge con_j \cap con_i = 0 \quad (20)$$

7. Context-Aware Policy Analysis Algorithm

Figure 8 presents the flow diagram of the CAPA algorithm, which is based on the tree structures described in Section 5 and implements the method for detecting rule conflicts outlined in Section 6. The flow diagram describes how CAPA detects common and dependent resource conflicts between the policies defined in a use case.

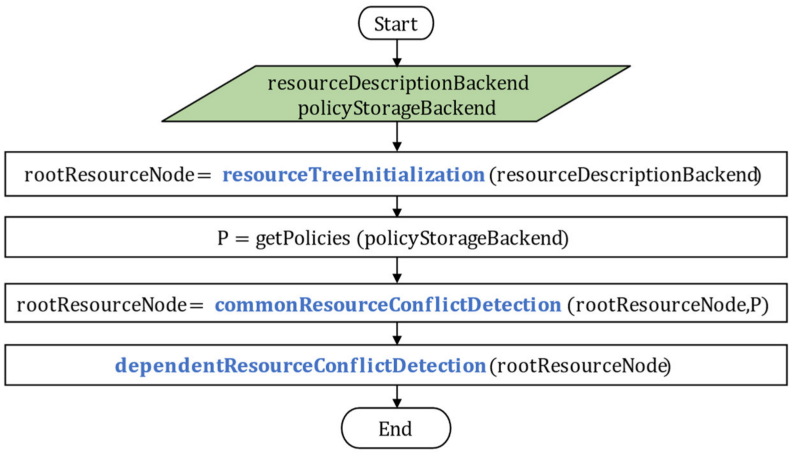

Start
resourceDescriptionBackend
policyStorageBackend
rootResourceNode= resourceTreeInitialization (resourceDescriptionBackend)
P = getPolicies (policyStorageBackend)
rootResourceNode= commonResourceConflictDetection (rootResourceNode,P)
dependentResourceConflictDetection (rootResourceNode)
End

Figure 8. CAPA Algorithm Flow Diagram.

Initially, the algorithm retrieves the information needed to connect to the domain-specific ontology where the resources of a given use case are described (*resourceDescriptionBackend*) and to the backend where the target policies to be analyzed are stored (*policyStorageBackend*). From the *resourceDescriptionBackend*, a first instance of the RT is built in the *rootResourceNode* through the *resourceTreeInitialization* algorithm by sequentially storing child resources as child *resourceNodes* based on their dependent relationships. Next, from the *policyStorageBackend*, the target policies P to be analyzed are retrieved. Then, from the *rootResourceNode*, through the *commonResourceConflictDetection* algorithm, each rule of P is sequentially checked for common resource conflicts. As a result, the *rootResourceNode* is retrieved with the rules of each policy stored in the PT of the corresponding *resourceNode*. Lastly, dependent resource conflicts are finally detected by the *dependentResourceConflictDetection* algorithm by running a backtracking process from the *rootResourceNode* along all its branches.

The *resourceTreeInitialization*, the *commonResourceConflictDetection* and the *dependentResourceConflictDetection* algorithms (marked in blue in Figure 8) illustrate the impact of our contributions on the policy analysis procedure, leading in addition to the optimized performance provided by CAPA. The following subsections describe these algorithms in depth. Furthermore, for each of them the time complexity is identified to highlight their performance.

7.1. Resource Tree Initialization Algorithm

With the *resourceDescriptionBackend* as input, Figure 9 represents the flow diagram of the algorithm that builds the RT in the *rootResourceNode* for a given use case.

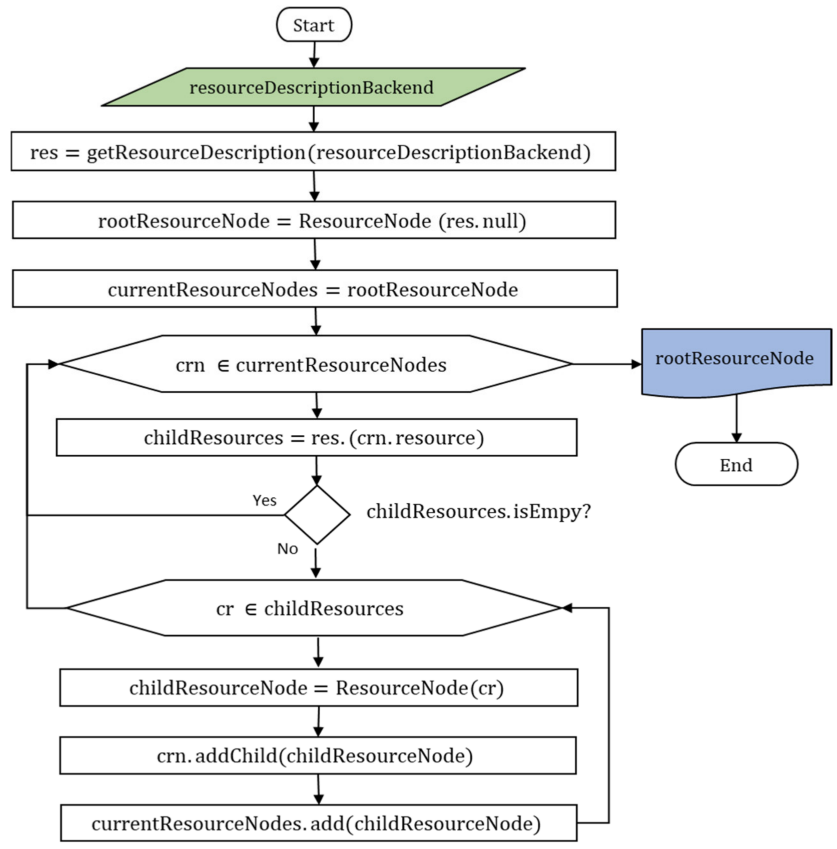

Start
resourceDescriptionBackend
res = getResourceDescription(resourceDescriptionBackend)
rootResourceNode = ResourceNode (res. null)
currentResourceNodes = rootResourceNode
crn ∈ currentResourceNodes
rootResourceNode
End
childResources = res. (crn. resource)
Yes
childResources. isEmpy?
No
cr ∈ childResources
childResourceNode = ResourceNode(cr)
crn. addChild(childResourceNode)
currentResourceNodes. add(childResourceNode)

Figure 9. Resource Tree Initialization Algorithm Flow Diagram.

