Multi-Tenant Management in Secured IoT Based Solutions

Kerem AYTAÇ[1 2], Ömer KORÇAK[2]
[1]Koç Digital R&D Center, [2]Marmara University
Istanbul, Turkey
keremaytac@gmail.com, omer.korcak@marmara.edu.tr

***Abstract*—The existence of Internet of Things (IoT) can be noticed in many different areas. There are lots of IoT implementations in different domains which facilitate existing workloads and solve many kinds of struggles within that specific domain. The excessive growth in IoT based solutions for both home and industry applications caused developers to pay great attention over the requirements such as reusability, modifiability, and interoperability with a robust security. Many IoT products should present a core feature which serves plenty of customers from different domains and provide domain-specific features for each of them. This approach also forces the IoT products to be multi-tenant supported to satisfy the requirements. Multi-tenancy brings more strict security considerations, organizational approaches, policies, roles, and identity managements. In this paper, we propose a solution for implementing multi-tenancy and tenant management in an IoT product, and drill down to details in order to disclose how to make that product modular and give the ability to serve unlimited number of vertical solutions.**

I. INTRODUCTION

There is a steady increase in the data generated by things and in both quantity and variety of Internet of Things (IoT) solutions which manage those data and enable data-oriented problem solving within single or multiple domains. These solutions are very crucial in especially industry where there are lots of machines to be sensed and monitored. However, many IoT products are really tended to be designed use-dispose style which blocks reusability and modifiability and only serve a single customer with a single domain within a specific time. It is required to have a core IoT product which not only provides interoperability and modifiability, but also multi-tenancy which brings reusability with a high security. These requirements are especially apparent for the industrial IoT solutions.

There are many types of customers for such a product. Some of them are companies with top-to-bottom hierarchical organization, while some of them are holding companies which uses this product in many internal or subsidiary companies that belong to that holding. Moreover, some of them use this product to adopt other companies into the product and act as a product distributor in a specific domain. All of them strictly require a multi-tenant management which is the pillar of Software as a Service (SaaS) products in cloud [1].

There are several studies on how to manage tenants in multi-tenant environments. Kalra et al. [2] handles performance issues against an Application Programming Interface (API) for a SaaS Application in cloud that is used by many tenants. They propose a methodological framework in order to boost resource utilization. Similarly, Mace et al. [3] proposes an original resource management framework for shared distributed services. This framework observes and tracks the resource consumption either inside a distributed system or amongst multiple distributed systems and they extract some vitals to a centralized system via API and evaluate the system in terms of tenants and intervene to throttle and fairly distribute the resources across tenants.

Besides performance approaches, there are also role-based access control approaches in the literature. As an example, Tang et al. [4] works on inter-tenant communication methodology for trusted tenants. They setup some relations between each other and they provide some detailed authorization which also asserts that very low delays happen for end users while being authorized and scalable. Bien et al. [5] develop a design pattern that is aimed to be used to build PaaS and SaaS frameworks for software developers. They named the approach as hierarchical multi-tenant pattern. Hamilton et al. [6] studies data security for multi-tenant cloud computing infrastructures. They focus on how to let the users configure easily their cloud infrastructures while also providing greater control over data security.

Moreover, many kinds of Customer Relationship Management (CRM) Applications, or Business Applications use role-based approaches within the system. Nowadays, nearly every application has a SaaS correspondence in cloud which serves online to multi-tenants. Levchenko et al. [7] examine the effects of implementation of multi-tenancy in SaaS products to business and its misalignment with IT side resulting with lower business value realization and the struggles to develop a project. They introduced some functional requirements in order to implement and adapt multi-tenant SaaS approach by simulating it against reference business process models. This approach is validated on a well-known business application as well. Another role-based approach comes from Microsoft with a CRM application namely Dynamics CRM. In CRM world, organization management is the most vital thing to manage accurately [8]. Their role management is also very insightful and inspiring to design some parts of our approach.

