Literature Study on enforcing software policies

Radu-Marian Doroș
Vrije Universiteit Amsterdam, The Netherlands
r.doros@student.vu.nl

ABSTRACT

Today, there is a growing and constant societal concern regarding digital privacy and security. This study delves into the challenges faced by software engineers in designing systems that comply with diverse laws and regulations. We focus on policy-enforcement systems, which abstract policy details to accommodate generic policies. Our investigation centers on the control models expressed by policies, the elements targeted by policy languages, and identifying the common design components of systems managing control mechanics. The insights from this study have broad implications for the design of software systems in various fields where regulatory compliance is crucial.

KEYWORDS

Literature Review **software policy enforcement, access control, RBAC, ABAC, usage control**

1 INTRODUCTION

1.1 Motivation

Software Engineers often face a multitude of challenges when designing and implementing systems. These tasks are becoming increasingly complex due to the addition of various laws and regulations. For instance, the General Data Protection Regulation (GDPR) has raised concerns among many small and medium-sized companies due to the complexities involved in its implementation [8]. While the benefits of such regulations and policies are undeniable, they often lack simplicity in implementation and introduce increased complexity. There is extensive literature on this topic, with Hjerppe et al. [9] providing a comprehensive overview. However, a detailed discussion on the intricacies of GDPR is beyond the scope of this work.

GDPR is just one example of the regulations that software systems must comply with. Depending on the field for which a software system is designed, other similar regulations may apply. Healthcare is one such field, known for handling a significant amount of sensitive, private patient information. Yet, in many cases, GDPR might be the only set of regulations that one needs to satisfy.

Given the diversity of laws and regulations that systems might need to comply with, the field policy-enforcing systems have emerged. Thes field focuses on designing systems that can accommodate generic policies. This constitutes also the primary focus of this study.

Moreover, this field is not solely motivated by the aim to simplify the task of engineers in complying with regulations. Various applications of enforcing policies have been suggested, such as Mahiru [20], where policies are a feature of the system and enable more trusted exchange between its collaborating users.

1.2 Research Questions

During this study, we focused primarily on gathering information related to or answering the following questions:

(1) **What are the control models expressed by the policies and their scope, and what elements of these models do policy languages target?** Policy enforcement systems, while aiming for more generic policies, must consider the expected control models during their design phase. The system needs to be capable of granting the policy access to the object in its model world. For instance, if the policies are expected to express predicates concerning the system-time of actions, the underlying system needs to accommodate this requirement.
Simultaneously, it's crucial to understand which types of control specifications policy languages can handle and under which conditions. This combined understanding of control models and policy languages will guide the design and implementation of a robust and effective policy enforcement system.
We will therefore explore the scope and limitations of control models expressed by policies. We explore the control models of Role-Based Access Control (RBAC), Attribute-Based Access Control (ABAC), and Usage Control (UCON).

(2) **What are the common design components of systems that handle control mechanics?** To support different control mechanics, what kind of components are typically observed in these systems?
While it is relatively simpler to design a system running a central policy, how are systems able to handle multiple policies?
For this, we look at finding which architectural components systems need to extend with to support the desired control models.

2 MODELS OF CONTROL

2.1 Access Control

Access control constitutes one of the most studied category of control models. Literature in the field is quite vast, we limit the scope of the section to extracting the general topics in the most popular instances of access control.

2.1.1 *Role-Based Access Control (RBAC).*

RBAC [15] methods define a simplified set of rules, that enjoy widespread use in settings where role-based semantics coincides with a convetional organizational structure.

RBAC works on the basis of *users, roles, permissions*. Users are assigned roles and roles are assigned permissions. Queries of the form "user x wants to perform action a, are they authorized?" are resolved by finding the permission set of the user (generally first finding the roles of the user and then in turn finding the permission set of the role) and then verifying if the action *a* is in the permitted operation set.

This works for many of the cases, since usually such access control systems suffice. The system then models the scope of these permissions.

To illustrate, consider a system that maintains a database of users and their associated roles. Upon a user logging in, the system queries the user's role, retrieves the permissions assigned to that role, and then determines the operations the user is permitted to perform within the system. For instance, users with basic roles may be prohibited from performing delete operations in a database.

However, RBAC is not without its limitations. One notable weakness is its difficulty in adapting to more flexible demands, e.g. in handling mechanisms that expire, such as temporary permissions or time-limited roles.

2.1.2 *Attribute-Based Access Control (ABAC).*

A survey of open problems and questions related to *ABAC* is presented in [17]. We summarize ABAC based on their section describing ABAC background.