Initially, the information stored in the domain-specific ontology is retrieved through the *getResourceDescription* algorithm and stored in a HashMap *res* containing the URI of the resource as the key and the list of URIs of the resources that satisfy the dependent relationship as the value. Then, the information about the root resource is stored in the *rootResourceNode* and the RT is built by sequentially analyzing in *res* the resources that satisfy the dependent relationship. Specifically, for each *resourceNode* denoted as *crn*, dependent resources are identified from *res* (*childResources*). If there are child resources (*cr*), a *resourceNode* is constructed (*childResourceNode*) for each one and they are joined to the RT through the parent *resourceNode* and stored to be analyzed in the following iteration.

As a result, for a set of resources *Res*, the *resourceTreeInitialization* algorithm takes $O(Res)$ to build the RT.

7.2. Common Resource Conflict Detection Algorithm

Figure 10 shows the flow diagram of the algorithm that detects common resource conflicts for a set of policies P. It is built on the RT previously stored in the *rootResourceNode*.

To that end, all the rules r defined on each of the policies $p \in P$ are sequentially analyzed. Specifically, for each r, the *resourceNode* from the RT that belongs to the resource defined in r is first identified through the *findResourceNode* algorithm. Second, the condition application domain L_{ad} is identified from the condition that refines r. Third, through the *conflictDetection* algorithm and from L_{ad}, conflicts are efficiently searched for and detected between r and the relevant rules already stored in the PT of the *resourceNode* found. The condition *cond* that refines r is also retrieved in its reference format. Finally, r is added with *cond* to the PT of the corresponding *resourceNode* through the *addRuleToPT* algorithm so that it can be efficiently compared in the analysis of the following rules. As a result, the RT is retrieved as an output with all the rules stored in the corresponding PTs of each *resourceNode*.

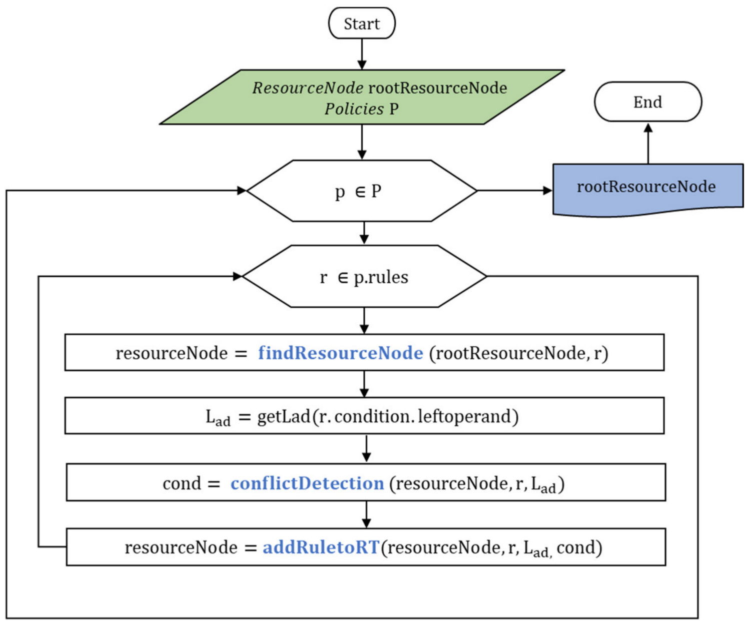

Start
ResourceNode rootResourceNode
Policies P
End
p ∈ P
rootResourceNode
r ∈ p.rules
resourceNode = **findResourceNode** (rootResourceNode, r)
L_{ad} = getLad(r. condition. leftoperand)
cond = **conflictDetection** (resourceNode, r, L_{ad})
resourceNode = **addRuletoRT**(resourceNode, r, L_{ad}, cond)

Figure 10. Common Resource Conflict Detection Algorithm Flow Diagram.

The *findResourceNode* and the *conflictDetection* algorithms become too complex, so they are further analyzed in the following subsections. In the opposite way, the *addRuleToPT* algorithm is limited to adding a rule to the HashMap that makes up the PT. This algorithm has a time complexity of $O(1)$.

7.2.1. FindResourceNode Algorithm

Taking the *rootResourceNode* and the target rule *r* as inputs, Figure 11 describes the flow diagram of the algorithm that retrieves from the RT the *resourceNode* belonging to the resource defined on *r*.

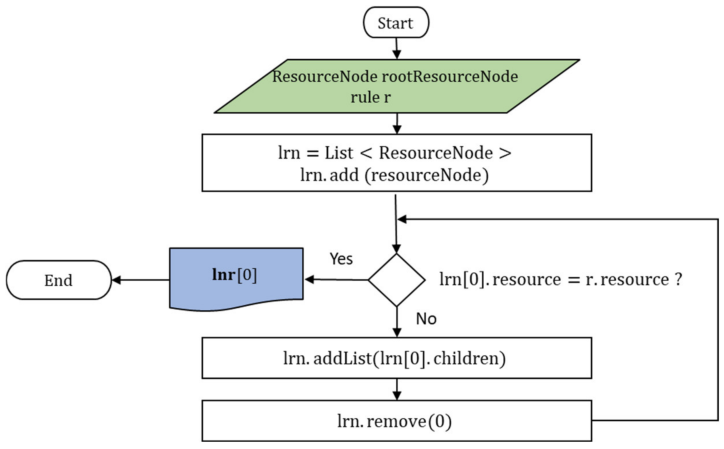

Start
ResourceNode rootResourceNode
rule r
lrn = List < ResourceNode >
lrn. add (resourceNode)
End
lnr[0]
Yes
lrn[0]. resource = r. resource ?
No
lrn. addList(lrn[0]. children)
lrn. remove(0)

Figure 11. FindResourceNode Algorithm Flow Diagram.

Starting with the *rootResourceNode*, the *resourceNodes* belonging to each level of the RT are sequentially analyzed. If the *resourceNode* belongs to the resource defined in *r*, the search is stopped and the *resourceNode* is returned. Otherwise, its child *resourceNodes* are stored for further analysis.

Therefore, for a set of resources *Res* it takes $O(1)$ to find the corresponding *resourceNode* in the best case if *r* is defined for the *rootResourceNode* and $O(Res)$ in the worst case if all the branches of the RT are analyzed until the *resourceNode* is found.