Every customer should be separated and isolated in a product as their data belongs to themselves. For the sake of clarity, we can describe the approach with a hotel example. We can assume the customers as separate hotel buildings with different facilities or features in a street and each building has its own security guards

and barriers which only allow authorized access for that building. However, organizations or departments in a company are more like the rooms in the same hotel. They can be fully or partly accessed by some people with a given privileged card such as managers, cleaners or single accessed by room residents with a card grants access only a specific room. Sometimes, some companies (i.e., holding or group companies) assume themselves as a single tenant, and wants to manage its own subsidiary or sub-companies with a very high privilege access key within the same tenant. They are more of a group of hotel buildings that reside in different places (i.e., street, city, country), but owned by same owner. Each of them has similarities where some differences present in terms of facilities or features, but they serve with the same name differentiating with some place names (i.e., Hotel Foo Garden– City A and Hotel Foo Suites – City B). So, there are different customer types with different access privileges. Residents only book some room from City A or City B and reside where he books and unable to access the other. Cleaners are the employees that belong to and be paid by only one of the hotels. However, the owner or co-founder of the hotel group can access and manage any of them regardless of in which city it is.

Thus, the IoT product should show a behavior which can handle these structure types of companies in a secure way. Additionally, that IoT product should be attachable by some other vertical solutions to empower the regarding domain. This vertical solution also should present the capabilities of core mentioned above. In this paper, we address all these issues and propose a viable approach for implementing multi-tenancy and tenant management in an IoT product by enabling unlimited number of vertical solutions. We provide technical details of how we manage the tenants and give sample scenarios that cover various cases on policy and role management issues in an IoT product.

The rest of the paper is organized as follows. Next section has a glance at the company structures. Section III deep dives into the IoT product structure and the detailed explanation of proposed solution. Section IV gives the implementation details and some useful visual demonstrations to create better understanding. Finally, Section V concludes the paper.

II. OVERVIEW OF COMPANY STRUCTURES

As introduced in the previous section, without loss of generality, expressing with the hotel example is a convenient way to define the problem. Fig. 1 illustrates a sample tenant structure of a company, namely Company A, which owns two hotels and one cleaning company which uses the IoT Product. We call the root company as "**tenant**" which buy and use the product. Companies have very top-to-bottom hierarchical and tree-branch like organizations where some of them are parents, and some of them are children. Parents manage the children; children report to parents. For example, in Fig. 1, Front Desk organization manages two organizations, Reception and Welcoming Committee. Such organizations with low access privileges are simply called "**Child Organization** or **Organization**".

As also described in the previous section, there are some companies which break down into subsidiaries or other sub-companies and they want to use that product with a single tenant rather than multiple tenants per company. Yes, they are different companies to each other, nevertheless they owned by the same root company, meaning that they are not isolated as much as different tenants are. We call those isolations as "**Zone Organizations**". Although they behave like separate tenants, zones are managed by its regarding tenant**.**

Moreover**,** tenant itself is also an organization which is treated as root. The root is one and only one. We call that organization as **"Root Organization"**, where all the zones are parented by. Root organization is more of a virtual organization to represent a joint point for all organization at the top.

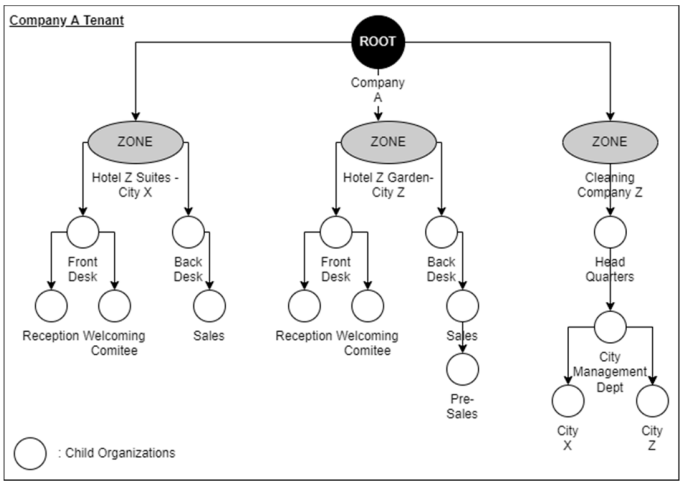

Fig. 1 Sample Tenant Structure of Company A

In Fig. 1, Company A also founded a cleaning company to dedicate to hotels which the company owns and performs the cleaning stuff in-house by not outsourcing to other cleaning companies. Each cleaning city organization is responsible with the hotels reside in the same city. Hotel Z Garden and Hotel Z Suites has similar organization schema, except for a Pre-Sales child organization as Hotel Z Garden needed to invest for pre-sales operations as it is less demanded hotel comparing to the other hotel.