ABAC was introduced to address the greatest considered problem of RBAC, flexibility. Systems adopting ABAC models define more flexible access control rules based on:

- Attributes of the subject (user). May include arbitrary information that is relevant for the system, including job title, age etc.
- Attributes of the object (resources of the system). May include metadata information about the resources, date of creation, size, owner, security level or content-derived information (was some policy-preserving pre-processing applied).
- Attributes of the Environment like time of the day, number of users in the system, a workflow's properties.
- Connection attributes, information about the session of the user, location (for mobile systems) etc.
- Administrative attributes. Examples of these include like threat levels at which policies some policies start playing a role, minimum trust levels (how much trust a user has to have to be able to access resources), maximum session length and so on.

Access policies are then defined using policy languages (XACML [6]) that define access decisions based on Boolean expressions over the set of available attributes. This model allows the systems to design the set of relevant attributed necessary for the creators of policies and leaves the implementors of the policies the duty to define the decision expressions that grant or deny access.

Hu et al. [10] continues and provides an extensive guidelines for using and applying ABAC policies.

2.2 Usage Control

Previous subsections dealt with access control, there is however a need to specify rules about what parties are allowed to do once the access is already provided. The field of usage control explores such issues.

UCON$_{ABC}$ [14]: Introduced usage control. They define the 3 dimensions of usage control as *(ABC)*:

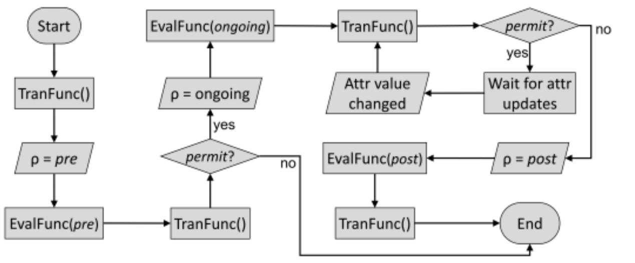

Figure 1: UCON Authorization Evaluation Flow (taken from [7]) - This shows the policy enforcement system tasks and has implications on what a system looks like.

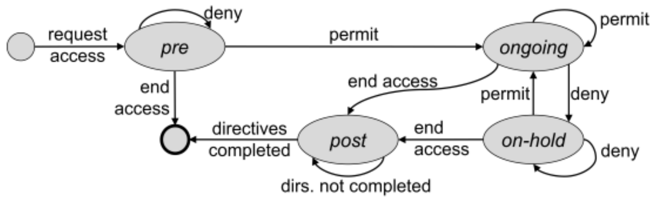

Figure 2: Extended UCON Authorization States (taken from [7]) - This shows the interaction from a clients perpective with the UCON model

- **A**uthorization - Access
- O**B**ligations - A specified set of actions subjects must perform before getting authorization (e.g. accepting terms of conditions)
- **C**onditions - Context/Environment conditions that can affect policy decisions

Following, they bring two configuration possibilities for each of the dimensions of the *UCON$_{ABC}$* model. Each dimension can be either a precheck, a continuous check or sometimes a post-check.

UCON Flow: In the UCON (Usage Control) framework, the process of managing policies and attribute updates is crucial for ensuring consistent and secure system behavior. The fig. 1 illustrates the high level flow of a UCON authorization system. Zooming in into these aspects:

In fig. 1 it is shown that there are three steps checked by policy reasoners. We can see that at the high level UCON running systems must manage 3 classes of defined policies - *pre*, *ongoing* and *post*. This constitutes the set of policies run. Another important mechanism needed to be covered by UCON systems is the interaction with the mutable attribute values. The application of the changes need to be consistent and synchronized with future evaluations of the attributes in a well-defined way. The exact modelling of this consistent updates of attributes is further explored in [16]. These aspects of update mechanisms are simplified in fig. 1 with the *TranFunc()* abstraction.

Policy Reasoners: In the UCON framework, policy reasoners are responsible for evaluating and enforcing access control policies. These policies are categorized into three classes:

(1) Pre-policies (*pre*): These policies are checked before a user's request is executed. They typically evaluate conditions such as authentication, authorization, and other prerequisites for access.
(2) Ongoing-policies (*ongoing*): These policies are continuously monitored during the execution of a user's request. They ensure that access remains valid and compliant throughout the interaction.
(3) Post-policies (*post*): These policies are evaluated after the execution of a user's request. They can perform actions such as auditing, logging, and updating the system state based on the outcomes of the request.

By managing these three classes of policies, UCON systems can provide fine-grained control over access to resources and enforce dynamic security requirements.

