UCLouvain

École polytechnique de Louvain

Democratic smart home

An Access Control model for co-living smart home environments

Author: **Stanislas Gorremans**
Supervisor: **Etienne Riviere**
Readers: **Etienne Riviere, Gorby Kabasele Ndonda, Annanda Rath**
Academic year 2022–2023
Master [120] in Computer Science and Engineering

Acknowledgement

To my family, especially my mum, who supported and helped me make this thesis possible. Thank you.

To Prof. Riviere and Dr. Zavalyshyn for their meaningful advice and guidance during the year. Thank you.

This paper incorporates the use of artificial intelligence tools, specifically ChatGPT and DeepL, for tasks such as translation, rephrasing, and summarization of ideas. ChatGPT was utilized to assist in generating translations, rephrased sentences, and summaries, while DeepL was employed for translation purposes. However this document is the fruit of personal labor and the author takes all responsibilities for the content of this document.

Table of contents

1. Introduction 4
2. State Of The Art 7
   2.1. History 7
   2.2. Basic Access Control 8
      2.2.1. Discretionary Access Control 8
      2.2.2. Mandatory Access Control 9
   2.3. Role Based Access Control (RBAC) 9
      2.3.1. Standard for RBAC 9
      2.3.2. Generalized RBAC 15
      2.3.3. Extended Generalized RBAC 17
      2.3.4. Role-Based Administration of Role-Based Smart Home IoT 18
      2.3.5. Attribute-Based Access Control 19
      2.3.6. Hybrid models of RBAC and ABAC 20
      2.3.7. Optimistic access control 21
3. Model 22
   3.1. Model construction 23
   3.2. Democratic Model Formalization 28
   3.3. Use-cases 30
      3.3.1. Use case 1 30
      3.3.2. Use case 2 33
      3.3.3. Use case 3 35
   3.4. Summary of model 35
4. Proof of concept (PoC) 36
   4.1. Why Home Assistant? 36
   4.2. How does Home Assistant work internally? 36
      4.2.1. Glossary 37
      4.2.2. Architecture 37
   4.3. Changes made to Home Assistant 39
      4.3.1. Current Control Access in Home Assistant 39
      4.3.2. Introduction of role heritage 40
      4.3.3. Introduction of decision administration 41
   4.4. Examples 44

4.4.1. Example 1 44
4.4.2. Example 2 47
4.4.3. Example 3 49
5. Conclusion 52
6. Bibliography 54

1. Introduction

The concept of smart homes has its roots in the early 20th century, where inventors and visionaries sought to create more intelligent and responsive living spaces. At the 1939 New York World's Fair, Westinghouse Electric showcased the "Electric Home of the Future" with automated temperature control, and a door-answering robot [1]. In 1966, the first smart home computer called the "Electronic Information System" (ECHO IV) was developed by Jim Sutherland at Westinghouse. This pioneering system could manage lights, appliances and thermostats [2].

Advancements in microprocessors, personal computers, wireless communication and the internet in the late 20th century opened up new possibilities for device connectivity and control. In 1984, the American Association of House Builders coined the term "smart house" to describe homes leveraging advanced technology for enhanced comfort and efficiency [3]. Philips played a significant role in popularizing the concept with its vision of "Ambient Intelligence", which focuses on creating adaptable homes that cater to the preferences and needs of users. As a result, smart homes became increasingly accessible and affordable to consumers [4].

The growth of smart homes has been fueled by the rise of the Internet of Things (IoT) in the 21st century. The convergence of technology, innovation, and human needs has given rise to a fascinating history of IoT and smart homes. It showcases how these elements have come together to create new possibilities for living. IoT technology enables the connection of physical devices to the internet, allowing them to communicate and exchange data. This connectivity has revolutionized various aspects of our daily lives, including remote control, energy management, security and health monitoring. It enhances convenience and customization in smart homes. The story of IoT and smart homes reflects the ongoing evolution of technology and its profound impact on our living environment.

As an application in housing, we can see that smart home is rapidly penetrating urban environments where co-living experiences are on the rise. Some buildings are now specially designed for co-living. This type of building is more than a simple place to sleep. It can provide various amenities like laundry or shared bike parking and access to common spaces whether to cook in a big kitchen or to relax or work or study in dedicated spaces. In urban lifestyle, the people are more prone to follow the trends, such as smart homes through IoT.

The evolution of co-living structures raises challenges like privacy concerns, security breaches, and issues with sharing objects and services. Access control can be in some situations a solution, but it needs to be tailored to the specific needs of co-living residents. The challenge is how to manage access to smart devices in a dynamic and diverse setting that is not currently supported by existing systems. Older control access systems need to evolve to better incorporate technology, which is intrinsically bound up with contemporary lifestyles.

Access control models must evolve and adapt to new issues. They should reflect everyday practices rather than imposing strict rules. That technology is inspired by life and not the other way around should be the guiding principle.

Conventional access control systems operate under the authority of a single administrator, resembling an autocratic system. However, historical evidence suggests that individuals desire a voice in decision-making processes to counteract the tyranny of an non-concerted decision. This demand has fostered the advancement of voting systems.

Voting has been an integral part of democracy for an extensive period. Its origins can be traced back to the Athenian democracy, followed by its adoption in Rome, where a simple majority was considered satisfactory [5].

Historically, it is crucial to consider the development of voting systems, which have evolved in various forms, such as open or closed, majority or proportional, and with suffrage based on census or equal voting. These diverse ways of expressing opinions through voting should be reflected upon. Different systems should be chosen accordingly, depending on the intended purpose. This is especially relevant in the context of participatory democracy within a small community.

In the current smart home systems, the single administrator appears to be inadequate in certain situations where it would be beneficial to distribute decision-making power and enable collective decision-making.

Three practical cases can be examined. To provide a clearer explanation of the different steps, the examples used to illustrate the system have been intentionally simplified. The examples serve to illustrate a general process rather than a specific usage. In reality, it will become more complex as a user will belong necessarily to multiple subgroups: place of residence (such as "*kots*" or dormitories divided into multiple floors), gender (boys/girls), common points of interest (such as playing music or practicing a sport), ... etc. The proposed system is capable of handling inherently complex situations.

To begin with, we will consider a situation where a differentiation is introduced between object administration and access to the same object. Subsequently, we will explore how to expand access to an object while preserving the oversight rights of other users. Lastly, it is interesting to investigate a scenario where an additional member is added to the decision-making authority.

1. Reduction of volume on speaker

   A co-living space could offer a big common room accessible to all residents to party and share a good time with other residents. But the noise emanating from the residents partying in this room could sometimes excessively disturb some of the other residents.

   To address this issue, the residents have decided to install a speaker system. Everyone is committed to creating a harmonious and pleasant living environment, and to that end, any member of the building can limit the volume of the speaker. This decision was taken, as any decision to be taken for the co-living, through consensus decision-making during a general meeting of the co-ownership.

   The system allows any resident to reduce the excessive noise level if they occasionally need a particular silent period of work or study or a restful sleep after a period of intense work. On the other hand the rest of the time the party room can be used for the purpose for which it was intended, i.e. partying and sometimes loudly.

2. Grant access to visitor

   In this shared building with a smart audio system in the party room, visitors are limited to the noise level voted on at the meeting, since they are guests and not members.

   What happens if a resident wants to allow a visitor to exceed the sound limit applied to all visitors? For instance, to have a guest become a DJ.

Any resident can ask the other residents to host a guest as DJ. At least one other resident should consent to the exceeding of the sound limit applicable to the visitors.

3. Add member to decision power

   A couple frequently chooses to go out by subscribing to a theater membership. Small children are not allowed in these events. To address this, the couple relies on a trusted babysitter who regularly cares for their children. With complete confidence in the babysitter, the parents wish to grant the babysitter membership on the decision board for the TV and gaming console. This would enable the babysitter to control and manage the children's access in the same manner as the parents.

In view of the problems raised, an access control model can be developed on the basis of an existing system. It's important to build on the continuity of systems that have already been widely developed, and which the public can easily understand and wisely use. What's more, developments have already been studied in existing systems, the results of which can be leveraged.

With this in mind, the first step is to present a State-of-Art analysis of the current situation. This will enable us to build a theoretical model, which will logically be followed by a Proof-of-Concept.

2. State Of The Art

This chapter delves into a comprehensive exploration of existing access control models with the aim of deepening the understanding of their limitations and areas for improvement especially for smart homes. The investigation begins with a brief historical overview, tracing the approach to access control employed by humanity prior to the advent of computers. By examining these earlier practices, a foundation is established for the subsequent examination of two fundamental models that, despite their simplicity, have yielded significant insights.

The subsequent sections delve into the evolution of Role-Based Access Control (RBAC), a model that categorizes users into groups to enhance access control management. RBAC assumes a central role in this paper as a fundamental reference point for exploring various concepts and ideas. Furthermore, the analysis focuses on how RBAC addresses the administration of user groups, laying the foundation for introducing an alternative approach in this master's thesis.

Moreover, various other access control models that challenge conventional notions are explored, offering a glimpse into the ongoing developments in this field. By providing insights into these alternative models, an overview of the current landscape of access control research is presented.

2.1. History

Access control has a rich historical background, stemming from the need to restrict entrance to authorized individuals and entities. In the early days, access control was primarily enforced through physical means, with human intervention ensuring that only permitted individuals gained entry to properties, buildings or rooms. However, the desire for automated access control systems led to the development of mechanical locking systems, which eliminated the need for constant human presence.

The origins of mechanical locks can be traced back to ancient civilizations [6]. While the first mechanical lock is often attributed to Egypt, primitive tumbler locks have also been discovered in the ruins of Iraq dating back to the 7th century BC. These early locks featured wooden keys with small brass pins that aligned with corresponding pins in the lock, allowing the bolt to be moved aside and granting access.

In the 20th century the concept of centralized access control and the ability to modify access rights gained finally prominence [7]. Access control mechanisms shifted from a focus on outright prohibition to a more nuanced approach of controlling and managing access to resources. This marked the birth of the formalized concept of access control as we understand it today.

In fact, the landscape of access control underwent a profound transformation with the advent of computers. A significant milestone was reached in the 1960s [8] when password-based user authentication systems were introduced, notably by MIT's Compatible Time-Sharing System (CTSS). CTSS, a pioneering time-sharing operating system, revolutionized the concept of access control by enabling multiple users to access a single computer.

As technology advanced, access control evolved and became increasingly automated. Modern access control systems incorporate sophisticated mechanisms and technologies to ensure secure and efficient access management in a wide range of contexts, including physical facilities, digital networks, and information systems. The continuous development and refinement of access control techniques reflect

society's ongoing efforts to strike a balance between providing authorized access and safeguarding resources from unauthorized individuals or entities.

Smart home is only a small part of all access control needed. But due the increasing popularity of smart home systems the need to have a access control system for everyone with different needs leads to research and experiments.

2.2. Basic Access Control

As computer-based access control was needed in the sixties. Two very popular access control models were formalized. Due to their simple configuration, they are not well suited for complex system, but they did pave the way.

2.2.1. Discretionary Access Control

Discretionary access control (DAC) [9] focuses access control on the discretion of the resource owner in determining access rights. It allows the resource owner to define and modify access permissions for individual subjects or groups based on their identity.

DAC is based on a few components, namely subjects, objects, access permissions, and access control lists (ACLs). Furthermore, all interactions between these components are visually depicted in Figure 1. Subjects refer to the individuals. Processes or entities seeking access refer to resources. Objects represent the resources themselves, such as files, folders, or databases. Access permissions define the specific actions that a subject can perform on an object, such as reading, writing, executing, or deleting. These permissions are typically set by the resource owner or an authorized administrator. Access control lists (ACLs) are data structures associated with each object, containing entries that specify the subjects or groups and their corresponding access permissions for that particular object. Together, these components form the basis of DAC, enabling resource owners to exercise discretion in determining access rights for subjects based on their identities and relationships.

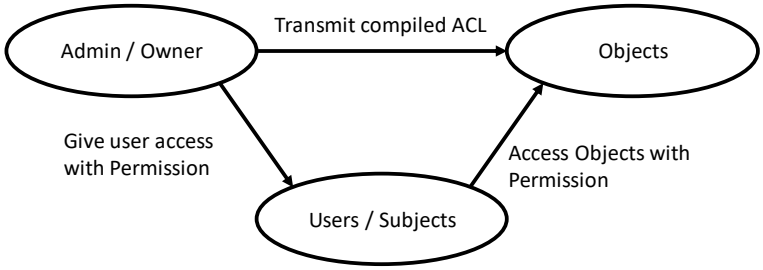

Figure 1: Discretionary Access Control diagram inspired from the book Encyclopedia of Cryptography and Security [9]

Discretionary Access Control (DAC) provides a user-centric approach to access control. It allows resource owners to manage access permissions for their resources. DAC is commonly used in basic access control systems found in file systems, collaboration platforms, web applications, and database management. Similar implementations of DAC can be found in smart home systems, such as Home Assistant.

2.2.2. Mandatory Access Control

Mandatory Access Control (MAC) is an access control mechanism that utilizes clearance levels to restrict access [10]. This approach is depicted in Figure 2. In MAC, each user (subject) and resource (object) is assigned a specific clearance level. Access is granted only if the user's clearance level is equal to or higher than that of the resource. MAC is commonly employed in military applications due to its rigid framework.

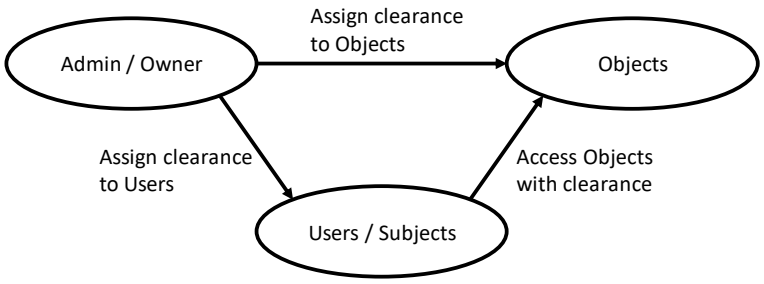

Figure 2: Mandatory Access Control (MAC) inspired from the book Encyclopedia of Cryptography and Security [9]

Although MAC may not be practical for many use cases. It offers an abstraction from both the user and resource perspectives by setting clearance levels. This abstraction follows a hierarchical structure. Despite being less commonly used, this hierarchical form of MAC has contributed to the expressiveness of other access control systems, including those implemented in smart home applications.

2.3. Role Based Access Control (RBAC)

Role Based Access Control (RBAC) has emerged as an additional access control model, introducing abstraction on the user side by grouping users based on the roles they play within in business world. Initially, RBAC was informally used alongside basic access control. However, the National Institute of Standards and Technology (NIST) played a crucial role in standardizing RBAC in the early 2000s to ensure consistency [10]. The NIST standard remains the authoritative reference for RBAC.

In this section, we will explore the NIST standard for RBAC and delve into further extensions specifically designed for smart homes and IoT applications.

2.3.1. Standard for RBAC

The National Institute of Standards and Technology (NIST) standard encompasses four types of Role Based Access Control (RBAC) that progressively build upon each other: flat, hierarchical, constrained, and symmetric. We will examine each type to understand their composition and benefits. Table 1 presents an overview of all the different layers. Each layer will be detailed.

Level	Name	RBAC Functional Capabilities
1	Flat RBAC	• users acquire permissions through roles • must support many-to-many user-role assignment • must support many-to-many permission-role assignment • must support user-role assignment review • users can use permissions of multiple roles simultaneously
2	Hierarchical RBAC	Flat RBAC + • must support role hierarchy (partial order) • level 2a requires support for arbitrary hierarchies • level 2b denotes support for limited hierarchies
3	Constrained RBAC	Hierarchical RBAC + • must enforce separation of duties (SOD) • level 3a requires support for arbitrary hierarchies • level 3b denotes support for limited hierarchies
4	Symmetric RBAC	Constrained RBAC + • must support permission-role review with performance effectively comparable to user-role review • level 4a requires support for arbitrary hierarchies • level 4b denotes support for limited hierarchies

Table 1 An overview of the layers in the National Institute of Standards and Technology's Role Based Access Control standardization

2.3.1.1. Flat RBAC

Flat RBAC is a foundational level of RBAC systems, providing a flexible framework for access control management. It enables users to possess multiple permissions and allows permissions to be allocated to multiple users, establishing a many-to-many relationship.

Roles are introduced in Flat RBAC as an abstraction layer between permissions and users. They categorize permissions based on criteria such as tasks or responsibilities, aligning the access control system with the organizational structure. Defining roles facilitates efficient management.

Furthermore, to support efficient management, Flat RBAC employs two assignments. The user-role assignment associates users with specific roles, while the permission-role assignment connects permissions with roles. These assignments are visually depicted in Figure 3.