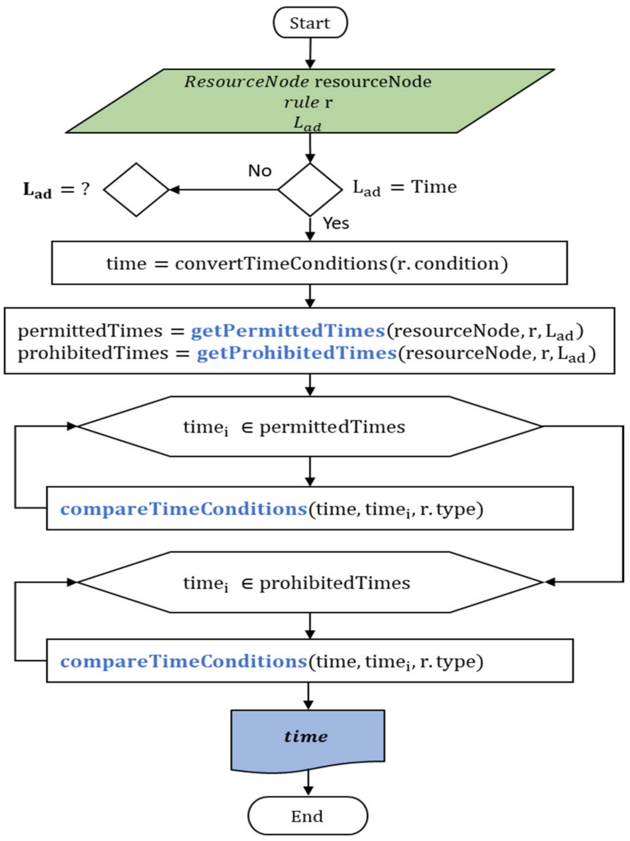
7.2.2. ConflictDetection Algorithm

Figure 12 shows the algorithm based on L_{ad} that efficiently retrieves and compares with r the relevant rules stored in the PT of the *resourceNode* previously found throughout the RT.

The flow diagram is represented for time-based conditions, but the logic is extended to other condition application domains. To detect conflicts, the condition that refines r is first converted to the reference format so that it can be efficiently compared with all the relevant time-based conditions stored in the PT of the *resourceNode*. Second, through the *getPermittedTimes* and *getProhibitedTimes* algorithms and based on L_{ad}, relevant time-based conditions related to r are respectively efficiently retrieved for permissions and prohibitions from the PT of the *resourceNode*. Thus, from them and through the *compareTimeConditions* algorithm that implements the evaluation formalized in Section 6.1, common resource conflicts are efficiently detected.

The time complexity of the *conflictDetection* algorithm can be analyzed in the following 2 complementary ways:

- Relevant conditions search: for a *resourceNode*, it takes $O\ (1)$ for *getPermittedTimes* and *getProhibitedTimes* algorithms to find either permission or prohibition relevant conditions through the HashMap that makes up the PT.
- Relevant conditions comparison: if done by brute force, comparing heterogeneous conditions requires one conversion for the condition that refines the target rule and another for each relevant condition stored in the PT. However, in the algorithm presented, condition comparisons through the *compareTimeCondition* algorithm for a set of R rules only require R conversions: one for each target rule.



Start
ResourceNode resourceNode
rule r
L_ad
No
L_ad = ?
L_ad = Time
Yes
time = convertTimeConditions(r.condition)
permittedTimes = getPermittedTimes(resourceNode, r, L_ad)
prohibitedTimes = getProhibitedTimes(resourceNode, r, L_ad)
time_i ∈ permittedTimes
compareTimeConditions(time, time_i, r.type)
time_i ∈ prohibitedTimes
compareTimeConditions(time, time_i, r.type)
time
End

Figure 12. ConflictDetection Algorithm Flow Diagram.

7.3. Dependent Resource Conflict Detection Algorithm

Figure 13 shows the backtracking process along the RT for dependent resource conflict detection via a flow diagram.

First, starting from the *rootResourceNode*, the algorithm checks whether each *resourceNode* denoted as *crn*, has child *resourceNodes*. If so, each child *resourceNode* denoted as *childrn* is stored for further backtracking and is then analyzed for conflict detection. In this analysis, a sequential search is performed over the actions of each subject that are controlled by at least one rule. For each one, if the *childrn* is also controlled for the same subject and action, conflicts are sequentially searched for and detected for each condition application domain L_{ad}. Figure 13 shows the flow diagram for time-based conditions, but the method is extensible to all other domains. This said, for each L_{ad}, through the *getTime* algorithm, the list of conditions defined for both the parent *resourceNode crn* and child *resourceNode childrn* are retrieved and the resulting *parentTime* and *childTime* permissions are respectively calculated. Thus, by executing the *compareDependentTimeConditions* algorithm that implements the assessment formalized in Section 6.2, any dependent resource conflicts are detected.

The time complexity of the dependentResourceConflictDetection algorithm can be analyzed in the following 3 complementary ways:

- Dependent Resource Conflicts Search: given a resource controlled for a set of subjects S on a set of actions A and refined through l_{ad} application domains, it takes at most $O\ (SxAxL_{Ad})$ to search for dependent resource conflicts for the *resourceNode*.
- Relevant conditions search: for a *resourceNode*, it takes $O\ (1)$ for the *getTime* algorithm to find the permitted and prohibited conditions through the HashMap that makes up the PT.
- Relevant conditions comparison: conditions are already stored in reference formats, so conversions in the *compareDependentTimeConditions* algorithm are not required.

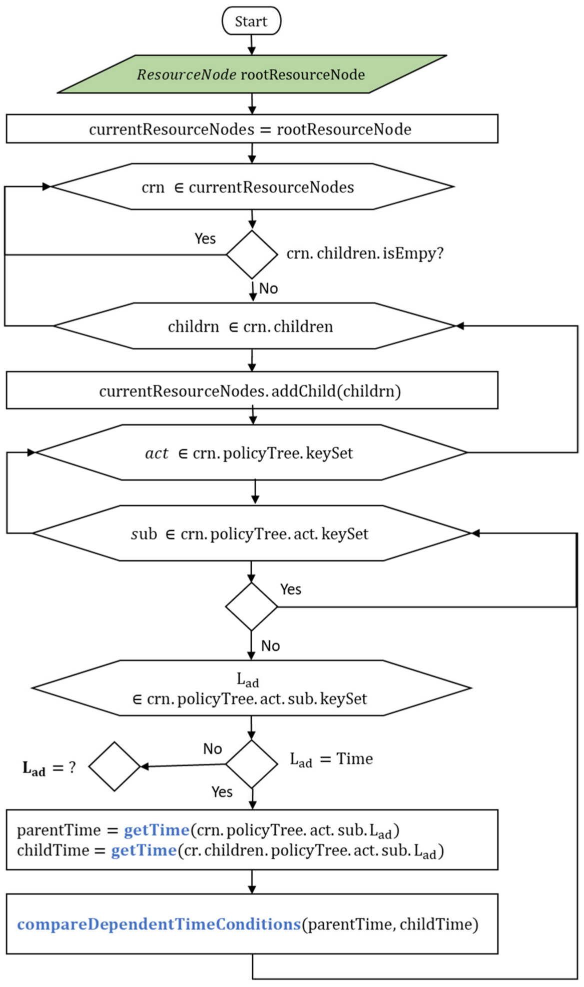

Start
ResourceNode rootResourceNode
currentResourceNodes = rootResourceNode
crn ∈ currentResourceNodes
Yes
crn. children. isEmpy?
No
childrn ∈ crn. children
currentResourceNodes. addChild(childrn)
act ∈ crn. policyTree. keySet
sub ∈ crn. policyTree. act. keySet
Yes
No
L_ad ∈ crn. policyTree. act. sub. keySet
L_ad = ?
No
L_ad = Time
Yes
parentTime = getTime(crn. policyTree. act. sub. L_ad)
childTime = getTime(cr. children. policyTree. act. sub. L_ad)
compareDependentTimeConditions(parentTime, childTime)