2.3 Languages

2.3.1 XACML [6]. The eXtensible Access Control Markup Language (XACML) is a widely used XML-based language in the realm of access control and policy-related research. Developed by the Organization for the Advancement of Structured Information Standards (OASIS), XACML provides a standardized framework for defining and enforcing access control policies. Its flexibility and extensibility have made it a popular choice in many scientific articles and real-world applications related to access control and policy management.

XACML is designed to express policies, rules, and access control decisions in a structured, comprehensive, and extensible manner. It supports ABAC by allowing policies to be defined based on the attributes of the user, the resource to be accessed, the action to be performed, and the context of the request.

In ABAC, access decisions are made by evaluating a set of attributes against policies. XACML's rich policy language and its ability to express complex, multi-dimensional access control policies make it well-suited for ABAC. It can handle a wide range of attributes and can express policies that take into account various combinations of these attributes.

Furthermore, XACML includes a request/response model for access control decision making and a policy language for expressing access control policies, both of which align well with the requirements of ABAC.

Many systems performing enforcement are based on XACML and include additional components as described by XACML, some of which are:

- Policy Decision Point (PDP) - This component evaluates the policy and makes a decision on whether to grant access to a particular resource based on the request.
- Policy Enforcement Point (PEP) - This component ensures that the requested resources are adequately protected and identifies which requests need further policy checks.
- Policy Information Point (PIP) - This component provides information about where to retrieve attribute values.
- Policy Execution Point (PXP) - This component is responsible for executing the decisions made by the PDP, enforcing the rules defined in the policy.
- Policy Administration Point (PAP) - The PAP allows for the administration of the policies, such as making changes or updating their configuration.
- Policy Retrieval Point (PRP) - This component is responsible for retrieving the relevant policies for the PDP to evaluate.
- Policy Management Point (PMP) - This component oversees the overall management of the policies, including their creation, deletion, and modification.
- Policy Translation Point (PTP) - This component is responsible for translating policies into a format that can be understood and enforced by the PEP.
- Event Processing Point (EPP) - This component processes events related to the enforcement of policies, such as the submission of a request for action, the completion of an action, the detection of a policy violation, or changes in the system policy state.

However, it can be noticed in the literature the terms are loosely used and maybe inconsistent. The definitions for PXP, PRP, PMP, and PTP are based on general understanding of these terms, as their specific definitions can vary depending on the context and the specific implementation of the XACML framework.

Take for example the Policy Execution Point (PXP) architectural component, which is responsible for executing the decisions made by the Policy Decision Point (PDP), enforcing the rules defined in the policy. This could involve actions like granting or denying access to a resource, logging the decision, or triggering other actions in the system.

Depending on the application context, the PXP responsibilities will vary greatly. For example, let's look at what the component would do in different contexts:

- In a network security system, the PXP might be responsible for controlling access to network resources based on the decisions made by the PDP. This could involve actions like opening or closing network ports, rerouting traffic, or blocking specific IP addresses.
- In a data management system, the PXP might be responsible for controlling access to data based on the decisions made by the PDP. This could involve actions like granting or denying read or write access to specific data records, encrypting or decrypting data, or triggering data backup or replication processes.
- In a cloud computing system, the PXP might be responsible for controlling access to cloud resources based on the decisions made by the PDP. This could involve actions like starting or stopping virtual machines, allocating or deallocating resources, or managing access to cloud storage or databases.

Policy Machine . Ferraiolo et al. [4, 5] describe the architecture of the Policy Machine show how they differentiate from $UCON_{ABC}$. Their focus is touching more upon *authorizations* and *obligations* and leave out aspects of *conditions*. That is, they avoid covering environmental (system) requirements in their design space. The focus seems to be more on the relationships and attributes of policy elements, and how these can be used to define and enforce a wide

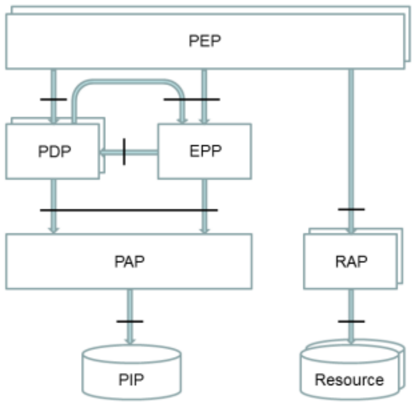

Figure 3: Policy Machine Architecture (taken from [5])

range of security policies. This makes it a very comprehensive specification document designing access control policy enforcement architecture systems. The fig. 3 illustrates the proposed architecture, showing the usage of most of the aforementioned XACML architectural components.