In all the diagrams of this paper, sets are depicted as ovals, and binary relations are represented by arrows. A single arrow denotes a "one" relationship, while a double arrow signifies a "many" relationship, particularly in the relationship between sets. And all the dotted lines means constraints.

In addition, the review process of the User Assignment in Flat RBAC is a crucial aspect. It allows organizations to determine the roles assigned to a user and identify users associated with specific

roles. This process helps ensure adherence to the principle of well-known least privilege[1] by detecting anomalies and correcting unnecessary permissions for roles and users.

Lastly, RBAC acknowledges that users often have multiple roles in an organization. This recognition allows users to leverage permissions from multiple roles concurrently, providing flexibility and efficiency in accessing the necessary permissions for their diverse responsibilities.

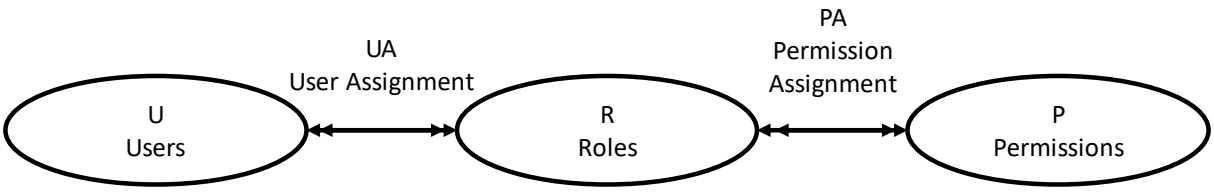

Figure 3 Flat Role Based Access Control visually depicted, inspired by NIST standardization of RBAC

2.3.1.2. Hierarchical RBAC

Unfortunately, Flat RBAC lacks a hierarchical structure commonly found in human communities. Hierarchical RBAC addresses this limitation by introducing support for role hierarchies. In this model, roles are organized in a partial order, representing a seniority relation where senior roles inherit permissions from junior roles. The senior roles are typically depicted at the top, while the junior roles are positioned at the bottom in diagrams.

NIST defines two sub-models within Hierarchical RBAC: general and limited Hierarchical RBAC. Limited Hierarchical RBAC restricts the structure to simple configurations like tree or inverted tree structures. On the other hand, general Hierarchical RBAC allows for arbitrary structures, accommodating diverse organizational needs. The use of limited inheritance in Hierarchical RBAC helps mitigate the risk of concentrating excessive power in a single role, reducing the potential for significant mistakes or misuse of permissions.

In Figure 4, three hierarchical structures are depicted. The inverted tree hierarchy represents an Engineer Department (ED) consisting of two teams: Engineers E1 and E2. Each team has a Quality Engineer (QE) and a Production Engineer (PE). In this structure, engineers inherit all the permissions associated with the roles below them.

In hierarchical tree structure the top position is held by the Director (DIR), who manages Project Leaders (PL). The Project Leaders lead different Engineering teams that include both Quality Engineers and Production Engineers.

On the last graph, we can observe a general hierarchy that can take the form of trees or inverted trees. It is important to underline that loops are not allowed in the hierarchy structure.

[1] "The principle that a security architecture should be designed so that each entity is granted the minimum system resources and authorizations that the entity needs to perform its function." From NIST glossary [11]

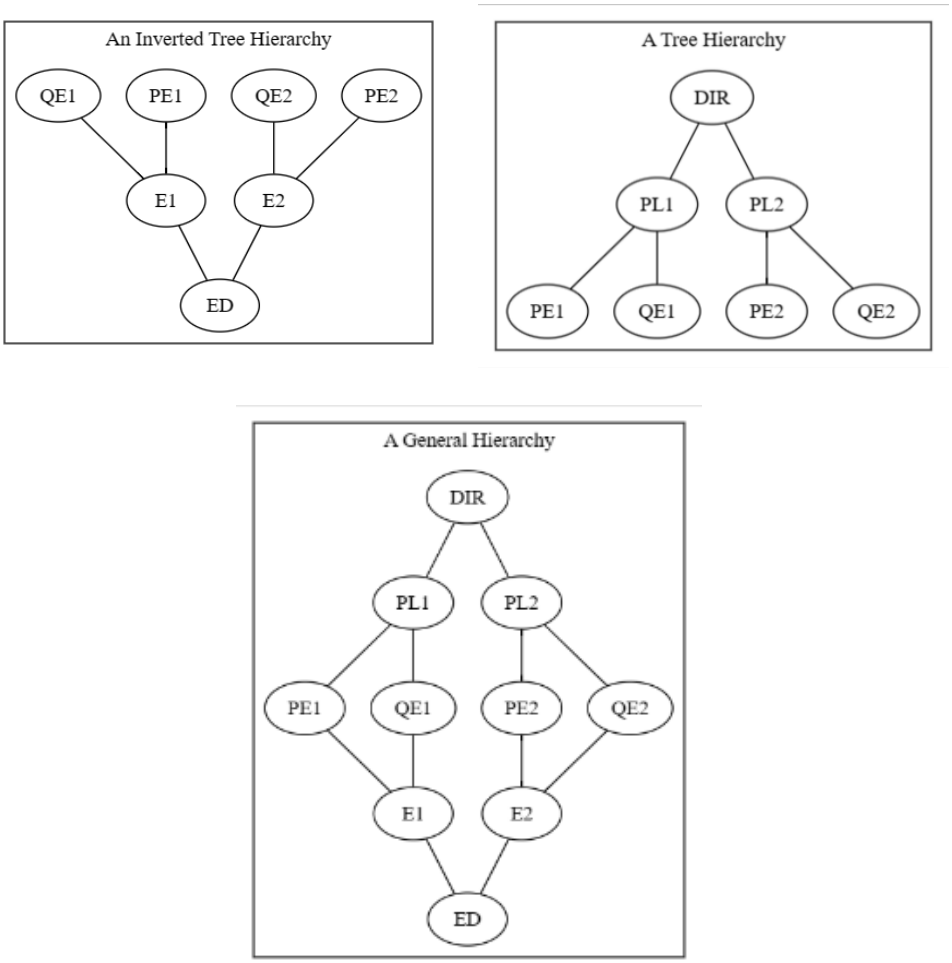

Figure 4 On the first line, we have an inverted tree structure and a tree structure, respectively, representing limited hierarchy. On the second line, we have the general hierarchy. These figures are based on the NIST RBAC standardization.

In smart home systems, we often encounter scenarios where permissions are granted to multiple groups. There may also be an upper group that requires permissions from multiple roles concurrently. General hierarchy, although not universally implemented, is beneficial for representing complex structures.

Role hierarchy is depicted in Figure 5. We start with Flat RBAC as the foundation and incorporate a many-to-many connection between roles. This relationship, known as Role Hierarchy (RH), visually represents the hierarchical structure and the connections between roles in the RBAC system.

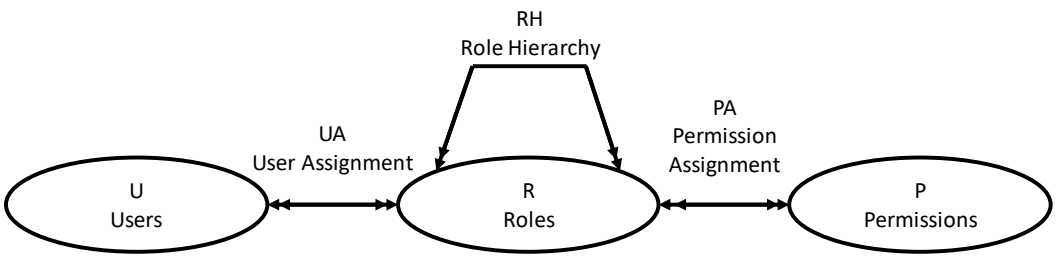

Figure 5 Hierarchical Role Based Access Control visually depicted, inspired by NIST standardization of RBAC

2.3.1.3. Constrained RBAC

Constrained RBAC complements Hierarchical RBAC with the concept of Separation of Duties (SoD), which is particularly relevant in business contexts to prevent fraudulent activities by ensuring that a single user cannot perform multiple roles simultaneously. However, in the smart home context, where users are typically curious rather than malicious, the need for strict Separation of Duties is less pertinent.

In addition to SoD, Constrained RBAC introduces the concept of sessions, where users can activate and deactivate their own roles. This concept provides a dynamic aspect to the access control system, allowing users to perform certain roles even in situations where SoD constraints would typically restrict them. This dynamic nature adds flexibility to the system compared to the static nature of User Assignments.

The constraints in Constrained RBAC can be applied either statically or dynamically. Figure 6 and Figure 7 illustrates respectively the Hierarchical RBAC structure with static and dynamic SoD constraints.

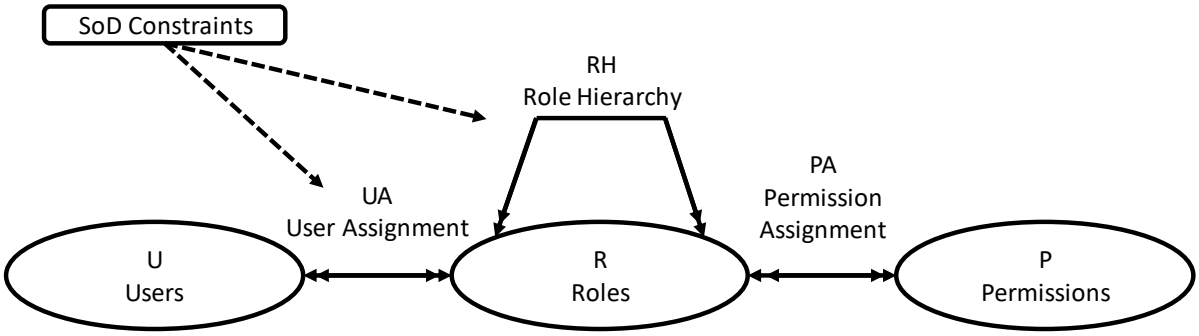

Figure 6: Static Constrained Role Based Access Control visually depicted, inspired by NIST standardization of RBAC

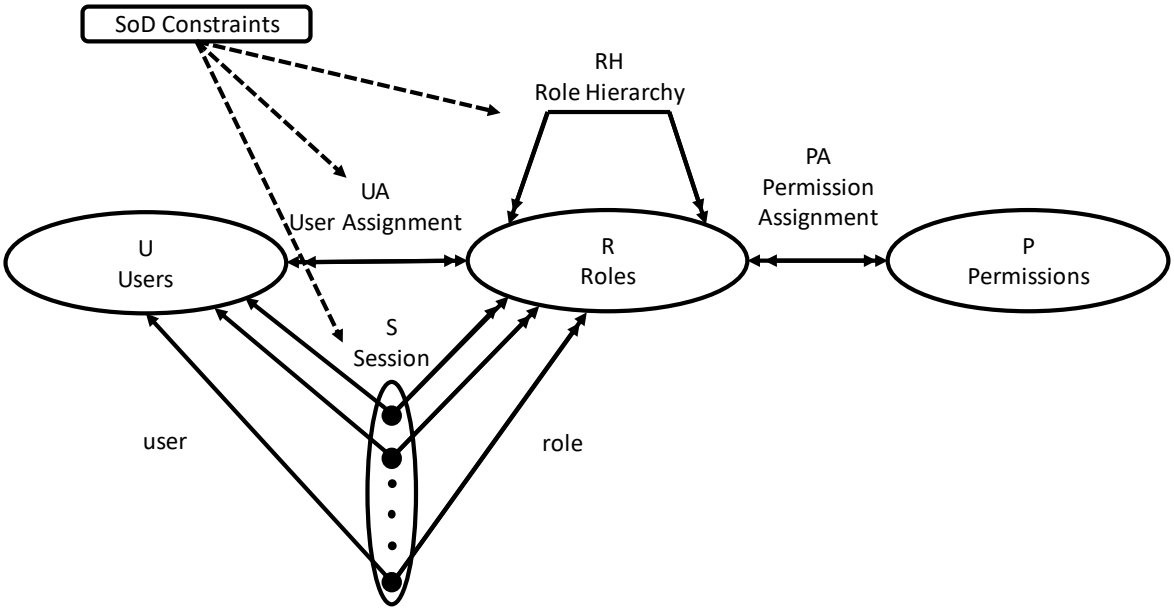

Figure 7: Dynamic Constrained Role Based Access Control visually depicted, inspired by NIST standardization of RBAC

2.3.1.4. Symmetric RBAC

Symmetric RBAC extends Constrained RBAC by enabling permission-role review. It upholds the principle of least privilege, allowing for evaluation and adjustment of permissions as needed. Unlike a static approach, Symmetric RBAC recognizes the dynamic nature of modern environments, where permissions must adapt to changing needs in smart home systems. This flexibility ensures users have appropriate access rights for their responsibilities.

In addition, Symmetric RBAC offers the option to apply Separation of Duties [2](SoD) constraints to permission assignments, although it may have limited relevance in smart home contexts. Nonetheless, this feature is important for future models built on this standard.

Figure 10 and Figure 11 illustrate the integration of SoD constraints in Permission Assignments, providing a comprehensive overview of how Symmetric RBAC incorporates SoD principle.

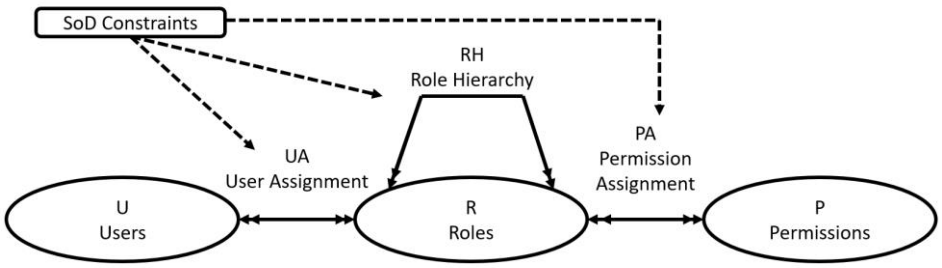

Figure 8: Static Symmetric Role Based Access Control visually depicted, inspired by NIST standardization of RBAC

[2] "Refers to the principle that no user should be given enough privileges to misuse the system on their own." From NIST Glossary [12]

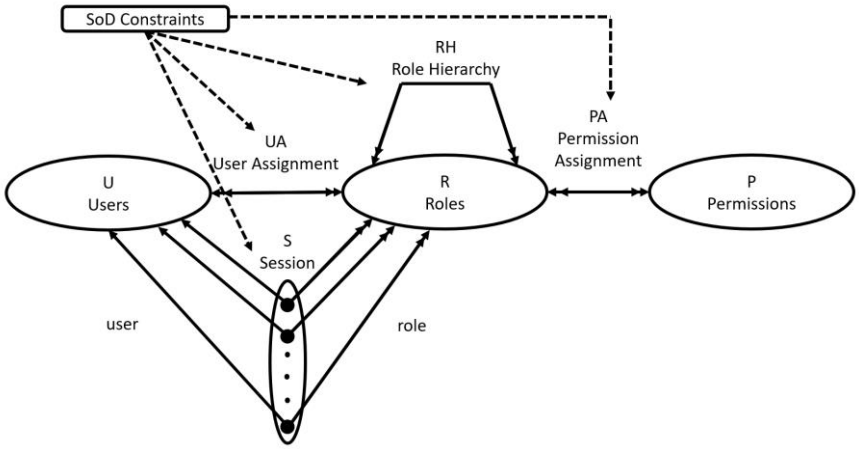

Figure 9: Dynamic Symmetric Role Based Access Control visually depicted, inspired by NIST standardization of RBAC

2.3.1.5. Finishing words for NIST RBAC model

In conclusion, the NIST standard for RBAC offers valuable guidelines for the implementation of access control systems, covering various aspects of RBAC. The exclusion of negative permissions in the standard eliminates potential confusion and encourages the use of well-defined constraints to achieve desired outcomes. The standard purposely avoids specifying the nature of permissions, allowing organizations to adapt them according to their specific use cases.

Furthermore, the NIST standard refrains from prescribing specific methods for assigning users to roles, assigning permissions to roles, or establishing role relationships. This flexibility accommodates the diverse requirements of different organizations.

However, there are still unresolved issues within the standard. One significant concern is the role revocation, especially in dynamic environments. Determining whether to revoke roles during an active user's session poses a complex challenge, particularly in distributed systems.

Overall, the NIST standard serves as a comprehensive framework for RBAC implementation, but it acknowledges the ongoing need for further exploration and refinement. Organizations adopting RBAC should consider these factors and customize their approach to align with their specific needs and operational dynamics. By adhering to the principles and recommendations outlined by NIST, organizations can establish robust and secure access control systems that effectively address their unique requirements.

Having established that the NIST RBAC standard provides a solid foundation, it is worth noting that several researchers have expanded upon it. Let us explore a few examples of such extensions.