Figure 13. Dependent Resource Conflict Detection Algorithm Flow Diagram.

8. Experimental Assessment

The aim of the experimental assessment is to prove that CAPA provides security by detecting all inconsistencies and redundancies on all the policies defined by a data owner in a DUC solution that has been implemented in a wind energy use case. The performance provided by CAPA is also analyzed by measuring its efficiency. It is worth mentioning that this considerable efficiency increase is obtained by including support for the dependent relationship of conditions. To that end, the lack of context-aware policy analysis approaches leads to a Basic Policy Analysis, denoted as BPA, being implemented. BPA implements the algorithms described in Section 7 but omits the contributions presented in Section 5 related to the efficient search and comparison of DUC relevant rules. In this regard, conditions are not structured in the PT based on their dependent relationship and they are stored in the same format in which they are expressed.

On this basis, this section presents the most significant results obtained in the wind energy use case where CAPA and BPA are implemented and tested. To do so, the components of the use case that are most relevant in terms of implementation are identified and explained. Next, due to the lack of large-scale sets of rules related to DUC, specific rule sets are created for the purposes of assessment. The settings used in the environment where CAPA and BPA are deployed are also detailed. Finally, the results of the experimental assessment are analyzed.

8.1. Wind Energy Use Case

In wind energy, wind farm competitiveness is closely connected with the maintenance of wind turbines. Therefore, analyzing and learning from the data collected from the wind turbine operation is of the utmost importance [8]. However, due to current reluctance to share private data, the data collected from wind turbine operations is retained by wind farm operators and OEMs. As the data owners, they are the only actors who usually extract added value from the data [5]. Thus, other stakeholders in the value chain such as component suppliers and third-party service providers are missing the opportunity to analyze and learn from wind turbine operation data. This issue has a major impact not only on the competitiveness of each individual stakeholder, but also on that of the entire value chain. To increase competitiveness, reluctance to share data must be overcome by encouraging trusted data sharing. In this regard, there are already architecture models such as IDSA [19] which provide a reliable reference for driving the development of ecosystems that can boost data sharing by providing trust, data sovereignty, and interoperability.

Figure 14 presents a use case for wind energy, where the IDSA ecosystem is adopted to enable trusted sharing of gearbox-related data across the value chain. This use case features a company that deploys sensors which monitor the gearbox of a wind turbine and provide condition data from the lubrication system. From that data, edge computing services developed by a service company estimate gearbox health status. The gearbox is one of the components of a wind turbine that has the greatest impact on maintenance. Therefore, sharing such information with other stakeholders in the value chain such as the component supplier is of great interest because their combined data and expertise can significantly improve gearbox operation and maintenance (e.g., by improving component design, by establishing specific maintenance policies, etc.). To promote trustworthy sharing of this data by providing data sovereignty, a technology provider deploys two IDS DataSpace Connectors to share data between a data owner and user, an Identity Provider to provide trust between the IDS DataSpace Connectors, and a Vocabulary Provider for resource description. The Connectors and the Vocabulary Provider are directly involved in this DUC demonstration, so they are described in more detail below.

IDSA DataSpace Connectors (https://international-data-spaces-association.github.io/DataspaceConnector/ accessed on 26 September 2022) are the technical components responsible for correct data sharing between a data owner (e.g., the wind farm operator) and a data user (e.g., a component supplier) through the trusted IDSA ecosystem. Among other tasks, they are responsible for providing data sovereignty through DUC. On the side of the data provider, they are responsible for implementing the DUC policies defined by the data owner. On the side of the data consumer, they are responsible for enforcing DUC policies correctly. To provide semantic interoperability through these connectors, DUC policies are implemented following the IDSA UPL by integrating the IDSA IM through a Java Library. In the IDSA IM, usage requirements are defined through IDSA contracts (called simply contracts hereafter). A contract comprises two sections: The contract metadata and the IDSA Usage Control Policy. The metadata contains contract-specific information (e.g., the date of issue and the participants), while the IDSA Usage Control Policy, following the IDSA UPL, comprises IDSA Rules that describe the usage control statements that must be examined for policy quality. Therefore, policy analysis takes place within the IDSA DataSpace Connector in the role of the data provider through the implementation of CAPA or BPA.