2.3.2 eFLINT [19]. The eFLINT language, as described in [19], is built upon the Hohfeld legal framework, which focuses on two fundamental relations between individuals in normative matters:

(1) *'Duty-Claim'* Relation - This relation arises when one individual has a duty or obligation towards another individual. It signifies that the first individual is obligated to perform or refrain from certain actions, and the second individual has the corresponding claim to expect the fulfillment of that duty.
(2) *'Power-Liability'* Relation - This relation exists when one individual possesses the power or authority to perform a particular action, and another individual is liable or subject to the effects or consequences of that action. It denotes the ability of one individual to exert control or influence over another individual's rights or interests.

By incorporating the Hohfeld legal framework, eFLINT provides a foundation for expressing and reasoning about these core relations within normative systems. This framework enables the specification and analysis of duties, claims, powers, and liabilities, facilitating the modeling and enforcement of complex normative policies in various domains.

eFLINT employs a general mechanics where a configuration is dynamically updated by *events* and *actions*. The system's rules of duties are designed to detect violations of specified norms. Events trigger changes in the configuration, while actions represent individual behaviors. *Duties* define expected behaviors and obligations, and violations are monitored by the system.

Normative actors - fig. 4. The *Normative actors* section details the building of a system with *normative actors* that answer *eFLINT* queries. The *normative actors* update their state based on these queries which can change the state (actors can issue *actions* that might trigger *duties* or cause *violations*). *Normative actors* will then notify actors of any such occurrences.

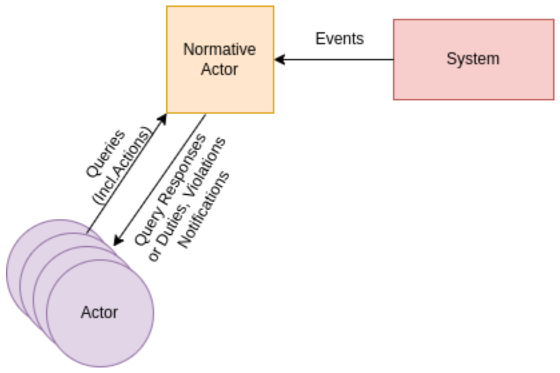

Figure 4: eFLINT Normative Actors

The paper claims eFLINT is able to be used in systems employing policy enforcement mechanisms and can be used to fulfill four types of enforcement:

(1) *Ex-ante enforcement of permissions:* This involves the prevention of actions that are not permitted, thereby ensuring that all activities are within the boundaries of established rules and regulations.
(2) *Ex-ante enforcement of positive duties:* - informing actors of their duties to perform certain actions
(3) *Ex-post enforcement of violations of prohibitions* - giving reasons for why actions were not allowed
(4) *Ex-post enforcement of violated duties* - giving reasons for why duties were violated

Some variations are also mentioned to support different kinds of applications:

(1) *Monitoring actors - fig. 5 fig. 6* - a set of actors that monitor the behavior of actors and report actions based on these observations to the *Normative actors.*
(2) *Internal Normative actors* in *multi-agent systems* - actors (encompassing agents) can use internally managed *normative actors* to maintain internal beliefs and states.

eFLINT case study GPDR. The case study examined a practical application of eFLINT, in a banking environment with respect to the General Data Protection Regulation (GDPR). The study focused on the GDPR requirement of obtaining a data subject's consent prior to processing their data.

The example showed that the use of eFLINT allowed for the translation of this normative act - obtaining consent - into enforceable rules based on specific actions and duties. The use of eFLINT also facilitated enhanced clarity and facilitated the effective implementation of enforcement mechanisms.

Related, [18] describe a way to extend eFLINT to have cooperative policy checkers. Authors extended *eFLINT* with two new constructs for the purpose of formalizing dependencies between social policies and the connections between social policies and system-level policies. For this they introduced the *Extends* and

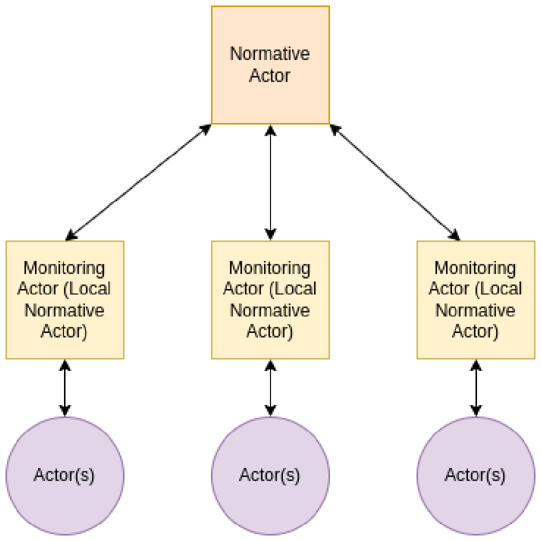
Normative Actor
Monitoring Actor (Local Normative Actor)
Monitoring Actor (Local Normative Actor)
Monitoring Actor (Local Normative Actor)
Actor(s)
Actor(s)
Actor(s)