2.3.2. Generalized RBAC

Generalized Role-Based Access Control (GRBAC) [12] extends the traditional RBAC model by introducing additional role types beyond user-centric roles. GRBAC introduces Object Roles to organize permissions more effectively and Environment Roles to consider additional parameters such as location or time of day. This makes GRBAC a hybrid between RBAC and attribute-based access

models (ABAC), as explained in chapter 2.3.6. With user roles, object roles, and environment roles, GRBAC provides a comprehensive and flexible access control framework.

Although GRBAC was not specifically designed for IoT or smart homes, it offers valuable abstractions that can be utilized in other models. To provide a comprehensive understanding, we present a diagram, of the GRBAC model, highlighting the non-user-centric roles (see Figure 10). This diagram illustrates the distinction between user roles and other types of roles, showcasing the versatility of the GRBAC framework.

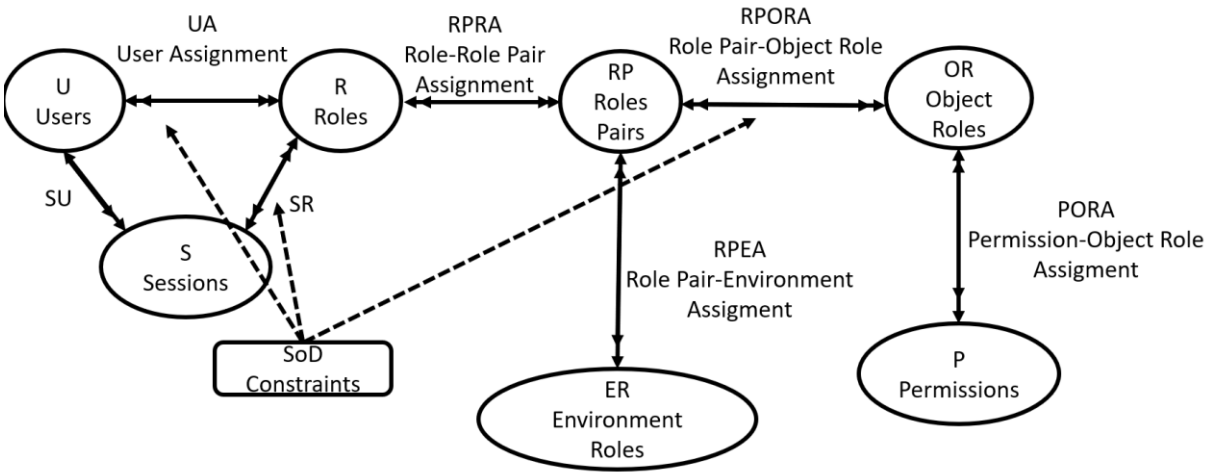

Figure 10: Generalized Role-Based Access Control model inspired from the EGRBAC paper [13]

2.3.3. Extended Generalized RBAC

Extended Generalized Role-Based Access Control (EGRBAC) [13] emerges as a refined adaptation of the original GRBAC model, specifically tailored for the unique challenges posed by smart homes in the field of IoT. The aim of EGRBAC is to explore the limitations of applying RBAC concepts within the context of home IoT systems.

One notable modification compared to GRBAC is the redefinition of the "Object" component, which is subdivided into devices and operations. This adjustment allows a better representation of IoT-enabled smart homes environments.

Authorization challenges in home IoT systems arise in three significant areas. Firstly, the presence of multiple users interacting with the same device, such as a smart door lock, introduces unique considerations and complexities when managing access rights. This scenario requires careful handling to ensure easy and secure shared usage. Secondly, the intricate social relationships among household occupants create novel threat models. Situations like a mischievous child attempting to manipulate the smart light in their sibling's room or a current or former partner attempting to exploit the privileges of other residents highlight the dynamics at play. Lastly, the prevalence of IoT devices lacking conventional input interfaces, such as screens and keyboards, presents a hurdle in implementing convenient and hands-free access control mechanisms. Overcoming these challenges require innovative solutions that balance usability with robust security measures.

Given these characteristics and challenges, there is a clear need for a dynamic and fine-grained access control model specifically designed for smart home IoT systems. EGRBAC aims to provide such a solution, enabling the establishment of precise and adaptable access control policies that effectively govern the interactions between users and resources within the constraints of the smart home environment. By addressing the unique challenges posed by smart homes, EGRBAC strives to enhance security and privacy while ensuring seamless and efficient user experiences in IoT-enabled domestic settings. Unfortunately EGRBAC has a unique administration, which makes an asymmetrical relation between the administrators who can decide of the assignments and the other users who are completely submitted to the administration.

Presented below as Figure 11, we observe the graphical representation of the separation between device and operation, a crucial requirement in smart home and IoT environments.

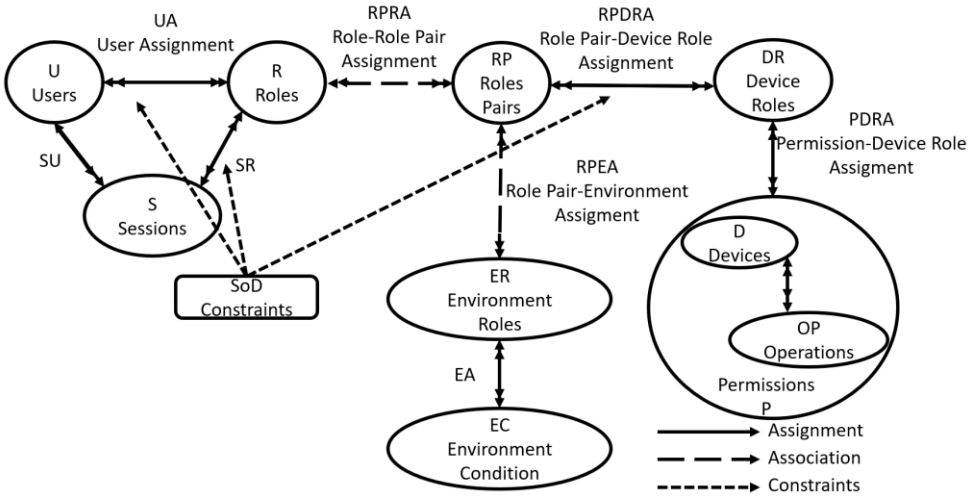

Figure 11: Extended Generalized Role-Based Access Control model, figured inspired from the EGRBAC paper [13]

2.3.4. Role-Based Administration of Role-Based Smart Home IoT

The proposed model by *Mehrnoosh Shakarami and Ravi Sandhu* [12], known as Role-Based Administration of Role-Based Smart Home IoT, builds upon the foundation of EGRBAC and introduces additional concepts to enhance administrative control. In traditional systems, a single administrator or a group of administrators handled all assignments, resulting in non-democratic behaviors and an asymmetric power dynamic between administrators and users.

To address this issue, researchers in the late nineties [13] proposed an administration model that can be summarized as RBAC administration for RBAC. While it was an improvement over the traditional single administrator approach, it still faced the same problem as RBAC itself, as administrators had the sole authority to determine who can become an administrator and manage user assignments.

In their paper, "Role-Based Administration of Role-Based Smart Home IoT," the researchers propose adapting RBAC administration for smart home applications. The model adds an

administrative layer on top of the operational model, as shown in Figure 12. This idea will be partially incorporated into my master's thesis model.

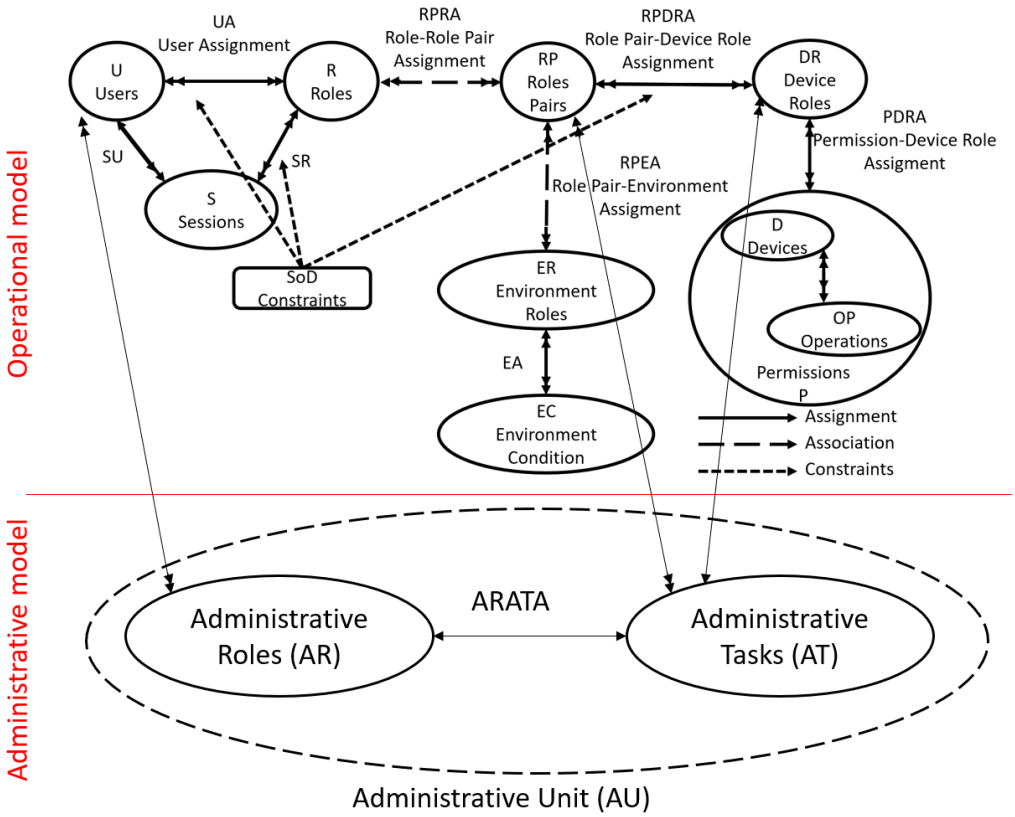

Figure 12 Administrative model inspired by a diagram the original paper [14]

Having explored extensions that build upon RBAC, which focus on aspects such as administration separation and the introduction of abstractions for IoT devices, we will now delve into other domains of access control to gain a comprehensive understanding of the current state-of-the-art.

2.3.5. Attribute-Based Access Control

ABAC, or Attribute-Based Access Control, offers an alternative approach to granting or denying user requests based on a combination of user attributes, object attributes, and environmental conditions. This method takes into account globally recognized attributes and conditions that are relevant to the specific policies in place.

In ABAC, access to resources or actions is determined by evaluating attributes associated with the subject (user), object, requested operations, and environment conditions. By considering these attributes in policies or rules, ABAC enables authorization based on specific attribute combinations. Figure 13 illustrates this concept.

For instance, consider a scenario where access to company facilities is restricted. In an ABAC system, authorization for access might be granted only if the subject possesses a valid company badge (a user attribute) and if the current time falls within the designated working hours (an environmental condition). The evaluation of these attributes and conditions against the established

policies or rules will dictate whether the requested operations, such as entering the facility, are allowed or denied.

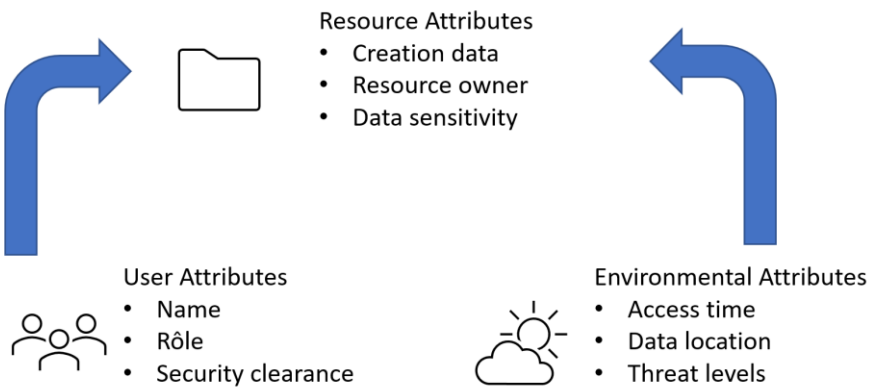

Figure 13: Presentation of an ABAC model inspired by solarwindssoftware [15]

By leveraging ABAC, organizations can achieve a more granular and context-aware access control system, enhancing security and enabling efficient resource allocation.

While ABAC offers numerous advantages, it is important to acknowledge its associated drawbacks. One notable challenge lies in the intricate task of precisely reviewing user access to specific permissions. Due to the dynamic nature of attribute-based evaluations, determining the complete set of permissions granted to a particular user can become arduous. As the number of attributes and policies grows, managing and maintaining this complexity becomes increasingly burdensome.

Furthermore, the decentralized nature of attribute-based policies can introduce challenges in maintaining a comprehensive overview of access control. Unlike role-based systems, where permissions are typically assigned to predefined roles, ABAC allows for more fine-grained control. This granular approach makes it difficult to create a concise and easily auditable list of all permissions granted to a specific user.

All those drawbacks made us decide to build upon on RBAC system instead. We consider that it was more pertinent to build a system which is more auditable and understandable.

2.3.6. Hybrid models of RBAC and ABAC

As discussed previously, several researchers and experts have made efforts to enhance the RBAC system by incorporating elements of ABAC (Attribute-Based Access Control). One key addition has been the introduction of environmental conditions, which provide a higher level of expressiveness to RBAC while still allowing for effective review mechanisms.

By incorporating environmental conditions into RBAC, organizations can strike a balance between adaptability and manageability. The review mechanisms can still be applied to ensure that access control policies remain in compliance with the organization's requirements. This combination empowers organizations to effectively manage access control while considering the dynamic nature of environmental factors.

2.3.7. Optimistic access control

Access control systems typically follow a default-denied approach, where access is not granted unless explicitly allowed. An alternative approach called Optimistic Access Control [16] challenges this traditional paradigm.

The fundamental principle of Optimistic Access Control is to assume that all access requests are permissible by default, granting access without intervention. It differs from the default-deny approach by only intervening when the user's behavior deviates from the expected or authorized actions. This approach aims to streamline the access control process and reduce the need for constant verification and permission management.

Although the optimistic approach offers advantages in specific contexts. However, it may not be universally applicable, especially when dealing with sensitive devices or entities. In these cases, a more prudent and stringent access control methodology is often necessary to minimize potential vulnerabilities and safeguard the system's confidentiality and integrity.

One notable advantage of Optimistic Access Control is its potential to streamline the management of smart home systems. By assuming de facto access, it reduces the complexity associated with coordinating multiple smart hubs or devices that would otherwise need to constantly communicate and validate permissions. This streamlined approach can enhance the efficiency and user experience of smart home environments.

While the idea of allowing permissive access by default may sound appealing, it could pose risks in the context of smart homes. Users may unintentionally expose private objects if access is not restricted. Therefore, it is important to consider a more collaborative approach built on top of the optimistic model. Differentiating between sensitive and non-sensitive devices and implementing different administration strategies can help mitigate these risks and ensure appropriate access control in the smart home environment.

3. Model

In the previous chapter on the State-of-the-art, we discussed existing systems developed for smart homes, with a specific focus on enterprise or single-family home environments. However, it seems that there is a need for a smart home control access system tailored for shared living spaces, where the democratic administration should play a central role.

We proposed the creation of a system that builds upon the foundation of a traditional Role-Based Access Control (RBAC) system. We incorporated the concept of Device Roles from Extended Generalized RBAC (EGRBAC) [13] and drew inspiration from the administration structure of RBAC administration layered on top of EGRBAC [14].

Current administration models are based on RBAC where all the power it kept in the handle of a few administrators . We propose in this paper model new administration for co-living environment, where the administration is handled by decision makers through a voting system. This shift promotes inclusivity and transparency, which are crucial in the context of co-living where residents share common spaces and value their privacy. The participatory nature of the proposed administration model fosters trust among residents, as they have the opportunity to be actively involved in the decision-making process. This inclusion not only empowers individuals but also strengthens the sense of community and cooperation within the co-living environment.

To ensure clarity and a shared understanding, it is important to define key terms in the model. In this context, an "action" refers to a change in the administrative model, and each action is subjected to a voting process based on predefined governance rules. The outcome of this process is a "decision" that determines whether the proposed action will be implemented. As the system operates on a decision-based approach, multiple decisions can accumulate, and an action is executed only when the votes yield a positive result. This process governs various aspects, including role hierarchy, role-to-user assignments, permission assignments, and overall system administration

This chapter will be structured in four step. Firstly, we will present the structure of the model (named Democratic Model). Following that, we will provide a formalized model. Afterward the use cases presented in the introduction will be mapped into the model. Finally, this chapter concludes with a summary of the key points discussed.

3.1. Model construction

In this section, we will explore the structure of the model, as illustrated in Figure 14. We will provide a step-by-step explanation of each element represented in the diagram. It is worth noting that the overall structure of the model has remained largely consistent with previous models presented. This intentional decision reflects the desire to build upon the work and reflections of other researchers in the field. Most concept come from older papers [10], [13], [14], [17]. The novel concepts come in how the model handles administration.