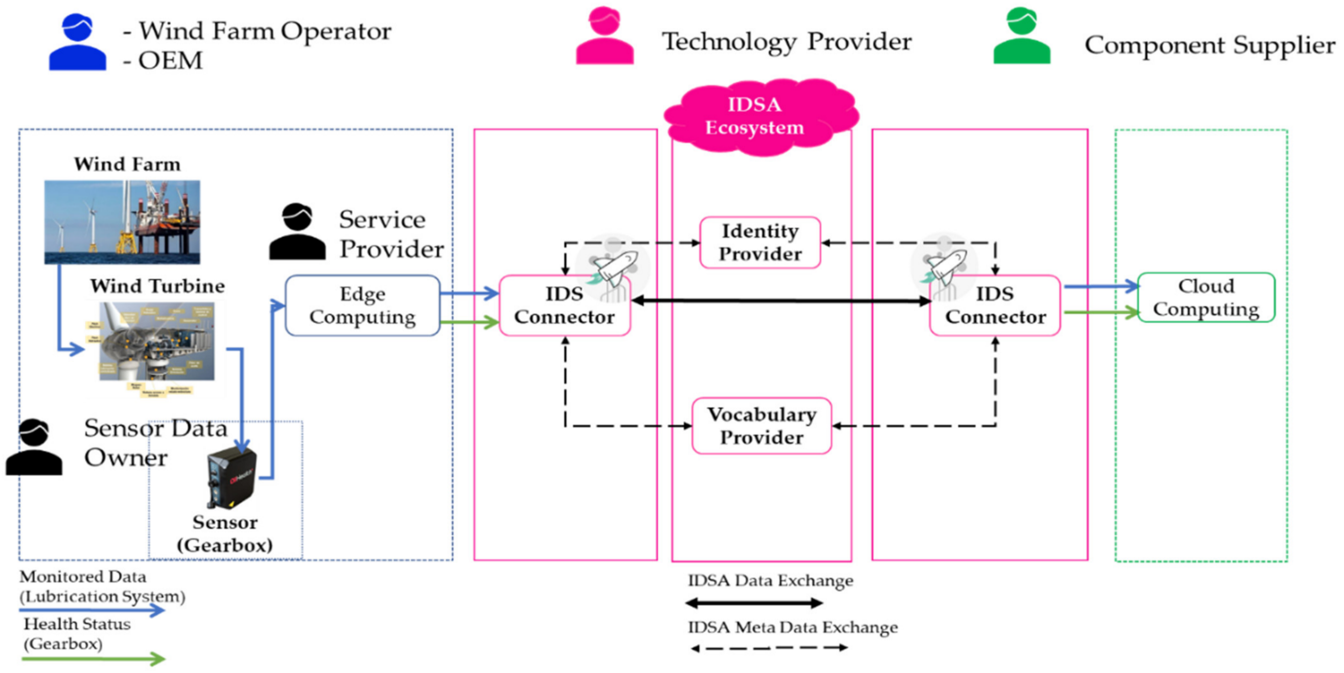


Figure 14. IDSA Wind Energy Use Case.

The **Vocabulary Provider** manages and provides a domain-specific Wind Farm Ontology (in this case WFOnt) (https://w3id.org/wfont) that describes the resources shared by the IDSA DataSpace Connectors through the IDSA ecosystem. As a result, not only common but also dependent resource conflicts can be detected through policy analysis by CAPA and BPA. WFOnt is inspired by the SANDIA Report [29] and reuses the AffectedBy and EEP (execution-executor-procedure) ontology design patterns to discover sensors or actuators that observe or act on a given quality or feature of interest.

8.2. Datasets and Setting

For the experimental assessment, we have developed a dataset generator that creates the rule sets summarized in Table 1.

These rule sets have been created following these guidelines:

- Each rule set contains 200, 2000, or 20,000 rules.
- For 50 resources available in a use case, which are classified into 5 levels of dependency, the rules of each rule set are equally divided into 10, 1000, or 10,000 rules defined for two dependent resources: the deepest one and its parent.
- For each of these two resources, the combined number of subjects and actions controlled varies exponentially from 1 to 10, 100, and 1000. In the rest of the analysis, this combination of subjects and their controlled actions is denoted as the product *AxS*.

- All subjects have the same number of rules defined for each action, 10 or 100, which are refined respectively by 2 or 20 Conditions for each of the following 5 application domains: Time, Connector, Location, Security Level, and Count.

Table 1. Experimental Assessment Rule Sets.

Rule Set	***R***	***Res***	***AxS***	***C***	L_{ad}
1	200	2	1 × 10	10	5: Time, Connector, Location, Security Level, and Count
2			10 × 1		
3			1 × 1	100	
4	2000		1 × 100	10	
5			10 × 10		
6			100 × 1		
7			1 × 10	100	
8			10 × A		
9	20,000		1 × 1000	10	
10			10 × 100		
11			100 × 10		
12			1000 × 1		
13			1 × 100	100	
14			10 × 10		
15			100 × 1		

CAPA and BPA are implemented in Java 17. All the experiments were performed on a Linux Server running Ubuntu 20.04 with 1 core and 2 GB RAM.

8.3. Results

In the following subsections, we first indicate the extent to which CAPA and BPA ensure security by detecting all inconsistencies and redundancies. Next, we focus on the measurement and analysis of CAPA's performance for this purpose compared to BPA. To provide a better insight, CAPA and BPA performances regarding common and dependent resource conflict detection are individually analyzed below.

8.3.1. CAPA and BPA Security

Conflicts were intentionally generated in the rule sets to assess the detection capabilities of the tree-based approach (included in both CAPA & BPA).

A 100% conflict detection is observed for CAPA and BPA. Therefore, we consider that the current policy analysis algorithm approach provides full security regarding policy consistency and minimality. In this way, this algorithm can boost data sharing in these ecosystems by providing data owners with trust and thus helping to overcome their reluctance to share their data.

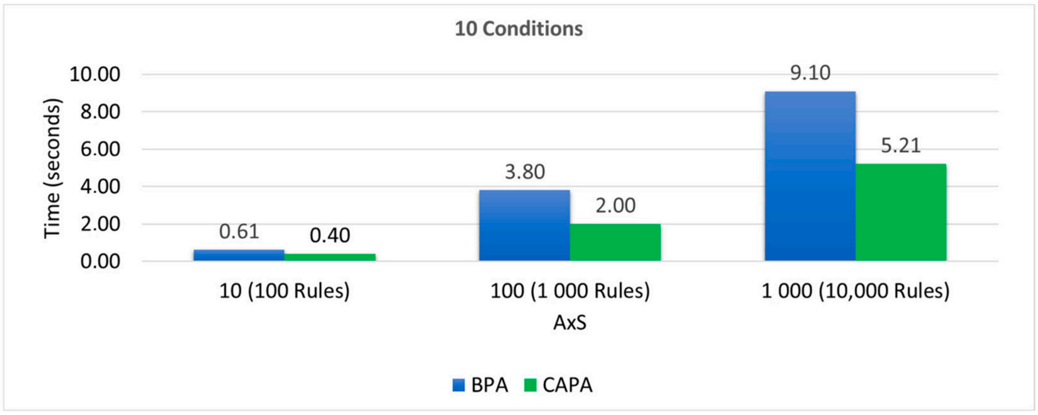
8.3.2. CAPA and BPA Performance on Common Resource Conflict Detection

For a more detailed, more comprehensive analysis, we split the analysis of the performance provided by both algorithms regarding common resource conflict detection into *findResourceNode*, *conflictDetection*, and *addRuletoRT* algorithms, as done in Section 7.2 in describing the conflict detection algorithm.