Figure 5: eFLINT "sketched" Monitoring Actors

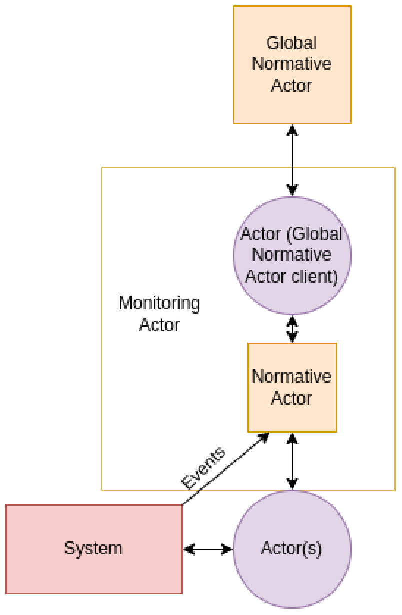

Figure 6: eFLINT Monitoring Actors Zoom-In: Possibly a composition of a client to a "higher" normative actor and a normative actor itself, Interacting with its subActors clients and receiving events from the System based on their interaction.

Syncs with keywords. These enable more possibilities for the norms represented by eFLINT state to be changed.

They however, never fully make a "local" policy evaluation runtime. They only use their already mentioned client-server implementation, where the server is a normative actor and clients are actors. Clients then collaborate into the same server and create the Facts and Actions they want. The other actors are themselves notified whenever violations or duties are raised.

Parizi et al. [13] design normative multi-agent systems, by introducing normative agent components. Introduces BDI enhanced with normative reasoning. They refer to the *belief-desire-intention* (BDI) architecture of Jason to contrast their addition with. Their enhancement is two-fold:

- A replacement of the belief-base with normative reasoner. Beliefs are now held in the form of norms. Instead of only belief-updates, agents handle duty-events, act-events, violation events.
- An enhancement over the set of actions that can be performed compared to fact-updates. System enables more general act-performing actions.

The general layout of the components and communication is shown in fig. 7. Agents interact with their own normative advisors and take actions based on these interactions with the environment (and other agents). Agents get informed when normative events happen (duty events raised, norm violation etc.) and furthermore, might choose when to follow, extend or violate rules themselves.

Furthermore, the authors connect their MAS mechanisms to a way of implementing stricter "normative protocols" for agents collaborating and conforming to enforcing norms. Compared to the general layout in fig. 7, authors introduce *enforcer* agents, that watch over norms and can provide incentives to comply.

3 CONCLUSION

RQ1 What are the control models the policies express? What is their scope? We looked at multiple control models. In Kayes et al. [12] we find a taxonomy of the Access Authorization Models as shown in fig. 10. The relationship between Access Control and Usage control is illustrated in fig. 9. We enumerate the control models:

- Discretionary Access Control (DAC): In this model, the owner of the resource decides who can access it and what they can do with it. The scope of DAC is typically limited to individual users and resources.
- Mandatory Access Control (MAC): This model is often used in environments that require a high level of security. Access decisions are made based on the classification of information and the security clearance of users. The scope of MAC is typically an entire system or organization.
- Role-Based Access Control (RBAC): In RBAC, access decisions are based on the roles that users have within the system. This model is often used in large organizations where roles are defined according to job competency, authority, and responsibility.
- Attribute-Based Access Control (ABAC): ABAC is a more flexible and comprehensive model. Access decisions are made based on attributes of the user, the resource, the action, and the environment. ABAC can express a wide range of policies, making it suitable for complex, dynamic environments.
- Usage Control (UCON): UCON is a comprehensive model that considers not only the access but also the usage of resources. It can express policies that continue to apply even

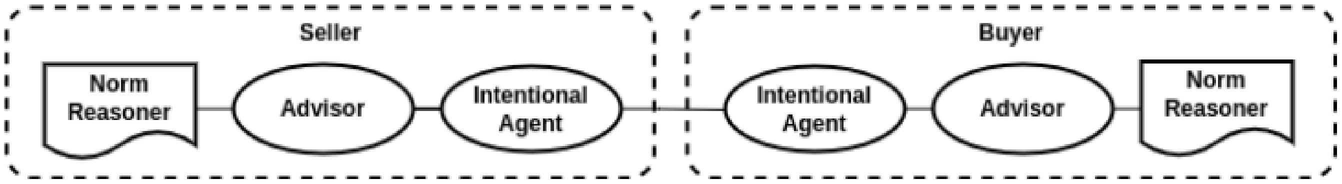

Figure 7: eFLINT handling beliefs in MAS for a digital marketplace (taken from [13])