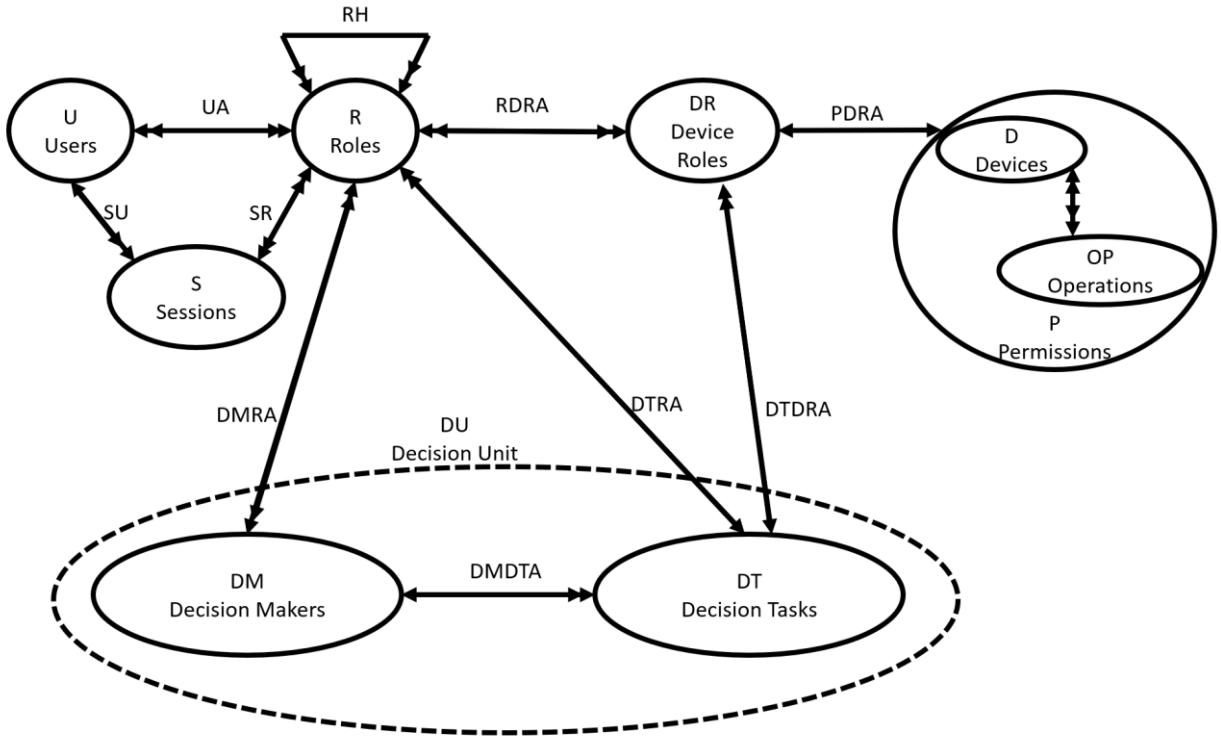

Figure 14: The democratic administration model, a novel approach proposed in this master's thesis.

Users, Roles and Sessions

In RBAC systems, there are common sets such as Users (U), Roles (R), and Sessions (S). A user refers to a person being who interacts with smart home. The User-Role Assignment (UA) which is many-to-many, specifies the assignment of users to roles. Users can establish sessions during which they can activate a subset of the roles assigned to them. It is possible for a user to have multiple active sessions simultaneously. The Session-User Assignment (SU) is many-to-one and maps each session to its unique controlling user. Similarly, the Session-Role Assignment (SR) is many-to-many and maps each session to the set of roles associated with it.

In Table 2 afterwards, we provide example data to illustrate the various notions discussed. The table includes two Users, Alice and Bob, and three Roles: Visitor, Resident, and SharedLight. Alice is assigned the Visitor ole and the SharedLight role, while Bob is assigned the Resident role. Additionally, Alice has an active session for the Visitor role but has disabled her SharedLights role, whereas Bob has chosen to have the Resident role active.

User (U)	Alice, Bob
Role (R)	Visitor, Resident, SharedLights
User Role Assignment (UA)	(Alice, Visitor), (Alice, SharedLights) (Bob, Resident),
Session (S)	Session_1, Session_2
Session User Assignment (SU)	(Alice, Session_1), (Bob, Session_2)
Session Role Assignment (SR)	(Session_1, Visitor), (Session_2, Resident)

Table 2 Example data to present Users, Roles and Session for the democratic model

Devices, Operations, Permissions and Device Roles

A Device (D) represents a smart home device, and Operations (OP) denote specific commands or actions that can be executed on these devices according to their manufacturer-defined functionalities. Permissions, in this context, signify the granted authorization to perform a particular operation on a specific device. They are represented as pairs comprising a device and an operation (D, OP) to indicate the approved action. The set of permissions, denoted as P, forms a subset of D × OP.

Both in EGRBAC and our democratic model, Device Roles (DR) are employed to categorize permissions associated with different devices. The assignment of permissions to device roles is regulated by the many-to-many Permission-Device Role Assignment (PDRA) relation.

Table 3 provides a visual representation of dummy values to exemplify the type of data that could be stored in each set. Specifically, we begin with the Device Role "HVAC," which consists of three linked devices: Heating, Ventilation, and Air Conditioning. Each device is associated with the "On" and "Off" operations. These devices and their corresponding operations are then assigned to the permissions set. In addition, we introduce the Device Role "LightsRole" and a device called "Lights" with the operations "On" and "Off". The relationship between "LightsRole" and "Lights" is established in the PDRA.

Device Role (DR)	HVAC, LightsRole
Device (D)	Heating, Ventilation, Air Conditioning, Lights
Operation (Op)	On, Off
Permission (P)	(Heating, On), (Heating, Off), (Ventilation, On), (Ventilation, Off), (Air Conditioning, On), (Air Conditioning, Off), (Lights, on), (Lights, off)
Permission Device Role Assignment (PDRA)	((Heating, On), HVAC), ((Heating, Off), HVAC) ((Ventilation, On), HVAC), ((Ventilation, Off), HVAC) ((Air Conditioning, On), HVAC), ((Air Conditioning, Off), HVAC) ((Lights, on), LightsRole), ((Lights, off), LightsRole)

Table 3 Example data to present Devices, Operations, Permissions and Device Roles for the democratic model

Role Device Role Assignment

The RDRA (Role to Device Role Assignment) establishes the relationship between devices and device roles. Through this assignment, each role gains access to all the device roles that have been assigned to it.

Table 4 provides additional examples illustrating the assignment between roles and device roles. We utilize the previously presented roles and device roles and establish their connection through the Role-Device Role Assignment (RDRA) mechanism. We link here only the HVAC to the Resident Role and LightsRole to the SharedLights Role.

Role	Resident, Visitor, SharedLights
Device Role	HVAC, LightsRole
Role-Device Role Assignment (RDRA)	(Resident, HVAC), (SharedLights, LightsRole)

Table 4 Example data to present Role-Device Role Assigment for the democratic model

Role Heritage

Role Heritage establishes a many-to-many relationship between roles, enabling the assignment of roles as parents and children (or seniors and juniors in NIST's terminology). This relationship is represented by a set of pairs, where each pair consists of a parent role and its corresponding child role. It is essential to maintain the acyclic nature of the Role Heritage, ensuring that no child role can become a parent role to avoid the formation of cyclic dependencies. The terms "child" and "parent" are commonly used to describe familial relationships and inheritance, but they can also be applied to represent non-familial relationships of subordination or hierarchical structures.

We provide an example of a non-familial relationship in Table 5. In this example, we observe that the "Resident" role acts as the "parent" of the "Visitor" role, and the "Visitor" role acts as the "parent" of the "SharedLights" role. As a result of this hierarchical inheritance, the "Visitor" role inherits the permissions of the "SharedLights" role, and the "Resident" role inherits both the permissions of the "Visitor" and "SharedLights" roles.

Role	Resident, Visitor, SharedLights
Role Heritage	(Resident, Visitor), (Visitor, SharedLights)

Table 5 Example data to present Role Heritage for the democratic model

Core administrative Components

We introduce now the administrative layer, which are all the administrative components of the system. This administrative layer is the novel approach of the master thesis.

Our model is composed of several key components, including Decision Makers (DM), Decision Unit (DU), Decision Task (DT), and Decision Maker Role Assignment (DMRA).

The Decision Task (DT) in the administrative layer is responsible for managing a specific set of Roles and Device Roles. It also stores the corresponding governance policies for each administrative action such as Role and Device Role assignment or revocation. These governance policies play a crucial role in the decision-making process by ensuring that the appropriate thresholds are met for each action.

The Decision Maker (DM) represents a collective group of roles responsible for jointly managing one or multiple decision tasks. Each DM has also governance policies for the management of the DM itself.

The Decision Maker Role Assignment (DMRA) is the relationship that associates the roles with the corresponding decision maker.

The Decision Unit (DU) serves as an abstraction that represents a distinct decision unit within the system. It encompasses the management scope of the decision makers associated with it. Each decision unit comprises two essential elements: a uniquely associated decision maker and a subset of potential authorization assignments referred to as decision tasks. Within a decision unit, any decision maker is authorized to initiate decision processes through a vote of an action to manage any of the authorization assignments included in its corresponding decision task.

The *Decision Maker to Decision Tasks Assignment* (DMDTA) refers to the one-to-many relationships established to assign decision tasks to decision makers and defining the extent of control for each decision maker within a decision unit.

Similarly, the DMDUA (Decision Maker to Decision Unit Assignment) relation exists, also following a one-to-many relation, ensuring that only one decision maker can be assigned to a specific decision unit. Consequently, both the decision task and decision unit are uniquely associated with a decision maker.

In Table 6, we provide an example of data that can be represented using the concepts discussed earlier. The scenario presented here depicts a situation where the Resident group holds administrative power over the other roles and the HVAC appliance. Additionally, the lights are governed by both Visitors and Residents.

In this scenario, the Roles Resident, Visitor, and SharedLights are administered by a Decision Task named DT_group. The Device Role HVAC is administered by the Decision Task DT_appliances, while the lighting is administered by DT_common. The decision task DT_group and DT_appliances are governed by a Decision Maker named DM_resident, while DT_common is governed by DM_common.

DM_resident is governed solely by the Resident role, while DT_common is governed by both the Resident and Visitor roles. Additionally, DM_resident and DT_common have their respective Decision Units, namely DT_resident and DU_common.

Within the system, there exist distinct levels of decision-making authority. Specifically, the administration of lights involves a shared responsibility between the Visitor and Resident roles. Conversely, the administration of roles and the HVAC device is exclusively under the control of the Resident group.

Role	Resident, Visitor, SharedLights
Role Heritage	(Resident, Visitor), (Visitor, SharedLights)
Device Role	HVAC, LightsRole
Decision Task (DT)	DT_group, DT_common, DT_appliances
Decision Task Assignment	(DT_group, Resident), (DT_group, Visitor), (DT_group, SharedLights), (DT_common, LightsRole), (DT_appliances, HVAC)
Decision Maker	DM_resident, DM_common
Decision Maker to Decision Tasks Assignment (DMDTA)	(DM_resident, DT_resident), (DM_resident, DT_ appliances), (DM_common, DT_common)
Decision Unit	DU_resident, DU_common
Decision Maker to Decision Unit Assignment (DMDUA)	(DM_resident, DU_resident) (DM_common, DU_common

Table 6 Example data to present Core Administrative Components for the democratic model

Derived Relations

The Derived Decision Relations consist of a set of helpful functions designed to retrieve and establish decision-making relationships within the model. These functions serve as valuable tools and will be utilized throughout the implementation of the model. Rather than presenting these functions in a lengthy and difficult-to-read format, we have organized them as a concise table (see Table 7) for clarity and ease of understanding.

$User_{r \in R}$	Users in a specified Role
$Role_{dt \in DT}$	Roles in a specified Decision Task
$Role_{dm \in DM}$	Roles in a specified Decision Maker
$DT_{r \in R}$	Decision Task in a specified Role
$DT_{dr \in DR}$	Decision Task in a specified Device Role
$isChild_{child \in R,\ parent \in R}$	Boolean denoting if a child is a child of another role (through inheritance)
$CheckDR_{u \in U,\ dr \in DR}$	Boolean denoting if user has power to manage a specific Device Role
$CheckR_{u \in U,\ r \in R}$	Boolean denoting if user has power to manage a specific Role
$CheckDT_{u \in U,\ dt \in Dt}$	Boolean denoting if user has power to manage a specific Decision Task

Table 7 Derived Relation from democratic model

Decision

Decision actions are a vital component of each Decision Task (DT) and Decision Maker (DM). These actions are assignment/revocation of roles in a Decision Maker (DMRA), user-to-role assignment/revocation in User Assignment (UA), assignment/revocation of roles in the Role to Device Role Assignment (RDRA), role-to-role assignment/revocation in the Role Hierarchy (RH), and policy editing for all decision functions. Actions facilitate role management, hierarchy establishment, and policy customization.

To streamline the voting process within a decision task, three helper functions have been introduced. The first function, $\mathsf{DecisionDT}_{dr \in DR}$, handles the voting process for device roles. The second function, $\mathsf{DecisionDT}_{r \in R}$, is designed for regular roles. Lastly, the third function, $\mathsf{DecisionDT}_{dm \in DM}$, specifically

caters to decision makers. These functions facilitate the understanding of the voting process within the decision task.

Actions

Now, let's explore the potential actions within the model. Each action is accompanied by its specific policy, which is defined within the decision task or decision maker. Among these actions, there are four pairs of assignment and revocation actions related to RDRA, UA, RH, and DMRA. Additionally, one for roles and another for device roles. Lastly, we can edit the associated policies of the decision task or the decision maker using the EditPolicy action.

3.2. Democratic Model Formalization

In this section we will formalize the model. The diagram that this model is Figure 14 on page 23. The formalization of this model is inspired by former a model [13]. The novel part can be seen in Core Components and Actions.

Users, Roles and Sessions

- U, R and S are sets of users, roles and sessions respectively
- $UA \subseteq U \times R$, many-to-many users to role assignment
- $SU \subseteq S \times U$, many-to-one sessions to user relation that assigns each session to a single user who controls the session.
- $SR \subseteq S \times R$, many-to-many session to roles relation that assigns each session to a set of roles that can change under user control, where $(s_i, r_j) \in SR \Rightarrow (\exists u_k \in U) \mid (s_i, u_k) \in SU \wedge (u_k, r_j) \in UA]$; by definition of SU, u_k must be unique

Devices, Operations, Permissions and Device Roles

- D, OP, P and DR are sets of devices, operations, permissions and device roles respectively
- $P \subseteq D \times OP$, every permission is a device, operation pair (device manufacturer specified)
- $PDRA \subseteq P \times DR$, a many-to-many permissions to device roles assignment

Role Device Role Assignment

- $RDRA \subseteq R \times DR$, many-to-many roles to device roles assignment (specified by the corresponding DT)

Role Heritage

- RH $\subseteq$ R $\times$ R, many-to-many roles to roles assignment. It allows a parent role to inherit all the right of the child role. The heritage system is acyclic; thus, no role may become his own child.

Core Components

- DM, DU and DT are sets of decision makers, decision units and decision task respectively.
- DMDTA $\subseteq$ DM $\times$ DT, a one-to-many decision makers to decision tasks relationship.
- DMRA $\subseteq$ DM $\times$ R, a many-to-many decision maker to role assignment relationship.
- DTRA $\subseteq$ DT $\times$ R, a one-to-many decision task to role assignment relationship.
- DTDRA $\subseteq$ DT $\times$ DR, a one-to-many decision task to device role assignment relationship.

- DMDUA $\subseteq$ DM $\times$ DU, a one-to-one decision maker to decision unit assignment relationship.

Derived Relations

- $\text{User}_{r \in R} \subseteq 2^U$ determines which user are included in a specific role.
- $\text{Role}_{dt \in DT} \subseteq 2^R$ determines which role are included in a specific decision task.
- $\text{Role}_{dm \in DM} \subseteq 2^R$ determines which role are included in a specific decision maker.
- $DT_{r \in R} \subset R \times DT$: $DT_r = dt \in DT$: $r \in DTRA(r)$: determines which dt can manage the r.
- $DT_{dr \in DR} \subset DR \times DT$: $DT_{dr} = dt \in DT$: $dr \in DTDRA(dr)$: determines which dt can manage the dr.
- $DM_{dt \in DT} \subset DT \times DM$: $DM_{dt} = dm \in DM$: $dt \in DMDTA(dm)$: determines which dm can manage the dt.
- $\text{isChild}_{child \in R,\ parent \in R} = (parent, child) \in RH$: determines if a child has a corresponding parent.
- $\text{CheckDR}_{u \in U,\ dr \in DR}$: determines if the user u has decision rights on Device Role dr.
- $\text{CheckR}_{u \in U,\ r \in R}$: determines if the user u has decision rights on Role r.
- $\text{CheckDM}_{u \in U,\ dm \in DM}$: determines if the user u has decision rights on Device Maker dm.