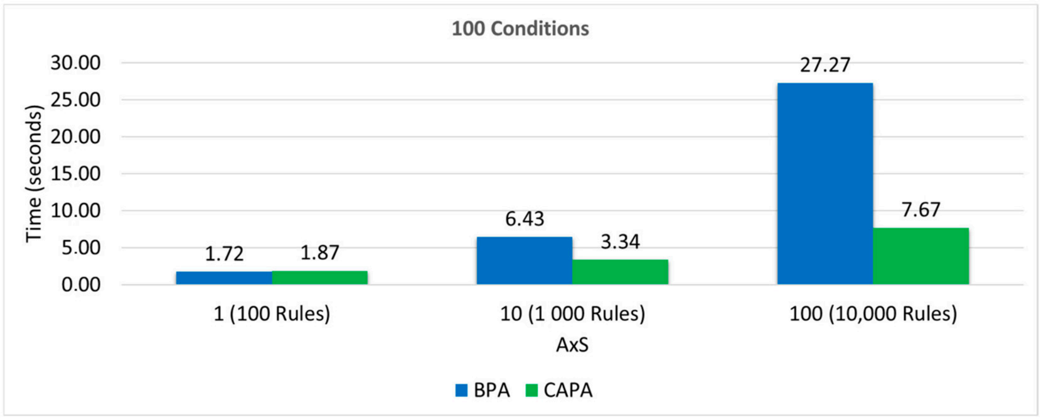
1. findResourceNode: Section 7.2.1 theoretically defines that the time complexity to find the resource defined in a rule within the RT is $O(1)$ in the best case if the rule is defined for the root *resourceNode* and $O(Res)$ in the worst case if all the branches of the RT are analyzed until the *resourceNode* is found. In the experiment performed, for the rules defined for the deepest resource the time required to find the *resourceNode* is not significant: it takes at most 1 ms for CAPA and BPA.

2. conflictDetection: from the *resourceNode* found by the *findResourceNode* algorithm, the time required to detect common resource conflicts for a rule depends on the time required to search and compare relevant conditions through the corresponding PT.

The experimental assessment proves that regardless of the number of subjects and actions, if the product *AxS* is the same (e.g., 1 × 100, 10 × 10, or 100 × 1) then the performance provided by the *conflictDetection* algorithm in CAPA and BPA depends only on the number of conditions (10 or 100). For a clearer analysis, Figure 15 represents the total time that the *conflictDetection* algorithm takes in CAPA and BPA to detect conflicts in all the rules defined for the deepest resource in each rule set based on *AxS*. In both Figures, the volume handled by the policy analysis algorithms increases from 10 to 100 and 1000 rules. In case of Figure 15a, *AxS* ranges from 10 to 100 and 1000 with 10 Conditions set in each controlled action for each subject. In Figure 15b *AxS* ranges from 1 to 10 and 100 *AxS* with 100 Conditions.



(**a**)



(**b**)

Figure 15. Total time required by the *conflictDetection* algorithm in CAPA and BPA: (**a**) 10 Conditions; (**b**) 100 Conditions.

Figure 15 shows that CAPA always provides a better performance. Conditions are demonstrated to be the component of the rule with a major impact on the performance provided by policy analysis algorithms. In this regard, CAPA also proves to be more adaptive to an increasing number of conditions.

On the one hand, for the same number of rules, for example, 1000, if conditions increase from 10 to 100 at the cost of reducing *AxS* from 100 to 10, both algorithms show a loss of performance, but CAPA drops by only 50% while BPA drops by 100%. In addition, for the same AxS, if conditions increase from 10 to 100, for 10*AxS* the *conflcitDetection* algorithm requires 6 s more in BPA, while in CAPA it requires just 3.

On the other hand, it is observed that these issues become even more evident as *AxS* increases. For 10*AxS* the *conflcitDetection* algorithm requires 6 and 3 more s in BPA and CAPA respectively when the conditions increase from 10 to 100, but for 100*AxS* CAPA requires 5 more s and in BPA the performance becomes clearly worse, requiring 24 more s. Furthermore, for 10,000 rules an increasing number of conditions at the cost of *AxS* also has a greater impact on performance with respect to 1000 rules. This is demonstrated by the fact that the performance of CAPA drops by just 2 s while BPA drops by 26 s.

3. addRuleToRT: when conflicts are detected, the time complexity of adding a rule to the PT of the *resourceNode* is theoretically defined as $O(1)$ in Section 7.2, but in practice it takes at most 1 ms for both policy analysis algorithms. Thus, it is considered non-significant.

In short, the results show that conditions are the component of the rule that has the greatest impact on the performance provided by CAPA and BPA in detecting common resource conflicts. This becomes even more critical as the number of conditions increases. In this regard, the results show that CAPA provides a better performance than BPA and is also more adaptable to an increasing number of conditions because of the dependent relationship of conditions that it supports.

8.3.3. CAPA and BPA Performance on Dependent Resource Conflict Detection

The experimental assessment only addresses the detection of dependent resource conflicts between the deepest resource within the RT and its parent. That is because this enables us to make a more accurate analysis of CAPA and BPA performance in this area, and the results can be extrapolated to the analysis of a larger number of dependent resources considering the time required for the backtracking process through the RT.

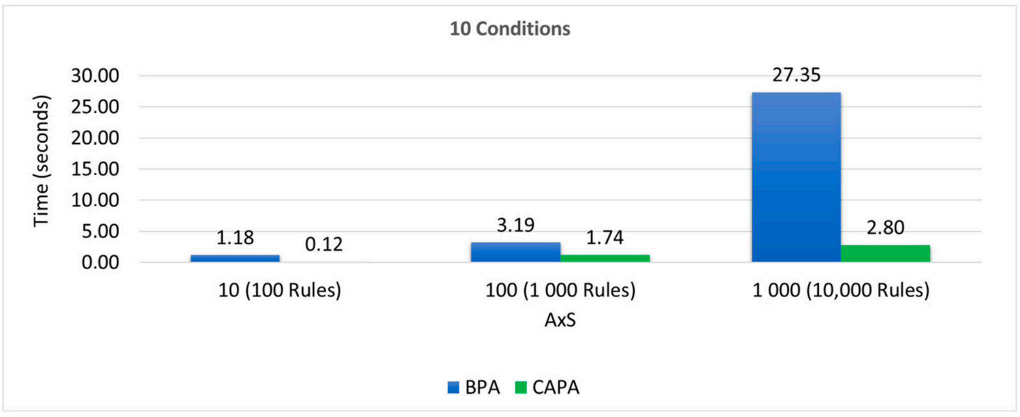
The *dependentResourceConflictDetection* algorithm performs a sequential search over the actions for each subject that are controlled for two dependent resources. So, if the product *AxS* is the same, for the same number of conditions then the performance provided for CAPA and BPA is observed to be the same. Therefore, to make a more comprehensive analysis, Figure 16 represents the total time that the *dependentResourceConflictDetection* algorithm requires to detect conflicts between the deepest resource and its parent based on *AxS* for 10 conditions and 100 Conditions.