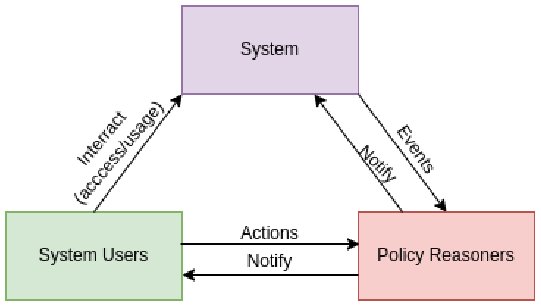

Figure 8: General Architecture - Policy Reasoner as a state updating Event loop

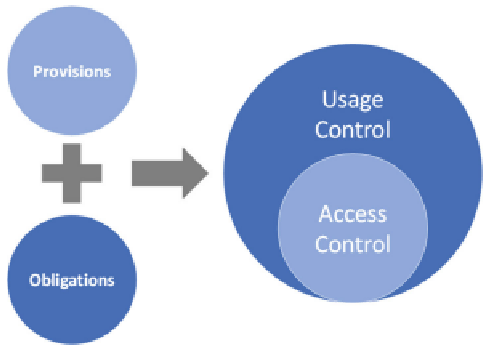

Figure 9: $AC \subseteq UCON$ **(taken from Jung et al. [11]) -**

after access has been granted, and it can adapt to changes in the environment.

Access control models, such as Discretionary Access Control (DAC), Mandatory Access Control (MAC), Role-Based Access Control (RBAC), and Attribute-Based Access Control (ABAC), offer varying degrees of complexity and flexibility. DAC and MAC, for instance, provide straightforward, structured control mechanisms, making them suitable for environments with clearly defined and static access requirements. RBAC and ABAC, on the other hand, offer more flexibility and are better suited to dynamic environments with more complex access requirements.

However, for systems that require control over not just access but also the usage of resources, Usage Control (UCON) provides a comprehensive solution. While UCON is inherently more complex due to its broader scope, it offers the flexibility to express a wide range of policies and adapt to changes in the environment.

In conclusion, the choice between access control and usage control models should be guided by the specific requirements of the system.

RQ2 What are the common design components of systems that handle control mechanics? We illustrate in fig. 8 the generalized design of policy reasoning systems. This diagram represents a state-updating event-loop, where users query (and get notified by) the policy enforcement services.

Systems in the literature represent a variation of it:

- Access Control: Hides the system behind the *Policy Reasoners* and acts as a gateway to the system services. In the context of an ABAC modelled system, the hidden services do not emit events to the Policy Reasoners, but they can update Attributes, effectively changing the access policies.
- Usage Control: In case of usage control we differentiate between the types of granularity of the usage control models. The more general case of UCON, is illustrated in 1, showing the general evaluation loop of the policy reasoner inside.
  - Pre-usage control: this can be viewed as access control, users ask permissions from the policy reasoners for they're usage. They are not monitored continuously if their usage keeps respecting the norms. It can be combined with post-usage checks, which represents auditing or offline usage control.
  - Continuous usage control, where the users interact with the system's assets and this usage is monitored by policy reasoners to respect norms online. This represents the more general model, and intuitively the more expensive one. The System continuously needs to emit events to the Policy reasoner and let it update its state.
  - User specified policies: Languages such as eFLINT, when forming the runtime of the policy reasoners and managing their state, empower users to not only perform actions but also modify the state of the policy runtime. This facilitates dynamism and the alteration of normative reasoning. For instance, in eFLINT, this can be achieved through defining new types and leveraging the *Extends* and *Syncs with* features.
  - Extra-deontic mechanics: Systems can be architected with policy reasoners that perform more than just deontic checks (e.g., an action violates the specified normative rules). They can also perform potestative checks, raising events related to a user's duties or detecting violations of unfulfilled duties.