Decision

- $\text{DecisionDT}_{dr \in DR}$: voting process in the corresponding dt for dr.
- $\text{DecisionDT}_{r \in R}$: voting process in the corresponding dt for r.
- $\text{DecisionDT}_{dm \in DM}$: voting process in the corresponding dt for dm.

Actions

For all action the model verify if user that initiate a decision is member of the corresponding decision making group. After that we go over all the different possible actions that could happen.

Firstly, the actions of assigning or revoking Role-Device Role Assignments involve first checking if the assignment already exists or not, and then adding or removing the Role - Device Role pair from the existing assignments.

- $\text{AssignRDRA}(u \in U, r \in R,\ dr \in DR) \equiv$
  $\text{CheckDR}_{u,dr} \wedge (r,dr) \notin RDRA \wedge \text{DecisionDT}_{dr} \Rightarrow RDRA' = RDRA \cup (r,dr)$
- $\text{RevokeRDRA}(u \in U, r \in R,\ dr \in DR) \equiv$
  $\text{CheckDR}_{u,dr} \wedge (r,dr) \in RDRA \wedge \text{DecisionDT}_{dr} \Rightarrow RDRA' = RDRA \setminus (r,dr)$

Secondly, the actions of assigning or revoking User–Role Assignment involve adding or removing a user–role pair from the existing User-role Assignment. By checking if the assignment already exists or not.

- $\text{AssignUA}(u \in U, u_2 \in U, r \in R) \equiv$
  $\text{CheckR}_{u,r} \wedge (u_2,r) \notin UA \wedge \text{DecisionDT}_r \Rightarrow UA' = UA \cup (u_2,r)$
- $\text{RevokeUA}(u \in U, u_2 \in U, r \in R) \equiv$
  $\text{CheckR}_{u,r} \wedge (u_2,r) \in UA \wedge \text{DecisionDT}_r \Rightarrow UA' = UA \setminus (u_2,r)$

Thirdly, the actions of assigning or revoking Role-Role assignments, also known as Role Heritage, involve adding or removing a role-role pair from the existing Role Heritage. Before making the assignment, it is necessary to check if the assignment already exists and to ensure that the future child role does not become its own parent, thus avoiding any potential loops in the hierarchy.

- AssignRH(u ∈ U, child ∈ R, parent ∈ R) ≡
  $\text{CheckR}_{\text{u,child}} \wedge \neg\text{isChild}(\text{parent}, \text{child}) \wedge \text{parent} \neg= \text{child} \wedge \text{DecisionDT}_{\text{r}}$
  $\Rightarrow \text{RH}' = \text{RH} \cup (\text{parent,child})$
- RevokeRH(u ∈ U, child ∈ R, parent ∈ R) ≡
  $\text{CheckR}_{\text{u,child}} \wedge \text{isChild}(\text{parent}, \text{child}) \wedge \text{DecisionDT}_{\text{r}}$
  $\Rightarrow \text{RH}' = \text{RH} \setminus (\text{parent}, \text{child})$

Forth, the actions of assigning or revoking Decision Maker–Role assignments

- AssignDMRA(u ∈ U, $r \in R$, dm ∈ DM) ≡ $\text{CheckDM}_{\text{u,dm}} \wedge (\text{r}, \text{dm}) \notin \text{DMRA} \wedge \text{DecisionDT}_{\text{dm}} \Rightarrow$
  $\text{DMRA}' = \text{DMRA} \cup (\text{r}, \text{dm})$
- RevokeDMRA(u ∈ U, $r \in R$, dm ∈ DM) ≡ $\text{CheckDM}_{\text{u,dm}} \wedge (\text{r}, \text{dm}) \in \text{DMRA} \wedge \text{DecisionDT}_{\text{dm}} \Rightarrow$
  $\text{DMRA}' = \text{DMRA} \setminus (\text{r}, \text{dm})$

Lastly, hereunder functions to edit the governance policy, either for the decision task or the decision maker.

- EditPolicy(u ∈ U, dt ∈ DT, dt_{new} ∈ DT) ≡ $\text{CheckDT}_{\text{u,dt}} \wedge \text{DecisionDT}_{\text{dt}} \Rightarrow$
  $\text{DMDTA}' = \text{DMDTA} \setminus (\text{DM}_{\text{dt}}, \text{dt}) \cup (\text{DM}_{\text{dt}}, \text{dt}_{\text{new}})$
- EditPolicy(u ∈ U, dm ∈ DM, dm_{new} ∈ DM) ≡ $\text{CheckDM}_{\text{u,dm}} \wedge \text{DecisionDM}_{\text{dm}} \Rightarrow \text{dm} = \text{dm}_{\text{new}}$

3.3. Use-cases

Let us analyze the use cases presented in the introduction where the democratic aspect is essential. These examples have been carefully selected to showcase situations where decision-making plays a crucial role. We begin by presenting each scenario and then demonstrate how it can be mapped onto the model. Following that, we discuss for each scenario the democratic system why they are interesting.

3.3.1. Use case 1

In the first scenario, we encounter a shared living space equipped with a smart speaker. To ensure a peaceful environment, the residents collectively decided that if the sound from the speaker becomes disruptive, each resident should have the authority to lower the volume to 50%. This control allows for immediate action to mitigate disturbances and maintain a harmonious atmosphere.

We can now incorporate the scenario into our model. In the first example, we will thoroughly explain each step to ensure a comprehensive understanding of the model.

In this scenario, we can consider a co-living environment with two residents, referred to as Resident1 and Resident2. Both Resident1 and Resident2 are members of the Resident group, as established by the User-Role Assignment. Two residents is the minimum number required to make this example relevant, but the model can accommodate a larger number of residents if needed. It is written under table form in Table 8.

User (U)	Resident1, Resident2
Role (R)	Resident
User-Role Assignment (UA)	(Resident1, resident), (Resident2, resident)

Table 8: User, Roles and UA depicted depicted for example 1 in Democratic Formal Model

Additionally, there is a speaker. Two device role are linked to the speaker one with permission of 50% volume named DG_audio and a second one name DG_audio_max with a 100% volume permission. It is written under table form in Table 8.

Device	Speaker
Operation	50%, 100%
Permission	(Speaker, 50%), (Speaker, 100%)
Device Role	DG_audio, DG_audio_max
Permission-Device Role Assignement (PDRA)	(DG_audio, (Speaker, 50%)), (DG_audio_max, (Speaker, 100%))

Table 9 Device, Operation, Permission, Device Role and PDRA depicted for example 1 in Democratic Formal Model

The resident group is self-managed through a decision task called DT_groups. This decision task is responsible for holding the governance policies related to the resident group. These policies include the policies for User-Role Assignment (AssignUA) and revocation (RevokeUA). Furthermore, the decision task incorporates governance policies for Role Heritage assignment (AssignRH) and revocation (RevokeRH). Lastly the decision group has also a policy for editPolicy.

We define certain values for the policies and explain their meanings. For the User-Role Assignment (UA) in this group, the policy is set to unanimity (or 100%), which means that all members of the decision task must agree in order to assign a new user to the role of Resident. On the other hand, the revoking policy is set to 75%, indicating that 75% of the decision makers in the group must agree to revoke the membership of a user in the Resident role. In our example with two residents, both residents must agree to remove a user from the Resident role. However, if there are five residents, the agreement of four residents would be sufficient to revoke the membership of the fifth user in the role. The same reasoning applies to the Role Heritage in this group. The policy for assigning a role heritage is unanimity, meaning that all members of the decision task must agree to assign a role as a parent to another role. On the other hand, the policy for revoking a role heritage is set to 75%, indicating that 75% of the decision makers in the group must agree to revoke the role heritage.

Additionally, we have the editPolicy, which allows the owners of the DT_group to modify the previously defined policies. For the editPolicy, we have chosen a unanimity policy as well, meaning that all members of the decision task must agree in order to make changes to the governance policies. It is written under table form in Table 8.

Role	Resident
Decision Task (DT)	DT_groups – AssignUA = 100% – RevokeUA = 75% – AssignRH = 100% – RevokeRH = 75% – editPolicy = 100%
Decision Task – Role Assignment (DTRA)	(DT_groups, Resident)

Table 10 Decision task for a role depicted for example 1 in Democratic Formal Model

We apply the same logic to define the decision groups DT_audio and DT_audio_max, which manage the DM_audio and DM_audio_max, respectively. We also need to define their corresponding governance policies for role-device role assignments (RDRA) and the editPolicy.

For both DT_audio and DT_audio_max, the governance policies for RDRA and editPolicy are set to unanimity. This means that all members of the decision task must agree on assigning a role to a device role and on making changes to the governance policies.

However, there is a difference in the RevokeRDRA policy for DT_audio_max. It is set to 1%, which is a small percentage. This means that in a residence group with fewer than 100 members, each resident has the ability to remove the RDRA assignment. This allows each user to easily revoke the assignment and shut down loud music, ensuring a comfortable living environment. Shown as table in Table 11.

It is important to note that in our model, administrators can easily handle the assignment themselves because our system relies on the decision-making process, where decisions are made collectively by the decision makers.

Device Role	DM_audio, DM_audio_max
Decision Task (DT)	DT_audio – AssignRDRA = 100% – RevokeRDRA = 100% – editPolicy = 100% DT_audio_max – AssignRDRA = 100% – RevokeRDRA = 1% – editPolicy = 100%
Decision Task – Device Role Assignment (DTDRA)	(DT_ audio, DR_audio), (DT_ audio_max, DR_audi_max)

Table 11 Decision task for a device role depicted for example 1 in Democratic Formal Model

Now, to complete the model, we need to include the decision maker DM_residence, which is governed by the residents themselves. DM_residence oversees the decision tasks DT_audio and DT_audio_max. In addition, DM_residence has its own governance policy for decision maker role assignment and revocation, both set at 100%. Shown as table in Table 12.

Setting the governance policies for decision maker role assignment and revocation to 100% means that all residents should agree add or remove a role from the decision maker group This ensures a fair and inclusive decision-making process within the residence group.

Decision Task	DT_audio, DT_audio_max, DT_groups
Decision maker	DM_residence – AssignDMRA = 100% – RevokeDMRA = 100% – editPolicy = 100%
Role	Resident
Decision Maker – Decision Task Assignment (DMDTA)	(DM_residence, DT_audio), (DM_residence, DT_audio_max), (DM_residence, DT_groups)
Decision Maker Role Assignement (DMRA)	(DM_residence, Resident)

Table 12 Decision Maker relations depicted for example 1 in Democratic Formal Model

Now that we have defined all the relationships for our model, we can compile all the fields to create Table 13, which represents a comprehensive overview of the model.

Users={Resident1, Resident2}
Roles={resident}
UA={(Resident1,resident),(Resident2,resident)}
RH={}
Device={Speaker}
Operation={50%, 100%}
Permission= {(Speaker, 50%), (Speaker, 100%)}
DR={ DG_audio, DG_audio_max }
PDRA={(DG_audio, (Speaker, 50%)), (DG_audio_max, (Speaker, 100%))}
RDRA={(resident, DG_audio), (resident, DG_audio_max)}
DM={DM_residence}
DMRA={(resident, DM)}
DT={DT_audio, DT_audio_max, DT_groups}
DTRA={(DT_groups, resident)}
DTDRA={(DT_audio, DG_audio), (DT_audio_max, DG_audio_max)}
DMDTA={(DM_residence, DT_audio), (DT_audio_max, DT_groups)}

Table 13 Example 1 transposition to theorical model without the governance policies. The assignement in red will be revoked by the decision process

In this first example, Resident1 has the authority to revoke the Role-Device Role Assignment (RDRA) for the maximum volume of the speaker. Because the governance policy threshold is very low for that specific case. Vote has directly a positive exist.

Decision = (action=RevokeRDRA, DG= DG_audio_max, from=Resident1, to=resident)

Table 14 Example 1 Decision of revoke permission to resident role

With the authority to define the governance policy, decision-makers have the power to remove a role, including their own, from the RDRA (Role-Based Data Access) system. This ability to modify roles enables interesting decision-making possibilities. By distinguishing between shared ownership and access rights, decision-makers can make choices that align with specific requirements and circumstances, allowing for more flexibility and adaptability in the decision-making process.

3.3.2. Use case 2

In the second use case, we revisit the smart speaker setting. Here, all visitors are granted access to the smart speaker but are limited to a maximum volume of 50%. However, there may be situations where a resident desires to grant visitors the privilege of accessing the speaker at 100% volume. To address this scenario, our model enables residents to initiate a decision process to collectively determine whether to authorize this exceptional access, ensuring transparency and fairness in the decision-making process.

In this instance, we will be more concise compared to the first example. We will directly go to the final transposal of the second example to the theorical model.

In this example (see Table 15) we introduce a new user called Visitor1 and a new role named Visitor. Visitor1 is assigned to the Visitor role. The Visitor role already has access to DG_audio. To grant access rights to DG_audio_max to Visitor1, we have four possible approaches:

1. The Visitor role becomes the parent of the Resident role.
2. DM_audio_max is directly assigned to the Visitor role.
3. A new group is created, and Visitor_1 and DM_audio_max are assigned to it.

4. Visitor_1 is assign the role of Resident, which itself contains the access right to the DG_audio_max

All four options can be perform, we choose here the second option, where DM_audio_max is directly assigned to the Visitor role. To do so the members of the DM responsible for the Device Group should agree to AssignRDRA under its respective governance policy. In our case the threshold is 100%, thus all members of the resident group should vote positively to grant access to DG_audio_max for the Visitor role.

A concise view of the example in the theorical model is noted in Table 15.

Users={Resident1, Resident2, Visitor1} Roles={Resident, Visitor} UA={(Resident1,resident),(Resident2,resident), (Visitor1, Visitor)} RH={} Device={Speaker} Operation={50%, 100%} Permission= {(Speaker, 50%), (Speaker, 100%)} DR={ DG_audio, DG_audio_max } PDRA={(DG_audio, (Speaker, 50%)), (DG_audio_max, (Speaker, 100%))} RDRA={(Resident, DG_audio), (Resident, DG_audio_max), (Visitor, DG_audio), (Visitor, DG_audio_max)} DM={DM_residence} DMRA={(Resident, DM)} DT={DT_audio, DT_audio_max, DT_groups} DTRA={(DT_groups, Resident), (DT_groups, Visitor)} DTDRA={(DT_audio, DG_audio), (DT_audio_max, DG_audio_max)} DMDTA={(DM_residence, DT_audio), (DT_audio_max, DT_groups)}

Table 15 Example 2 transposition to theorical model without the governance policies. In red allow a user obtain access to a permission

To facilitate the sharing of access rights, the decision-makers responsible for the DG_audio_max task, in this case, the Residents, need to make a collective decision. Since the governance policy is set at 100%, it means that all members of the Residents group must reach a consensus. Agreement from each member is necessary for the decision to be implemented. It is depicted under

Decision = (action=AssignRDRA, DG=DG_audio_max, from=Resident1 and Resident2, to=Visitor)

Table 16 Decision needed in example 2 to give Visitor1 access to DG_audio_max

Making a decision with multiple users sharing decision-making power enables collaboration among the members. The sharing decision-making power approach not only promotes collaboration but also serves as a safeguard against making mistakes.

3.3.3. Use case 3

Two owner wants to add a new member, Visitor1, to their decision makers' group for managing the TV. It would give to opportunity to add the visitor in the decision if whether or not the Child may have access to the TV. To express this we create Table 17. The example can really be simplified to the table X.

Users={Owner1, Owner2, Visitor1, Child1}
Roles={Owner, Visitor, Child}
UA={(Owner1, Owner),(Owner2, Owner), (Visitor1, Visitor), (Child1, Child)}
RH={}
Device={TV}
Operation={on, off}
Permission= {(TV, on), (TV, off)}
DR={ DG_TV }
PDRA={(DG_TV, (TV, on)), (DG_TV, (TV, off))}
RDRA={(Child, DG_TV)}
DM={DM_TV, DM_roles}
DMRA={(Owner, DM_groups), (Owner, DM_TV), (Visitor, DM_ TV)}
DT={DT_TV, DT_groups}
DTRA={(DT_groups, Owner), (DT_groups, Visitor), (DT_groups, Child)}
DTDRA={(DT_TV, DG_TV)}
DMDTA={(DM_roles, DT_groups), (DM_TV, DT_TV)}

Table 17 Example 3 transposition to theorical model without the goverance policies. In red, the new membership of visitor to the TV demcion maker group

That decision shall be taken by the DM_TV owners under the AssignDMRA governance policy. It will add the Visitor role to the DM_TV. It will allow the active participation of the Visitor role in the decision making process of the corresponding decision task (here DT_TV with DG_TV as device role). In Table 18 we find such decision.