For dependent resource conflict detection, CAPA also provides better performance and deals better with increasing numbers of conditions.

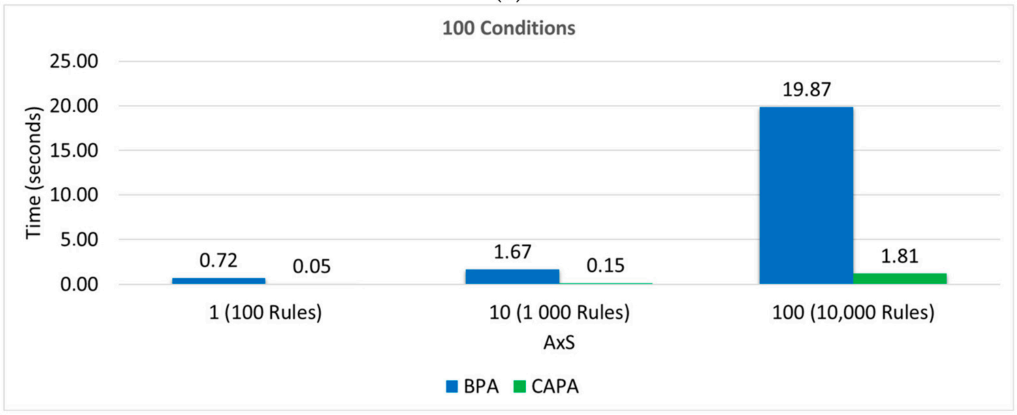
First, for the same number of rules, note that reducing *AxS* and increasing the number of conditions provides better performance. This is because of the time required for the sequential search for relevant rules for each subject in each action between dependent resources.

Second, for the same number of *AxS* (e.g., 10*AxS*), when the conditions increase from 10 to 100 CAPA shows little impact on performance, but BPA takes 500 ms longer. Furthermore, as the number of *AxS* increases, performance becomes even more critical. For 100*AxS* CAPA also shows the same performance for 10 and 100 Conditions, but in BPA there is a drop of 16 s.

Therefore, although the product of *AxS* seems to be critical for dependent resource conflict detection, the fact that CAPA supports the dependent relationship of conditions means that it not only always provides better performance but also reduces the impact of increasing numbers of conditions.



(**a**)



(**b**)

Figure 16. Total time required by the *dependentResourceConflictDetection* algorithm in CAPA and BPA: (**a**) 10 Conditions; (**b**) 100 Conditions.

9. Conclusions

Data sharing has become one of the main barriers to making significant progress in the energy sector. To solve this issue, data sovereignty must be ensured through DUC solutions. In this regard, it is essential to ensure security in DUC through policy quality.

On the understanding that context-aware control is one of the main new challenging features in the DUC context with respect to AC scenarios for providing data sovereignty in distributed ecosystems, this paper highlights its increased impact on policy quality and the challenges that it poses for efficient policy analysis. Not supporting conditions for policy analysis in DUC significantly increases the appearance of low-quality policies, leading to security breaches which may jeopardize the adoption of distributed collaborative ecosystems, but an efficient policy analysis which supports a wide set of heterogeneous conditions is challenging with respect to existing AC policy analysis approaches.

Accordingly, this paper presents CAPA, a tree-based modeling policy analysis algorithm for secure DUC approaches that ensures policy consistency and minimality for context-aware policies. To that end, CAPA is supported on previous AC policy analysis approaches: a consolidated area of research. Formalization and application to a wind energy use case shows that CAPA avoids security issues in DUC solutions, thus, facilitating data sharing throughout the value chain which have allowed and will allow researchers to advance in the knowledge of efficient wind energy generation. Furthermore, due to the complexity introduced by conditions and the lack of policy analysis algorithms that support them, it is not possible to compare performance with CAPA. However, by implementing a basic policy analysis algorithm (BPA), it is proven that CAPA significantly improves the performance provided in the policy analysis process by supporting the dependent relationship of conditions.

In conclusion, although data sharing presents great benefits, it depends largely on the implementation of secure, high-performance DUC solutions. In this regard, implementing good-quality policies is of the utmost importance. This paper addresses one of the main challenges in DUC: support for context-aware policies for policy consistency and minimality.

Future work will focus on extending CAPA to permit the real-time analysis of new rules by efficiently comparing them with relevant rules previously defined, analyzed, and thus stored in the tree structure. Completeness and relevance will be also addressed by complementing CAPA with transaction-based analysis is to provide full policy quality. Finally, the impact of action requirements on policy quality will be analyzed in depth, and an extension of CAPA will be studied to include adequate support for them. Additionally, based on the idea that ensuring security by detecting all the conflicts is crucial to promote data sharing, other approaches followed in the AC domain for policy analysis, different from tree-based modelling, will be explored. This analysis will have a particular focus on their ability to adapt to the complexity introduced by the DUC model, and, if any, comparing the performance provided with respect to CAPA. Moreover, unlike AC, policies in DUC are differently implemented by the data provider and consumer, negotiated and agreed to be afterwards enforced. In this regard, the applicability of CAPA in the policy negotiation process will be studied to ensure good quality agreed policies. Finally, future work will also address ongoing work related to the extraction of knowledge from the wind energy value chain, as the number of stakeholders and data grows, to make significant progress in its performance.

Author Contributions: Conceptualization, G.G., M.H. and A.A.; methodology, G.G., M.H. and A.A.; software, G.G.; validation, G.G.; formal analysis, G.G.; investigation, G.G.; writing—original draft presentation, G.G.; writing—review and editing, G.G., M.H., A.A., F.J.D. and E.J.; supervision, M.H., A.A. and F.J.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partly supported by the project HODEI-X (KK-2021/00049), funded by SPRI-Basque Government through the ELKARTEK program.

Data Availability Statement: Not applicable.