  Potestative elements features of eFLINT are similar to UCON obligations, with the added benefit of eFLINT also

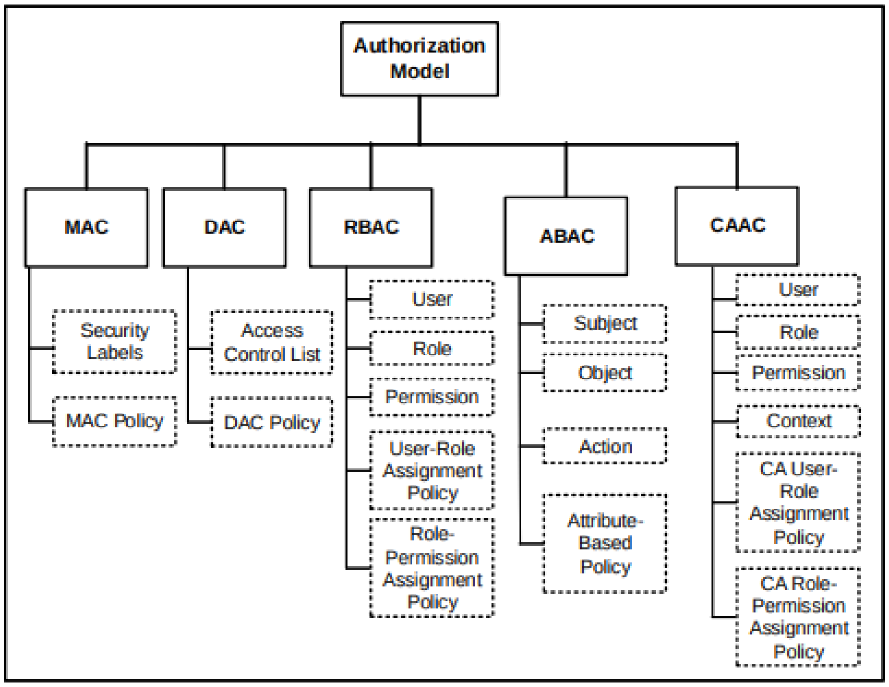

Figure 10: Taxonomy of Access Authorization Models (taken from [12]) -

issuing notification events to the actors. In UCON, on the other, subjects learn about their obligations on access and during usage.

- Composable Policies: [13] introduced in a multi-agent-system environment the idea of combining isolated policy reasoners with different components isolated on the user-side. Furthermore, methods are sketched out about how to combine the policy reasoners to create a higher level policy runner. These reasoners interact with the lower level reasoners, they update the beliefs of local policy reasoners and in turn influence the local agents decisions. While the local reasoners are designed to notify and aid in decision-making the local agents, the higher-level policy reasoner has a purpose of enforcing norms and triggering handling of eventual violations of the norms. Local policy reasoners, on the other hand, have a more consultative purpose, agents being allowed to deviate from norms when they don't correspond to the local agents *desires* as described in [13].

Difference between fig. 8 and fig. 1: fig. 8 in eFLINT depicts a loop that resembles the continuous checking of attribute values seen in fig. 1 of UCON, in the *ongoing* phase. The difference is that fig. 8 is more general in what is handled and is updating the normative state. In fig. 1 we have attributes and attribute updates updating permissions and potentially the future evaluations of the authorization reasoner.

A second difference is the possibility of systems following the fig. 8 to allow more than only deontic-style normative rules. Reasoners can notify users and systems of changes in the powers a normative actor might have.

Context Design. As discussed in Kayes et al. [12], there is a trend of designing the access control systems with a focus on designing context-awareness.

Similarly, it is possible for UCON models to employ similar techniques. The topic is explored in Bai et al. [1, 2].

Esterhuyse et al. [3] also introduce the idea of exposing the events to the policy reasoners. This way the running policy checkers are able to "build their own context state" through the monitoring of the event log. It is the checkers' choice which events to filter out and which events to filter in to update these internal states.

Designing the context the policy reasoners have access constitutes an important decision for the system. These decisions have an important impact on the flexibility of the subsequent policy specifications. Access to a well-designed context data can be leveraged into more powerful policies.

Concluding Remarks: A common feature among various policy reasoning systems, including certain implementations of RBAC, ABAC, UCON, and user-specified frameworks implementing policies in normative languages like eFLINT, is the utilization of a state-changing event loop handler. This mechanism, while not universally present, is integral to these systems when employed, enabling them to dynamically adapt to changes, enforce policies effectively, and maintain system integrity. It's important to note, however, that not all systems, particularly simpler RBAC implementations, may employ such a dynamic mechanism, instead relying on more static policy definitions and enforcement.

This shared characteristic has implications for the architecture of systems intending to implement any of these policy frameworks. The system design must be flexible enough to accommodate the event loop handler, which requires continuous monitoring of system events, efficient handling of these events, and dynamic updating of the system state in response to these events.

REFERENCES

[1] Guangdong Bai, Liang Gu, Tao Feng, Yao Guo, and Xiangqun Chen. 2010. Context-aware usage control for android. In *Security and Privacy in Communication Networks: 6th Itrnational ICST Conference, SecureComm 2010, Singapore, September 7-9, 2010. Proceedings 6*. Springer, 326–343.