Decision = (action= AssignDMRA, DM=DM_TV, from=Owner1 and Owner2, to=Visitor)

Table 18 Decision needed in example 2 to give Visitor1 membership to the corresponding DM

By having a new role in the Decision Making process for the TV it changes the dynamics Because now the Visitor has the power to initiate a decision or reject someone else's decision.

3.4. Summary of model

In this chapter, we presented a formalized model that addresses the limitations of existing models and demonstrated its capability through various use cases. By formalizing the model, we provided a clear structure and framework for managing smart home access control in shared living spaces. The use cases highlighted the flexibility and effectiveness of our model in handling complex scenarios that were not easily achievable in other models. Our approach opens up new possibilities for performing tasks that were previously deemed impossible, showcasing the practicality and advancements of our model in the field of smart home access control.

4. Proof of concept (PoC)

The objective of the proof of concept is to demonstrate the effectiveness of the previously defined model on a real system. To showcase this model, we have selected Home Assistant, the widely used open-source smart hub system. Initially, we will provide an explanation for choosing Home Assistant, followed by an overview of its internal workings. Subsequently, we will outline the necessary modifications that were made to incorporate the model. Finally, we will illustrate how the examples presented in the introduction are implemented in this proof of concept.

4.1. Why Home Assistant?

For our proof of concept, we needed to select an open-source smart hub platform. The main objective is to improve the authorization process. To assist us in the decision of the smart hub, we consulted a paper by *Brian Setz et al.* [16], which provided valuable insights on platform maturity and project support.

The author highlighted four leading platforms: Home Assistant [18], Domoticz [19], openHAB [20], and ioBroker [21]. However, we quickly ruled out Domoticz and ioBroker because we weren't familiar with their programming languages, C++ and JavaScript. This left us with two options: Home Assistant (Python) and OpenHAB (Java).

Considering the focus on role-based access control (RBAC) in the development of the system, it was crucial to select a platform that could effectively accommodate this feature. While there have been some efforts by another student, *Nicolas Gennart* [22], last year to implement RBAC in OpenHAB, it is still not been fully functional. On the other hand, Home Assistant has emerged as a well-maintained and widely adopted project. Home Assistant is currently using a DAC system and groups were introduced by a maintainer in 2018 [23].

As the two system are suited for the developed model in this thesis, we look for other criteria.

Both projects are developed on GitHub where some statistics are public. As of May 2023 Home Assistant collected 23.1k forks and more than 60.2k likes, in comparison OpenHAB had 379 forks and 770 likes. It seems that Home Assistant is 100 times more popular than OpenHAB.

Home Assistant is written in Python and OpenHAB in Java. My first hands experiences were more promising in Python than in Java. The project's structure was clearer in Python than Java. The Python project had a better documentation and more active members.

Therefore, we ultimately chose Home Assistant as the platform for implementing the RBAC model.

4.2. How does Home Assistant work internally?

By understanding how Home Assistant works internally, it will become easier to understand the changes that had to be made to implement the model.

Before delving into the internal workings of Home Assistant, it is crucial to establish clear definitions in a glossary to avoid any confusion.

When we refer to Home Assistant, we specifically mean the Core, which forms the foundation for all other associated projects. In this section, we will explore the inner mechanisms (architecture) of Home Assistant, providing insights into its underlying logic blocks and their utilization throughout the system.

4.2.1. Glossary

- A Device refers to a tangible object.
- A Component is a "virtual building-block" to create virtual model of all the device. We can mention examples such as *Switch*, *Lights*, S*ensor*. Complex devices, such as smart thermostats, can use multiple components to be represented.
- **Integration** connects a device with one or more components.
- An **Entity** refers to an individual representation of a component.
- A Platform serve as association form entities. If a light bulb uses the Philips Hue integration, the created entities are said to be on the Philips Hue Platform.

Hereunder, we will provide an example that demonstrates the practical application of the terms we have introduced.

Please refer to the diagram below (Figure 15). Let's imagine we have a smart light bulb (device) manufactured by Phillips Hue. Phillips Hue offers an integration specifically for their products. This integration utilizes two components: the Light Component and the Sensor Component. As a result, the smart light bulb will have a Light entity and a Sensor entity associated with it. Both of these entities belong to the Phillips Hue platform, which provides the necessary functionality and features for seamless integration within the Home Assistant system.

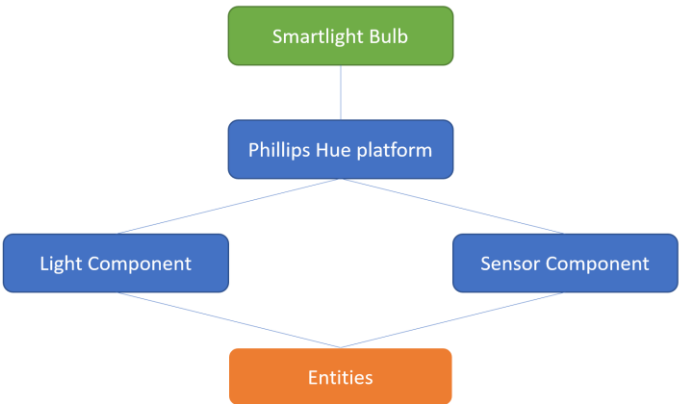

Figure 15: Glossary example

4.2.2. Architecture

The Home Assistant Core is designed around an Event Bus. It is the beating heart of Home Assistant. The current memory of the system is held in the State Machine, it is a cache version of memory that Home Assistant use to keep track of all entities states. An internal clock called Timer sends an event to the Event Bus each second. Finally, the Service Registry holds a list of all services published, where a service allows to change the internal state of an entity and inter components communication.

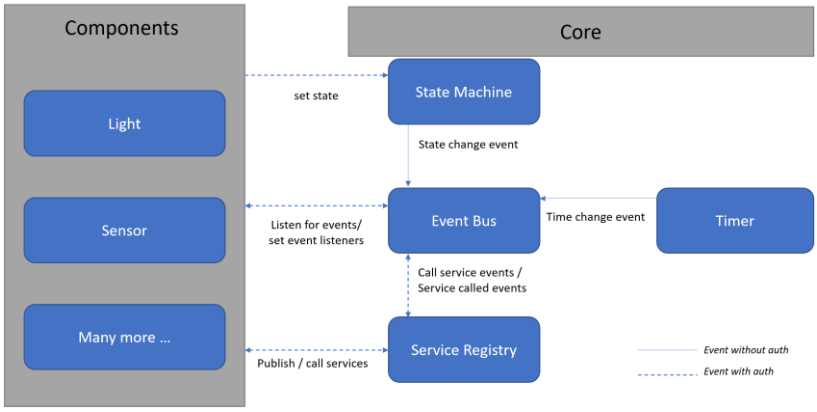

Figure 16: Home Assistant Architecture inspired from [4]

It is important to note that Home Assistant does not differentiate between Authentication and Authorization. Authentication verifies the identity of a user attempting to connect to the system, while Authorization determines whether the authenticated user is granted access to specific resources. Both are referred to as Auth [24].

Furthermore there are two elements in Home Assistant's Auth that communicate with each other. we will distinct those two elements as CoreAuth and ComponentAuth. But Home Assistant does not make that distinction.

Within CoreAuth, the storage of authentication information and the execution of authorization processes are addressed. ComponentAuth makes the interaction between the outside world and CoreAuth through a WebSocket, which enables a real-time, bidirectional connection between a client and a server. WebSocket allows seamless subscription to changes, facilitating efficient communication between users and the server. WebSocket is widespread, as highlighted by documentation provided by Mozilla [25].

In this thesis the focus is thus on authorization. All authorization and authentication information are stored in an JSON file. The JSON file is structured into four main sections: users, groups, credentials, and tokens. These sections collectively encompass the necessary information for effective authorization and authentication processes.

At present, in Home Assistant, the user system operates on a group-based structure, wherein users are assigned to specific groups that possess corresponding permissions. These permissions can be categorized based on areas (groups of entities organized by location), domains (groups of entities organized by components), entity IDs (specific entities) or all entities collectively. Each permission category consists of three operations: read, control, and edit. The read operation grants users access to view the state of entities, the control operation enables users to modify the state of entities, and the edit operation permits users to modify the definition or configuration of entities.

In RBAC, as mention by *Prof. Ravi Sandhu* [26] , a group refers to a collection of users who are assigned a specific set of permissions. These permissions are associated with the group and are inherited by the users who are members of that group. Essentially, a role can be seen as a collection of permissions that are granted to users when they assume that role. But in Home Assistant the term "group" is used to denote "groups" and "roles" interchangeably. we will therefore use the term "group" as well.

4.3. Changes made to Home Assistant

The aim of the PoC is to proof that the model can be implemented is real-world systems. To do so, some modifications will be made to Home Assistant.
The theorical model developed in this master thesis can be found in Figure 14 on page 23 to better understand the upcoming changes in Home Assistant current Access Control. After an analysis of a current access control, role heritage will be introduced and afterwards administration by decision.

4.3.1. Current Control Access in Home Assistant

Currently, in Home Assistant (depicted in Figure 17), there are users who can be members of groups. These groups are assigned access authorizations by one of them, the admin group, which is the only one to have the right to assign groups to users and modify their associated authorizations.

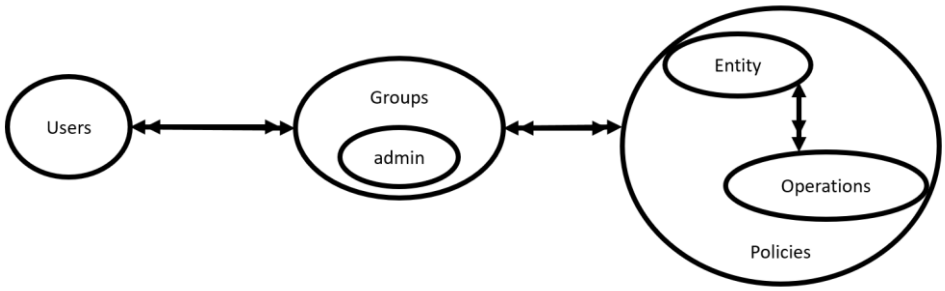

Figure 17: Current Access Control in Home Assistant

For a user, there is a 128-bit long unique identifier (id) represented as a string. A user can be member of a list of groups. In Home Assistant they are assigned a name and several Booleans values indicating if they are an owner, active, system-generated, or only allowed local access. These last information are not relevant for the model, but are shown for completeness.

A group also has an id represented as a string and a name, like a user. Additionally, a group possesses a set of policies. For clearer further explanation only single entity and its corresponding id (i.e. light.light1) will be taken into account and not set entities such as aeras, domains and all entities collectively.

An example using JSON for a user and a group will be as follow:

```
"users":[{
    "id": "46e84fae840d4beb81013e6c6231541b",
    "group_ids": [],
    "is_owner": false,
    "is_active": true,
    "name": "userA",
    "system_generated": false,
    "local_only": false
}],
```

```
"groups":[{
    "id": "a117f2d6d4ad4904f73cfffb1b97b075",
    "name": "newGroup",
    "policy": {
        "entities": {
          "entity_ids": {
            "light.light1": {
              "read": true
}}}}}]
```

4.3.2. Introduction of role heritage

The initial enhancement made to Home Assistant is the introduction of role heritage. This feature allows a group to reference a list of other groups, where the referencing group is considered the senior group and the groups in the list are the junior groups.

When checking for access rights, the groups are searched recursively.

It is important to note that a junior group cannot become the senior group itself, and the referencing group cannot become a junior of itself.

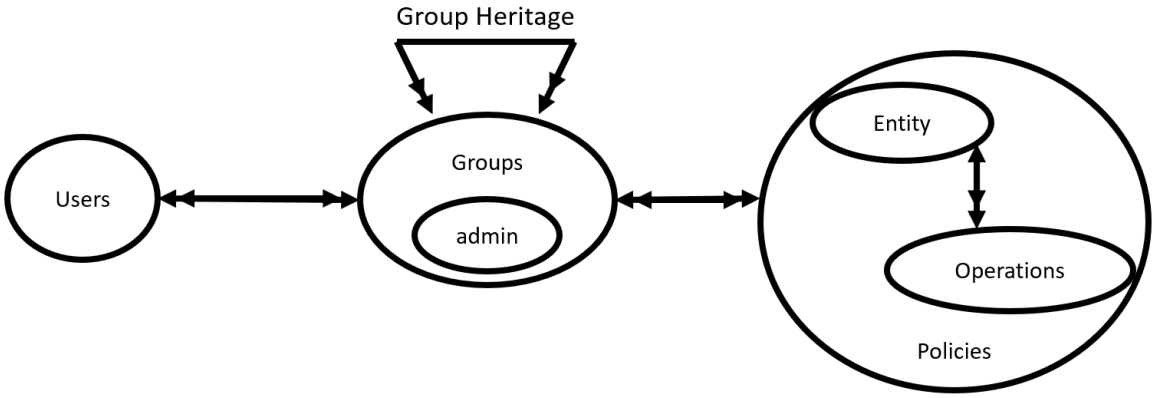

Figure 18: Home Assistant with heritage

To allow heritage we add a field *group_ids* (which is a list) in groups as follow:

```
"groups":[{
    "id": "a117f2d6d4ad4904f73cfffb1b97b075",
    "name": "newGroup",
    "group_ids":[],
    "policy": {
        "entities": {
          "entity_ids": {
            "light.light1": {
              "read": true
}}}}}]
```

4.3.3. Introduction of decision administration

A unique administrator access control system has been replaced by a multi collaborative access control system ruled by decision makers. Because we removed the Admin group, some admin only functions such for groups and policies creation/deletion are given to the users.

The different elements of this democratic system are:

- Device Group (DG): it is distinguished from the other groups with its DG prefix. It is responsible of the policies.
- Decision task group(s) (DT): it is distinguished from the other groups with its DT prefix. It is responsible :
  - of the policies management in Device Group with:
    - Group-Device Group assignment
  - and of the group management in Group with:
    - User-Group assignment
    - Group heritage
- Decision making group(s) (DM): it is distinguished from the other groups with its DM prefix. A decision making group has control over the owner of the decision making group itself. Each decision making group may have control over some decision tasks.

Note that all groups and device group should be manage through one and only one decision task.

Let visualized all that in the Figure 19 below.

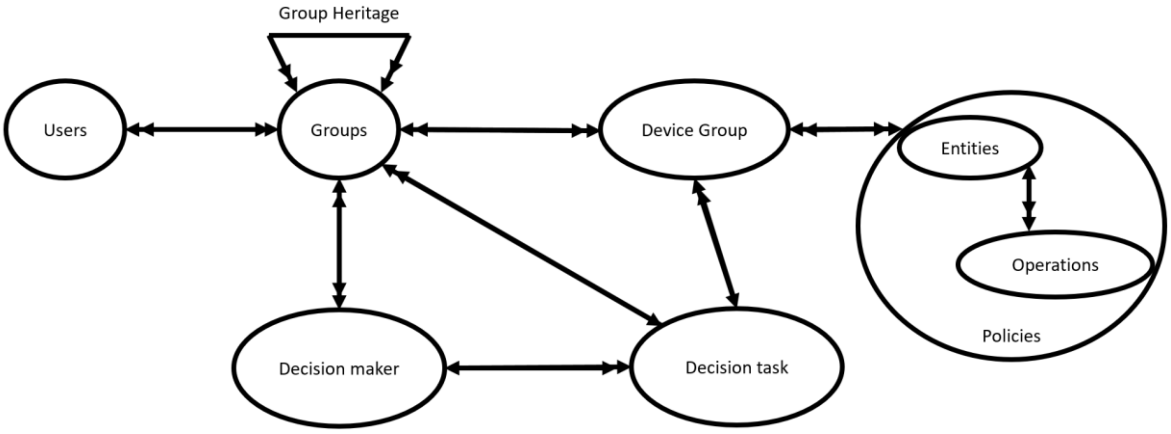

Figure 19: Complete model in Home Assistant

It is translated to JSON code that can be understood by Home Assistant

Firstly, each user can create a group. Each group needs a Decision Maker to manage the group, where for each DT and DM the governance policy can be set.

Secondly if a device is added, it adds a new Device Group (and a new entity). See Figure 20 below.