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

References

1. Scaria, E.; Berghmans, A.; Pont, M.; Arnaut, C.; Leconte, S. Study on Data Sharing between Companies in Europe: Final Report, Publications Office. 2018. Available online: https://data.europa.eu/doi/10.2759/354943 (accessed on 15 June 2022).
2. European Commission. A European Strategy for Data | Shaping Europe. Available online: https://digital-strategy.ec.europa.eu/en/policies/strategy-data (accessed on 15 June 2022).
3. Zhou, K.; Fu, C.; Yang, S. Big data driven smart energy management: From big data to big insights. *Renew. Sustain. Energy Rev.* **2016**, *56*, 215–225. [CrossRef]
4. Jarke, M.; Otto, B.; Ram, S. Data Sovereignty and Data Space Ecosystems. *Bus. Inf. Syst. Eng.* **2019**, *61*, 549–550. [CrossRef]
5. Kusiak, A. Renewables: Share data on wind energy. *Nature* **2016**, *529*, 19–21. [CrossRef] [PubMed]
6. Van Kuik, G.A.; Peinke, J.; Nijssen, R.; Lekou, D.; Mann, J.; Sørensen, J.N.; Ferreira, C.; van Wingerden, J.W.; Schlipf, D.; Gebraad, P.; et al. Long-term Research Challenges in Wind Energy—A Research Agenda by the European Academy of Wind Energy. *Wind Energy Sci.* **2016**, *1*, 1–39. [CrossRef]
7. Leahy, K.; Gallagher, C.; O'Donovan, P.; O'Sullivan, D.T.J. Issues with Data Quality for Wind Turbine Condition Monitoring and Reliability Analyses. *Energies* **2019**, *12*, 201. [CrossRef]
8. López De Calle, K.; Ferreiro, S.; Roldán-Paraponiaris, C.; Ulazia, A. A Context-Aware Oil Debris-Based Health Indicator for Wind Turbine Gearbox Condition Monitoring. *Energies* **2019**, *12*, 3373. [CrossRef]
9. Gil, G.; Arnaiz, A.; Higuero, M.; Diez, F.J. Assessment Framework for the Identification and Evaluation of Main Features for Distributed Usage Control Solutions. *ACM Trans. Priv. Secur.* **2022**. [CrossRef]
10. Pearson, S.; Casassa-Mont, M. Sticky Policies: An Approach for Managing Privacy across Multiple Parties. *Computer* **2011**, *44*, 60–68. [CrossRef]
11. Bertino, E.; Abu Jabal, A.; Calo, S.; Verma, D.; Williams, C. The Challenge of Access Control Policies Quality. *J. Data Inf. Qual.* **2018**, *10*, 1–6. [CrossRef]

12. Sandhu, R.; Park, J. Usage Control: A Vision for Next Generation Access Control. In *International Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 17–31. [CrossRef]
13. Aqib, M.; Shaikh, R.A. Analysis and Comparison of Access Control Policies Validation Mechanisms. *Int. J. Comput. Netw. Inf. Secur.* **2014**, *7*, 54–69. [CrossRef]
14. Abu Jabal, A.; Davari, M.; Bertino, E.; Makaya, C.; Calo, S.; Verma, D.; Russo, A.; Williams, C. Methods and Tools for Policy Analysis. *ACM Comput. Surv.* **2019**, *51*, 1–35. [CrossRef]
15. Lazouski, A.; Martinelli, F.; Mori, P. Usage control in computer security: A survey. *Comput. Sci. Rev.* **2010**, *4*, 81–99. [CrossRef]
16. Park, J.; Sandhu, R. The UCON $_{ABC}$ usage control model. *ACM Trans. Inf. Syst. Secur.* **2004**, *7*, 128–174. [CrossRef]
17. Kelbert, F.; Pretschner, A. Data Usage Control for Distributed Systems. *ACM Trans. Priv. Secur.* **2018**, *21*, 1–32. [CrossRef]
18. Hilty, M.; Pretschner, A.; Basin, D.; Schaefer, C.; Walter, T. A Policy Language for Distributed Usage Control. In *European Symposium on Research in Computer Security*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 531–546. [CrossRef]
19. Otto, B.; Steinbuss, S.; Teuscher, A.; Lohmann, S. IDSA Reference Architecture Model. Available online: https://internationaldataspaces.org//wp-content/uploads/IDS-Reference-Architecture-Model-3.0-2019.pdf (accessed on 15 June 2022).
20. Andreas, E.; Christian, J.; Robin, B.; Arghavan, H.; Sebastian, B.; Christian, K.; Pascal, B.; Gerd, B.; Mark, G.; Fabian, B.; et al. Usage Control in the International Data Spaces 3.0. Available online: https://internationaldataspaces.org/wp-content/uploads/dlm_uploads/IDSA-Position-Paper-Usage-Control-in-the-IDS-V3.pdf (accessed on 15 June 2022).
21. Bader, S.R.; Maleshkova, M. Towards Enforceable Usage Policies for Industry 4.0. In Proceedings of the International Workshop on Large Scale RDF Analytics (LASCAR) at Extended Semantic Web Conference (ESWC), Portorož, Slovenia, 3 June 2019.
22. Huang, C.; Sun, J.; Wang, X.; Si, Y. Inconsistency Management of Role Base Access Control Policy. In Proceedings of the 2009 International Conference on E-Business and Information System Security, Wuhan, China, 23–24 May 2009; pp. 1–5. [CrossRef]
23. Sandhu, R.; Coyne, E.; Feinstein, H.; Youman, C. Role-based access control models. *Computer* **1996**, *29*, 38–47. [CrossRef]
24. Sun, L.; Wang, H.; Tao, X.; Zhang, Y.; Yang, J. Privacy Preserving Access Control Policy and Algorithms for Conflicting Problems. In Proceedings of the 2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications, Changsha, China, 16–18 November 2011; pp. 250–257. [CrossRef]
25. Aqib, M.; Shaikh, R.A. An Algorithm to Detect Inconsistencies in Access Control Policies. In Proceedings of the Intl. Conf. on Advances in Computing, Communication and Information Technology (CCIT'14), London, UK, China, 1–2 June 2014; pp. 171–175. [CrossRef]
26. Organization for the Advancement of Structured Information Standards (OASIS). eXtensible Access Control Markup Language (XACML) Version 3.0. 2013. Available online: http://docs.oasisopen.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf (accessed on 15 June 2022).
27. Mohan, A.; Blough, D.M.; Kurc, T.; Post, A.; Saltz, J. Detection of Conflicts and Inconsistencies in Taxonomy-Based Authorization Policies. In Proceedings of the 2011 IEEE International Conference on Bioinformatics and Biomedicine, Atlanta, GA, USA, 12–15 November 2011; Volume 2011, pp. 590–594. [CrossRef]
28. Deng, F.; Zhang, L.-Y. Elimination of policy conflict to improve the PDP evaluation performance. *J. Netw. Comput. Appl.* **2017**, *80*, 45–57. [CrossRef]
29. Peters, V.A.; Hill, R.R.; Stinebaugh, J.A.; Veers, P.S. *Wind Turbine Reliability Database Update*; Sandia National Laboratories: Albuquerque, NM, USA; Livermore, CA, USA, 2009. [CrossRef]