[2] Guangdong Bai, Lin Yan, Liang Gu, Yao Guo, and Xiangqun Chen. 2014. Context-aware usage control for web of things. *Security and Communication Networks* 7, 12 (2014), 2696–2712.

[3] Christopher A. Esterhuyse, Tim Müller, L. Thomas Van Binsbergen, and Adam S. Z. Belloum. 2022. Exploring the Enforcement of Private, Dynamic Policies on Medical Workflow Execution. In *2022 IEEE 18th International Conference on e-Science (e-Science)*. 481–486. https://doi.org/10.1109/eScience55777.2022.00086

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

[5] David F Ferraiolo, Serban I Gavrila, and Wayne Jansen. 2015. Policy machine: features, architecture, and specification. (2015).

[6] Simon Godik and Tim Moses. 2002. Oasis extensible access control markup language (xacml). *OASIS Committee Secification cs-xacml-specification-1.0* (2002).

[7] Ali Hariri, Amjad Ibrahim, Theo Dimitrakos, and Bruno Crispo. 2022. WiP: Metamodel for Continuous Authorisation and Usage Control. In *Proceedings of the 27th ACM on Symposium on Access Control Models and Technologies*. 43–48.

[8] Matthew RA Heiman. 2019. The GDPR and the consequences of big regulation. *Pepp. L. Rev.* 47 (2019), 945.

[9] Kalle Hjerppe, Jukka Ruohonen, and Ville Leppänen. 2019. The general data protection regulation: Requirements, architectures, and constraints. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*. IEEE, 265–275.

[10] Vincent Hu, David Ferraiolo, D. Kuhn, A. Schnitzer, Knox Sandlin, R. Miller, and Karen Scarfone. 2014. Guide to attribute based access control (ABAC) definition and considerations. *National Institute of Standards and Technology Special Publication* (01 2014), 162–800.

[11] Christian Jung, Jörg Dörr, B Otto, M Ten Hompel, and S Wrobel. 2022. Data usage control. *Designing Data Spaces: The Ecosystem Approach to Competitive Advantage* (2022), 129–146.

[12] ASM Kayes, Rudri Kalaria, Iqbal H Sarker, Md Saiful Islam, Paul A Watters, Alex Ng, Mohammad Hammoudeh, Shahriar Badsha, and Indika Kumara. 2020. A survey of context-aware access control mechanisms for cloud and fog networks: Taxonomy and open research issues. *Sensors* 20, 9 (2020), 2464.

[13] Mostafa Mohajeri Parizi, L Thomas van Binsbergen, Giovanni Sileno, and Tom van Engers. 2022. A Modular Architecture for Integrating Normative Advisors in MAS. In *Multi-Agent Systems: 19th European Conference, EUMAS 2022, Düsseldorf, Germany, September 14–16, 2022, Proceedings*. Springer, 312–329.

[14] Jaehong Park and Ravi Sandhu. 2004. The UCONABC usage control model. *ACM transactions on information and system security (TISSEC)* 7, 1 (2004), 128–174.

[15] Ravi S Sandhu. 1998. Role-based access control. In *Advances in computers*. Vol. 46. Elsevier, 237–286.

[16] Ulrich Schöpp, Chuangjie Xu, Amjad Ibrahim, Fathiyeh Faghih, and Theo Dimitrakos. 2023. Specifying a Usage Control System. In *Proceedings of the 28th ACM Symposium on Access Control Models and Technologies*. 193–200.

[17] Daniel Servos and Sylvia L Osborn. 2017. Current research and open problems in attribute-based access control. *ACM Computing Surveys (CSUR)* 49, 4 (2017), 1–45.

[18] L Thomas van Binsbergen, Milen G Kebede, Joshua Baugh, Tom Van Engers, and Dannis G van Vuurden. 2022. Dynamic generation of access control policies from social policies. *Procedia Computer Science* 198 (2022), 140–147.

[19] L. Thomas van Binsbergen, Lu-Chi Liu, Robert van Doesburg, and Tom van Engers. 2020. EFLINT: A Domain-Specific Language for Executable Norm Specifications. In *Proceedings of the 19th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences* (Virtual, USA) *(GPCE 2020)*. Association for Computing Machinery, New York, NY, USA, 124–136. https://doi.org/10.1145/3425898.3426958

[20] Lourens E. Veen, Sara Shakeri, and Paola Grosso. 2022. Mahiru: a federated, policy-driven data processing and exchange system. *ArXiv* abs/2210.17155 (2022).