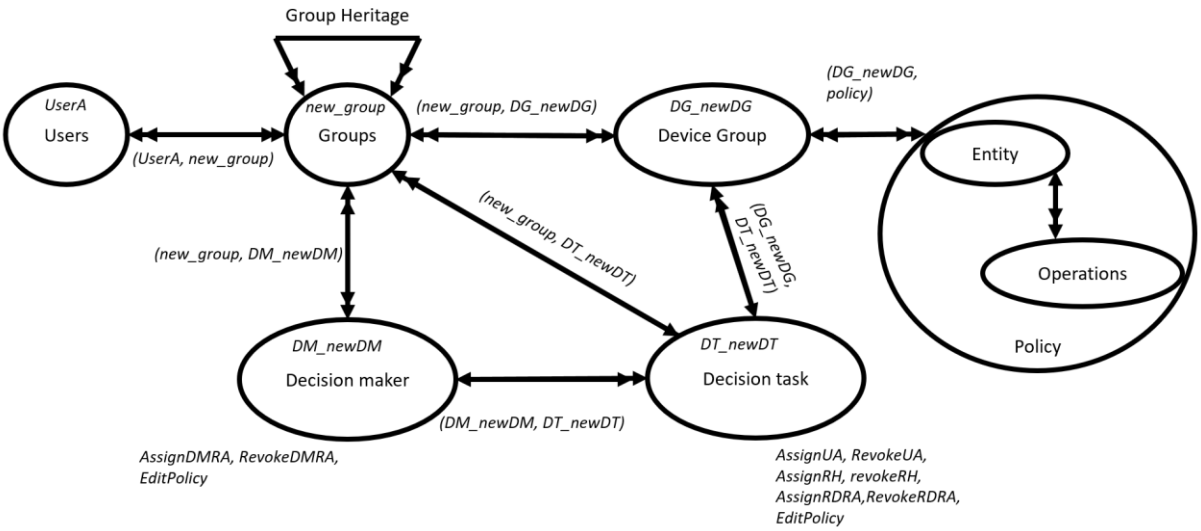

Figure 20: Complete model with dummy values

It is important to note that the "group_id" field serves different purposes. For "Groups," it is used for Group Heritage, where juniors cannot become seniors and determine membership in the decision makers' group. "DM" indicates the connected decision tasks. "DT" is utilized for group management, allowing the definition of Role Heritage or assigning a group to a user and Group-Device Group Assignment. Both decision makers and decision tasks have governance policies. “DG” is utilized for abstracting the policy to group-level.

To accommodate the change from the unique admin to collaborative decision making groups, the JSON structure is modified as follows:

```
"groups":[
{"id": "a117f2d6d4ad4904f73cfffb1b97b075",
"name": "new_group",
"group_ids":["117f2d6d4ad4904f73cfffb1b97b07"]
},

{"id": "117f2d6d4ad4904f73cfffb1b97b07",
"name": "DM_newDM",
"group_ids":["8d0c4c1a937af270b50d4bb58302cc71"],
"policy": { "group": {
    // decision maker-role assignment
    "AssignDMRA": 1.0,
    "RevokeDMRA": 0.75,
    // edit above policies
    "editPolicy": 0.75
}}},

{"id": "8d0c4c1a937af270b50d4bb58302cc71",
"name": "DT_newDT",
"group_ids":["a117f2d6d4ad4904f73cfffb1b97b075"],
"policy": { "group": {
    // user-role assignment
    "AssignUA": 0.5,
    "RevokeUA": 0.5,
    // role hierarchy
    "AssignRH": 1.0,
    "RevokeRH": 0.75,
    // edit above policies
    "editPolicy": 0.75
}}}]
```

Group creation

```
"groups":[
{"id": "ea6acf330ca24c1f1cfafcb692a1c1e1",
"name": "DG_newDG",
"group_ids":[],
"policy": {
    "entities": {
      "entity_ids": {
        "light.light1": {
          "read": true
}}}}},
{"id": "8d0c4c1a937af270b50d4bb58302cc71",
"name": "DT_newDT",
"group_ids":["ea6acf330ca24c1f1cfafcb692a1c1e1"],
"policy": { "group": {
    // role-device role assignment
    "AssignRDRA": 0.5,
    "RevokeRDRA": 0.5,
    // edit above policies
    "editPolicy": 0.75
}}}]
```

Device group creation

We have explained the structure, now we need to show how the actions will be expressed through voting (the voting system has been chosen in the model).

The various elements to be taken into account are a source user, who initiates the vote, and a target (group or user) depending on the action.

The process consists of adding the Group on which the change is to be made and choosing the action to be implemented. The different votes of different users (whether positive or negative) can be expressed. Only users of the corresponding decision-making group are authorized to vote.

At the end, the result of the votes appears, as shown follow:

```
"decision": [
{"source": "userA",
"target": "newGroup",
"group": "DR_group",
"action": "AssignRDRA",
"reject": [],
"approve": ["userC", "userD"],
"result": true
}]
```

The source code of this proof of concept implementation can be found on GitHub at https://github.com/StanislasG/core, specifically in the "democracy" branch. This repository is a fork of the main Home Assistant Core project.

The different features of this section have been implemented and verify through manual testing. The objective of the code was to demonstrate the proof the feasibility of implementing the proposed model on a real-world platform. The changes made were mainly in a few file linked with the authorization or the authentication.

4.4. Examples

In the following discussion, we will present the various examples of the introduction to demonstrate how they can be implemented within Home Assistant. Firstly, we will provide a brief recap of each example, followed by defining the necessary terms within the Home Assistant framework. Next, we will display a diagram highlighting the specific changes, denoted in bold red letters. Finally, we will provide the corresponding JSON files required to accomplish the desired outcome. The examples include user configuration, group configuration (with the identified changes), and the decision-making process necessary for achieving the intended result.

4.4.1. Example 1

In a residential setting, there is an audio system where all residents initially have access to the maximum volume. However, each resident has the ability to revoke the access of the resident group to the maximum volume. To facilitate this, two residents (Resident1 and Resident2) are members of the resident group, which is managed by the decision task DT_groups. DT_groups, in turn, is managed by the decision maker's group DM_residence. DM_residence is managed by the residence group. Additionally, there are two device groups, DG_audio and DG_audio_max, representing the

normal volume and the maximum volume level of the audio system, respectively. Both device groups are managed by the decision task DT_audio, which is itself managed by DM_residence.

Each resident can make a decision to revoke access to DG_audio_max from the resident group. This decision is governed by the RevokeRDRA governance policy, as specified in the code. To ensure that every resident can revoke this access, the RevokeRDRA threshold needed to revoke this access should be set at a very low level so that it is possible for each resident alone to fulfill. In order to restore the access, the residents need to make a decision with the AssignRDRA governance policy, which has a higher threshold.

In Figure 21, all entities and relationships are depicted. Notably, in red bold letters, it is indicated that the assignment should be revoked to prevent the user from having access to the highest volume of the speaker.

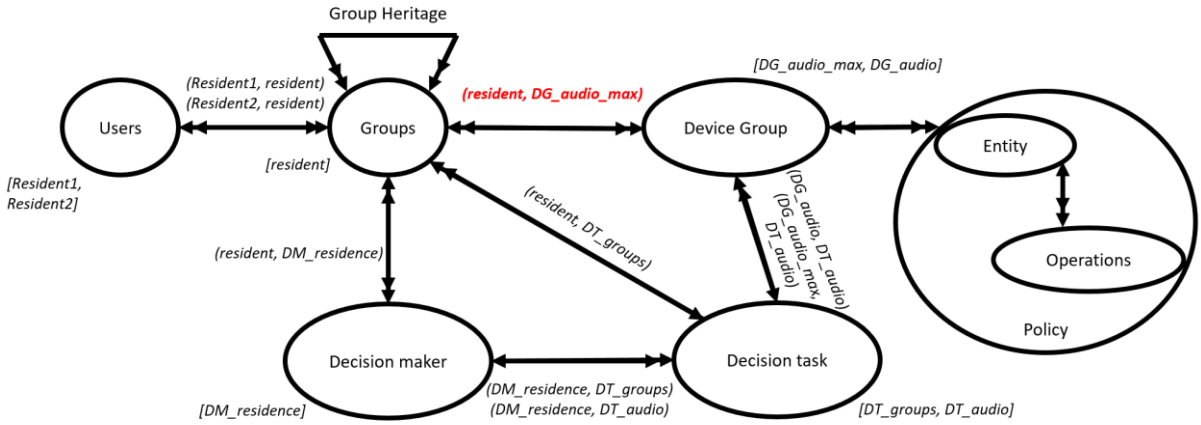

Figure 21: PoC model for example 1 with red bold letters would should be easily revoked

Now we come to the JSON representations, first we see all the two resident user, both have the User-role assignment to the resident role by mentioning the group id in the group_ids field

```
"users":[
{"name": "Resident1", "id": "46e84fae840d4beb81013e6c6231541b",
"group_ids": ["a117f2d6d4ad4904f73cfffb1b97b075"], //resident
"is_owner": false,"system_generated": false,
"is_active": true,"local_only": false},

{"name": "Resident2", "id": "9ed2bbd147abf7c785a693dcdabe38c0",
"group_ids": ["a117f2d6d4ad4904f73cfffb1b97b075"], //resident
"is_owner": false,"system_generated": false,
"is_active": true,"local_only": false},
],
```

Now, in the bulk part, all groups are defined and assigned to each other. The comments indicate the correspondence between the ids and names of each group. It is observed that the device groups (DG) hold permissions to the actual objects, while the decision task and decision maker groups have their managed groups denoted. Additionally, both decision task and decision maker groups have their respective governance policies.

Lastly, the user group is at the top, indicating whether the role has direct access to permissions, role inheritance, or decision maker power. This information is provided through the group ids listed in the "group_id" field. All governance policies are expressed as a float to express a percentage threshold that user should vote to have a positive exit.

```
"groups":[//Group
{"name": "resident", "id": "a117f2d6d4ad4904f73cfffb1b97b075",
"group_ids":["90d7a7010ad0848e6cebfba148a611ef",   //DM_residence
             "869561de7edb65ca3fd23bdfb196cfa3",   //DG_audio
             "ea6acf330ca24c1f1cfafcb692a1c1e1"]}, //DG_audio_max
             //=> futur remove DG_audio_max

//Device Group
{"name": "DG_audio_max", "id": "ea6acf330ca24c1f1cfafcb692a1c1e1",
"group_ids":[],
"policy": { "entities": {"entity_ids": {
  "audio.audio1": { "read": true, "control": true }}}}},
{"name": "DG_audio", "id": "2df56f9331b33e1fcc86a91beb0e0e53",
  "group_ids":[],
  "policy": { "entities": {"entity_ids": {
    "audio.audio2": { "read": true, "control": true }}}},

//Decision Task
{"name": "DT_audio", "id": "8d0c4c1a937af270b50d4bb58302cc71",
"group_ids":["ea6acf330ca24c1f1cfafcb692a1c1e1",   //DG_audio_max
             "2df56f9331b33e1fcc86a91beb0e0e53" ], //DG_audio
"policy": { "group": { "editPolicy": 0.75,
  "AssignRDRA": 0.75, "RevokeRDRA":0.01}}},

{"name": "DT_groups", "id": "02a27a7864721c7ce85b1857c079d0b6",
"group_ids":["a117f2d6d4ad4904f73cfffb1b97b075"], //resident
"policy": { "group": {"editPolicy": 0.75,
    "AssignUA": 0.5, "RevokeUA": 0.5,
    "AssignRH": 1.0, "RevokeRH": 0.75}}},

//Decision maker
{"name": "DM_residence", "id": "90d7a7010ad0848e6cebfba148a611ef",
"group_ids":["8d0c4c1a937af270b50d4bb58302cc71",    //DT_audio
             "02a27a7864721c7ce85b1857c079d0b6"],   //DT_groups
"policy": { "group": {"editPolicy": 0.75,
   "AssignDMRA": 1.0, "RevokeDMRA": 0.75}}}
],
```

To facilitate the decision-making process, there is also a decision part included in this JSON structure that indicates the current status of the decision process. In the JSON provided below, we have an example where Resident1 revokes access from Resident to DG_audio_max through the RevokeRDRA action.

Given that there are two members in this decision maker group and the governance policy for RevokeRDRA is set at 1%, Resident1 can make the decision to revoke access on their own without requiring the approval of the other member.

```
"decision": [
{"source": "Resident1",
"target": "resident",
"group": "DG_audio_max",
"action": "RevokeRDRA",
"reject": [],
"approve": ["Resident1"],
"result": true
}]
```

4.4.2. Example 2

In this residential setting, an audio system with two volume settings (normal and maximum) is present. Alongside the resident members, there are also visitors. Visitors are granted regular access, which is subject to a decision made by the DM_residence. Furthermore, visitors may be granted access to the audio system's maximum volume level. To ensure a comprehensive explanation, the visitor is overseen by the DT_groups decision task, and a user named "visitor" is associated with the visitor group.

In Figure 22 this example is visually represented. The diagram highlights in red the relationship that need to be added by decision in order to grant the Visitor access to the maximum volume of the smart speaker.

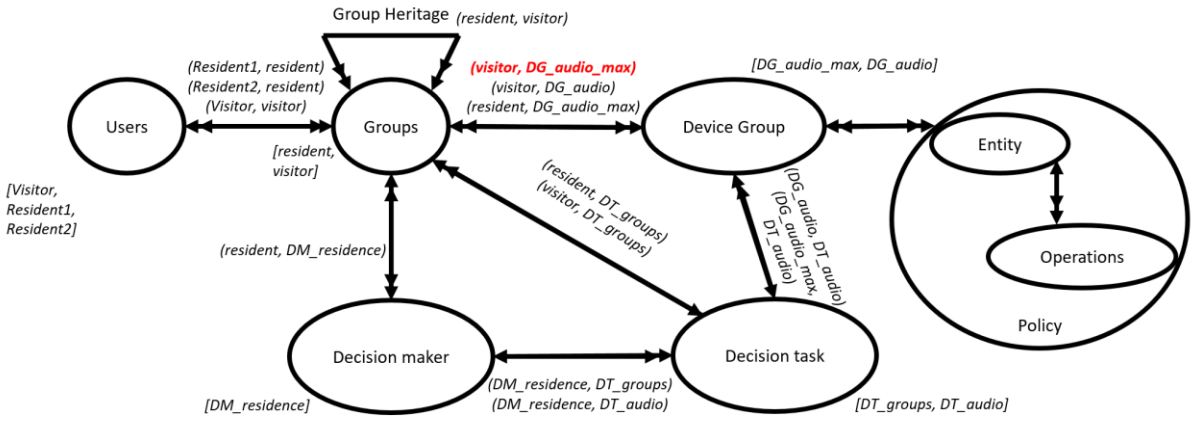

Figure 22: PoC model example 2

In this example, we have three users: two residents and one visitor. Hereunder, this is described in JSON format that Home Assistant understands.

```
"users":[
{"name": "Resident1", "id": "46e84fae840d4beb81013e6c6231541b",
"group_ids": ["a117f2d6d4ad4904f73cfffb1b97b075"], //resident
"is_owner": false,"system_generated": false,
"is_active": true,"local_only": false},

{"name": "Resident2", "id": "9ed2bbd147abf7c785a693dcdabe38c0",
"group_ids": ["a117f2d6d4ad4904f73cfffb1b97b075"], //resident
"is_owner": false,"system_generated": false,
"is_active": true,"local_only": false},

{"name": "Visitor", "id": "964786cbb43b7dd0971bc540cc84583f",
"group_ids": ["869561de7edb65ca3fd23bdfb196cfa3"], //visitor
"is_owner": false,"system_generated": false,
"is_active": true,"local_only": false}
],
```

Same as in example 1, we show here all the groups needed and their relationships. Particularly noteworthy is the governance policy of DT_audio, where it is observed that 75% of the decision maker group must agree to assign a new device group to a group.

```
"groups":[//Group
{"name": "resident", "id": "a117f2d6d4ad4904f73cfffb1b97b075",
"group_ids":["ea6acf330ca24c1f1cfafcb692a1c1e1",   //DG_audio_max
             "90d7a7010ad0848e6cebfba148a611ef",   //DM_residence
             "869561de7edb65ca3fd23bdfb196cfa3"]}, //visitor

{"name": "visitor", "id": "869561de7edb65ca3fd23bdfb196cfa3",
"group_ids":["2df56f9331b33e1fcc86a91beb0e0e53"]}, //DG_audio
//=> futur DG_audio_max

//Device Group
{"name": "DG_audio_max", "id": "ea6acf330ca24c1f1cfafcb692a1c1e1",
"group_ids":[],
"policy": { "entities": {"entity_ids": {
  "audio.audio1": { "read": true, "control": true }}}}},
{"name": "DG_audio", "id": "2df56f9331b33e1fcc86a91beb0e0e53",
  "group_ids":[],
  "policy": { "entities": {"entity_ids": {
    "audio.audio2": { "read": true, "control": true }}}}},

//Decision Task
{"name": "DT_audio", "id": "8d0c4c1a937af270b50d4bb58302cc71",
"group_ids":["ea6acf330ca24c1f1cfafcb692a1c1e1",   //DG_audio_max
             "2df56f9331b33e1fcc86a91beb0e0e53" ], //DG_audio
"policy": { "group": { "editPolicy": 0.75,
    "AssignRDRA": 0.75, "RevokeRDRA":0.01}}},


{"name": "DT_groups", "id": "02a27a7864721c7ce85b1857c079d0b6",
"group_ids":["a117f2d6d4ad4904f73cfffb1b97b075",  //parent
             "869561de7edb65ca3fd23bdfb196cfa3",  //children
             "aac04c91ec459fd4c5b464bbec1d344a"], //babysitter
"policy": { "group": {"editPolicy": 0.75,
    "AssignUA": 0.5, "RevokeUA": 0.5,
    "AssignRH": 1.0, "RevokeRH": 0.75}}},

//Decision maker
{"name": "DM_residence", "id": "90d7a7010ad0848e6cebfba148a611ef",
"group_ids":["8d0c4c1a937af270b50d4bb58302cc71",    //DT_audio
             "02a27a7864721c7ce85b1857c079d0b6"],   //DT_groups
"policy": { "group": {"editPolicy": 0.75,
   "AssignDMRA": 1.0, "RevokeDMRA": 0.75}}}
],
```

Finally, the last block of JSON in this example shows the Home Assistant representation of how the assignment occurred. Initially, Resident1 initiated a decision process to assign a group to a device group. The required threshold for approval is 75%, which means that both residents need to approve the decision for it to be accepted.

```
"decision": [
{"source": "Resident1",
"target": "visitor",
"group": "DG_audio_max",
"action": "AssignRDRA",
"reject": [],
"approve": ["Resident1", "Resident2"],
"result": true
}]
```

4.4.3. Example 3

In the last example, let's say that two parent (Parent1, Parent2) have a child (Child) whom is often taken care by the babysitter (Babysitter). They want to give to the babysitter membership to the decision board to the tv (DG_TV) access, where both children and parent have access. To do so the parents make the decision to add her to the decision maker group (DM_TV).

Below in Figure 23, we have depicted all the different relations. The red marking indicates that the role "babysitter" will become a member of the decision maker group DM_TV. This means that the user "Babysitter" will gain decision-making power within DM_TV, enabling them to initiate decisions and vote in other decision processes within DM_TV.

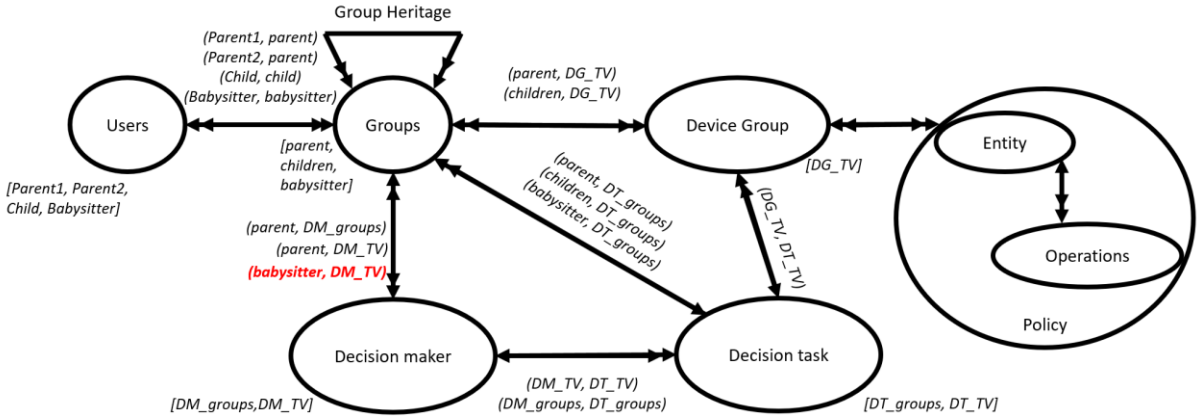

Figure 23: PoC model example 3

In this example, there are four users: two parents, a babysitter, and a child. Each user is assigned to their respective group.

```
"users":[
{"name": "Parent1", "id": "46e84fae840d4beb81013e6c6231541b",
"group_ids": ["a117f2d6d4ad4904f73cfffb1b97b075"], //parent
"is_owner": false,"system_generated": false,
"is_active": true,"local_only": false},

{"name": "Parent2", "id": "9ed2bbd147abf7c785a693dcdabe38c0",
"group_ids": ["a117f2d6d4ad4904f73cfffb1b97b075"], //parent
"is_owner": false,"system_generated": false,
"is_active": true,"local_only": false},

{"name": "Child", "id": "964786cbb43b7dd0971bc540cc84583f",
"group_ids": ["869561de7edb65ca3fd23bdfb196cfa3"], //children
"is_owner": false,"system_generated": false,
"is_active": true,"local_only": false},

{"name": "Babysitter","id": "4040f1f6fe592ef69b7a202167c3cb59",
"group_ids": ["aac04c91ec459fd4c5b464bbec1d344a"], //babysitter
"is_owner": false,"system_generated": false,
"is_active": true,"local_only": false}
],
```

Now, we have the bulk of the JSON where all the relations are noted. The most noteworthy aspect is the governance policy for AssignDMRA in the DM_TV group. It is set at 1.0 or 100%, which means that all owners of this Decision Maker group (in this case, the Parent group) must vote positively to allow the babysitter role to join their decision maker group.

```
"groups":[//Group
{"name": "parent", "id": "a117f2d6d4ad4904f73cfffb1b97b075",
"group_ids":["ea6acf330ca24c1f1cfafcb692a1c1e1",   //DG_TV
             "90d7a7010ad0848e6cebfba148a611ef",   //DM_TV
             "f510591d91fae3175d1e25ed8b27275b"]}, //DM_group

{"name": "children", "id": "869561de7edb65ca3fd23bdfb196cfa3",
"group_ids":["ea6acf330ca24c1f1cfafcb692a1c1e1"]}, //DG_TV

{"name": "babysitter", "id": "aac04c91ec459fd4c5b464bbec1d344a",
"group_ids":[]}, // <= futur DM_TV

//Device Group
{"name": "DG_TV", "id": "ea6acf330ca24c1f1cfafcb692a1c1e1",
"group_ids":[],
"policy": { "entities": {"entity_ids": {
  "TV.TV1": { "read": true, "control": true }}}}},

//Decision Task
{"name": "DT_TV", "id": "8d0c4c1a937af270b50d4bb58302cc71",
"group_ids":["ea6acf330ca24c1f1cfafcb692a1c1e1"],  //DG_TV
"policy": { "group": { "editPolicy": 0.75,
  "AssignUA": 0.5, "RevokeUA": 0.5,
  "AssignRH": 1.0, "RevokeRH": 0.75}}},

{"name": "DT_groups", "id": "02a27a7864721c7ce85b1857c079d0b6",
"group_ids":["a117f2d6d4ad4904f73cfffb1b97b075",  //parent
             "869561de7edb65ca3fd23bdfb196cfa3",  //children
             "aac04c91ec459fd4c5b464bbec1d344a"], //babysitter
"policy": { "group": {"editPolicy": 0.75,
    "AssignUA": 0.5, "RevokeUA": 0.5,
    "AssignRH": 1.0, "RevokeRH": 0.75}}},

//Decision maker
{"name": "DM_TV", "id": "90d7a7010ad0848e6cebfba148a611ef",
"group_ids":["8d0c4c1a937af270b50d4bb58302cc71"],   //DT_TV
"policy": { "group": {"editPolicy": 0.75,
   "AssignDMRA": 1.0, "RevokeDMRA": 0.75}}},

{"name": "DM_groups", "id": "f510591d91fae3175d1e25ed8b27275b",
"group_ids":["02a27a7864721c7ce85b1857c079d0b6"],   //DT_groups
"policy": { "group": { "editPolicy": 0.75,
    "AssignDMRA": 1.0, "RevokeDMRA": 0.75}}}
],
```

Finally, we have the decision where the babysitter role obtains membership to the DM_TV group. As mentioned earlier, all parents had to approve this decision to allow the babysitter to join them in the decision-making group.

```
"decision": [
{"source": "Parent1",
"target": "babysitter",
"group": "DM_TV",
"action": "AssignDMRA",
"reject": [],
"approve": ["Parent1", "Parent2"],
"result": true
}]
```

We have successfully validated the practical applicability of our theoretical model through the implementation of our Proof of Concept (PoC). This chapter serves as a testament to the versatility and effectiveness of our model in addressing the evolving needs of co-living environments. By embracing our model, co-livers can experience greater openness, improved communication, and enhanced collaboration within their shared living spaces. The model facilitates a sense of community and encourages collective decision-making, fostering a harmonious and inclusive living experience. With this innovative approach, we aim to empower individuals in co-living arrangements and create an environment that promotes active participation and shared responsibility.

5. Conclusion

In recent years, smart home and IoT have become more prevalent than ever. Traditional access control systems focus on individual homes or businesses, where a central authority controls access rights. At the same time, more co-living arrangements are spreading.

These systems seemed inadequate for the world of co-living. Co-living is familiar to us, especially as students, because it is similar to the system of shared university residences (named *kot* in Belgium) that include common areas.

We became convinced that there was a topic for reflection. The crux of the problem seemed to be access control. Therefore, in this paper, we set out to explore the state of the art in access control within the context of smart homes.

In our research, we found systems applicable to small private spheres, such as a single family, or to business-oriented environments.

The main characteristic of these systems was a high level of concentration of power in the hands of a few administrators, often limited to just one individual.

The system of a single administrator with full authority creates an asymmetry of power between the administrator and the users, which can potentially lead to conflicts.

Therefore, we were led to the idea of enhancing the administration by suggesting a new model that replaces the single administrator with decision-making groups.

To achieve this, four distinct concepts were introduced:

- Firstly, user groups adopt the concept of RBAC (Role-Based Access Control), where users are assigned roles. These roles can be part of an inheritance tree.
- Next, device groups are a collection of permissions associated with specific devices.
- Decision tasks incorporate governance rules applied to roles and device groups.
- Lastly, decision tasks are assigned to decision makers, who are responsible for making the final decision.

In order to understand how decision makers could make better decisions, we looked at participatory management systems in social and political contexts as an analogy. In other contexts, the voting system has been a solution to give voice to a larger number of people. Therefore, we introduced the voting system into access control within the field of co-living. Similar to traditional voting systems that establish predefined thresholds for a positive outcome, our system also includes decision-making thresholds in its governance rules.

Our model of access control has been implemented in a real-world smart hub system as a Proof of Concept.

By involving users in the decision-making process, it fosters transparency, collaboration, and a community-oriented approach to access control. This flexibility enables the system to adapt to changing circumstances and user requirements, ensuring its responsiveness and effectiveness.

In addition to the proposed solution, the writing of this paper has sparked personal reflections on the following new avenues.

The proposed system requires substantial user involvement and active participation. Users may initially feel hesitant about embracing a new system and may face difficulties in actively engaging with it. The development of an intuitive graphical user interface can be a solution to improve user interaction with the system.

People often overlook the importance of privacy safeguards until they experience issues in the complex and increasingly open virtual world. To ensure privacy, it is crucial to generate data embracing Privacy Enhancing Technologies. To minimize the potential impact of such privacy issues, it is important to avoid indefinitely storing the entire data history of devices. Furthermore mindsets still need to evolve in order to better anticipate the potential dangers associated with new technologies. This issue itself requires careful consideration and reflection.

The proposed system is not limited to a specific context. It can be applied to other administration systems in various shared models, not only within connected homes. The implementation of a voting system using blockchain technology can offer enhanced security measures. However, it is important to consider that this approach may introduce additional complexities in managing the system.

6. Bibliography

[1] 'The history of the smart home from 1963 - 2023: milestones', *https://www.smartest-home.com/en/*. https://www.smartest-home.com/en/history-of-smart-home/ (accessed May 29, 2023).

[2] 'ECHO IV', *Wikipedia*. Apr. 24, 2023. Accessed: May 29, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=ECHO_IV&oldid=1151504512

[3] 'How our homes became smart: The history of home automation', *Deccan Chronicle*, Jan. 12, 2019. https://www.deccanchronicle.com/technology/in-other-news/120119/how-our-homes-became-smart-the-history-of-home-automation.html (accessed May 29, 2023).

[4] 'Ambient intelligence', *Wikipedia*. May 23, 2023. Accessed: May 29, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Ambient_intelligence&oldid=1156509491

[5] 'Electoral system', *Wikipedia*. May 02, 2023. Accessed: May 29, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Electoral_system&oldid=1152798146

[6] 'Access control', *Wikipedia*. May 17, 2023. Accessed: May 18, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Access_control&oldid=1155277926

[7] 'De la serrurerie au contrôle d'accès | Welcomr'. https://www.welcomr.com/blog/52-de-la-serrurerie-au-controle-dacces (accessed May 18, 2023).

[8] 'Compatible Time-sharing System: An Overview of CTSS', *HitechNectar*. https://www.hitechnectar.com/blogs/compatible-time-sharing-system/ (accessed Apr. 29, 2023).

[9] H. C. A. Van Tilborg and S. Jajodia, Eds., *Encyclopedia of Cryptography and Security*. Boston, MA: Springer US, 2011. doi: 10.1007/978-1-4419-5906-5.

[10] R. Sandhu, D. Ferraiolo, and R. Kuhn, 'The NIST model for role-based access control: towards a unified standard', in *Proceedings of the fifth ACM workshop on Role-based access control*, in RBAC '00. New York, NY, USA: Association for Computing Machinery, Jul. 2000, pp. 47–63. doi: 10.1145/344287.344301.

[11] C. C. Editor, 'least privilege - Glossary | CSRC'. https://csrc.nist.gov/glossary/term/least_privilege (accessed Jun. 04, 2023).

[12] C. C. Editor, 'Separation of Duty (SOD) - Glossary | CSRC'. https://csrc.nist.gov/glossary/term/separation_of_duty (accessed Jun. 04, 2023).

[13] S. Ameer, J. Benson, and R. Sandhu, 'The EGRBAC model for smart home IoT', presented at the 2020 IEEE 21st International Conference on Information Reuse and Integration for Data Science (IRI), IEEE, 2020, pp. 457–462.

[14] M. Shakarami and R. Sandhu, 'Role-Based Administration of Role-Based Smart Home IoT', in *Proceedings of the 2021 ACM Workshop on Secure and Trustworthy Cyber-Physical Systems*, Virtual Event USA: ACM, Apr. 2021, pp. 49–58. doi: 10.1145/3445969.3450426.

[15] S. Contributor, 'RBAC vs. ABAC Access Control: What's the Difference? - DNSstuff', *Software Reviews, Opinions, and Tips - DNSstuff*, Oct. 31, 2019. https://www.dnsstuff.com/rbac-vs-abac-access-control (accessed May 31, 2023).

[16] N. Malkin, A. F. Luo, J. Poveda, and M. L. Mazurek, 'Optimistic Access Control for the Smart Home', presented at the 2023 IEEE Symposium on Security and Privacy (SP), IEEE Computer Society, Dec. 2022, pp. 2112–2129. doi: 10.1109/SP46215.2023.00121.

[17] M. J. Moyer and M. Abamad, 'Generalized role-based access control', in *Proceedings 21st International Conference on Distributed Computing Systems*, Mesa, AZ, USA: IEEE Comput. Soc, 2001, pp. 391–398. doi: 10.1109/ICDSC.2001.918969.

[18] 'Home Assistant'. Home Assistant, Apr. 27, 2023. Accessed: Apr. 27, 2023. [Online]. Available: https://github.com/home-assistant/core

[19] 'Domoticz'. Domoticz, Jun. 01, 2023. Accessed: Jun. 01, 2023. [Online]. Available: https://github.com/domoticz/domoticz
[20] 'openhab/openhab-core: Core framework of openHAB'. https://github.com/openhab/openhab-core (accessed Jun. 01, 2023).
[21] 'ioBroker/ioBroker: Automate your life!' https://github.com/ioBroker/ioBroker (accessed Jun. 01, 2023).
[22] N. Gennart, 'Role-based access control for openHAB users', UCLouvain, 2022. [Online]. Available: https://hdl.handle.net/2078.1/thesis:35683
[23] https://twitter.com/balloob, 'Can I Have User Permissions? | Home Assistant Developer Docs', Mar. 11, 2019. https://developers.home-assistant.io/blog/2019/03/11/user-permissions/ (accessed Apr. 27, 2023).
[24] 'Open letter for improving Home Assistant's Authentication system (OIDC, SSO) - Feature Requests', *Home Assistant Community*, Mar. 02, 2023. https://community.home-assistant.io/t/open-letter-for-improving-home-assistants-authentication-system-oidc-sso/494223?page=3 (accessed May 06, 2023).
[25] 'The WebSocket API (WebSockets) - Web APIs | MDN', Mar. 07, 2023. https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (accessed May 08, 2023).
[26] R. Sandhu, 'Roles versus groups', in *Proceedings of the first ACM Workshop on Role-based access control - RBAC '95*, Gaithersburg, Maryland, United States: ACM Press, 1996, pp. 7-es. doi: 10.1145/270152.270163.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**