Tenant structures in other domains are analogous to this example, and the hierarchical structure given in Fig. 1 can be adopted to many different domains. For example, we may consider the international banks with vertical and tall organization hierarchy. They have some regional organizations which manages a bunch of cities, where city organizations manage the branches in that city. Moreover, country representatives also manage the regions in that country. This depiction is the roughest one, they may have more mid-level parent organizations which manages a bunch of children. Generally, in the banks, above organizations have the privilege to access the below ones. Such as, a Human Resources (HR) employee in region can access some information about employees that belongs to city organization as well as branch employees. However, that HR employee cannot access any information about country representative manager or CEO of the bank.

III. OVERVIEW OF IOT PRODUCT STRUCTURE AND SOLUTION PROPOSAL

In this section, we will focus on the IoT software product structure and more technical details will be revealed. The IoT product contains a core as a generic and horizontal product which

provides a framework for different IoT applications, providing a user interface for tracing and tracking data, generating data-related reports, providing other value adding modules such as machine learning, rule engine, and so on. It does not present any domain-specific features, but serves to any kind of domain. Moreover, this IoT core exposes very helpful endpoints to be mounted by other solutions in order to extend, specialize and customize the capabilities of product to specific domains.

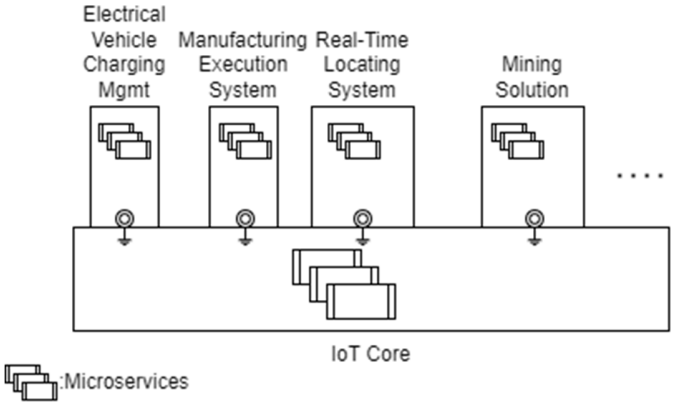

Fig. 2 IoT Product and Vertical Solutions Architecture Overview

Both core and vertical solutions as in the Fig. 2 (such as Manufacturing Execution System [9]) consist of lots of microservices which communicates each other by using recent and contemporary microservice architectural approaches. All microservices expose some useful endpoints to be used by user interfaces, other internal microservices, other vertical solution microservices, other 3rd party external services, etc. Those endpoints should be authorized and authenticated, otherwise it causes a great security leak. Authentication proves who you are, whereas authorization shows how privileged you are. Those two notions should be satisfied to make the system secure and robust. Authentication can only grant or deny a user to access to the product services, and if it is a valid user; the border-check can be passed easily. However, accessing to the objects that a user does not have to see or have to see or may partially see is matter of authorization which is the main security wall inside the software. For instance, a user is not allowed to see any objects that belongs to other customer (tenant) or another zone that he/she does not belong to, or an upper organization that the user must not access due to his/her job definition.

To define more what are the objects that reside in core or other solutions; let us just grab the instantiation of Company A tenant which adopts this IoT product, and everything is remote controlled or monitored with high-end devices. Let us assume that, in Hotel Z Garden in City Z, there is a door between front desk and back desk and this door can be ruled by only back desk employees. This door can be opened remotely by user interface that is provided by core. So, back door users should login into the system, find the necessary door record and hit the open button. However, front desk users should neither see nor command the door. Besides, the hotel management that works in related zone should command that door as well.

Let us just enhance the scenario, and say this hotel also bought a solution namely "RTLS (Real Time Locating System) [10] Management" for their auto guided robot vacuums where a study uses this solution to satisfy warehouse security and increase the efficiency on operations. Those vacuums clean up the hotel. So, these vacuums are also considered as another objects in core and RTLS solution. The door that we mentioned recently also should be known by RTLS solution, because vacuums should open or close the door when they need to clean beyond the door from front desk. So, the door is a shared object by core and RTLS. However, a temperature sensor that is embedded in the wall of reception is none of RTLS solution's business. So, this sensor only belongs to the core. Someone from reception can monitor the temperature.

More specifically, hotel building object is a generally shared object amongst any solution as it is related to any of them. Both core and RTLS solution or any possible solutions later probably need to show building information in their interfaces. So, in Fig. 3 a general overview of object distribution can be observed which covers the above scenario.

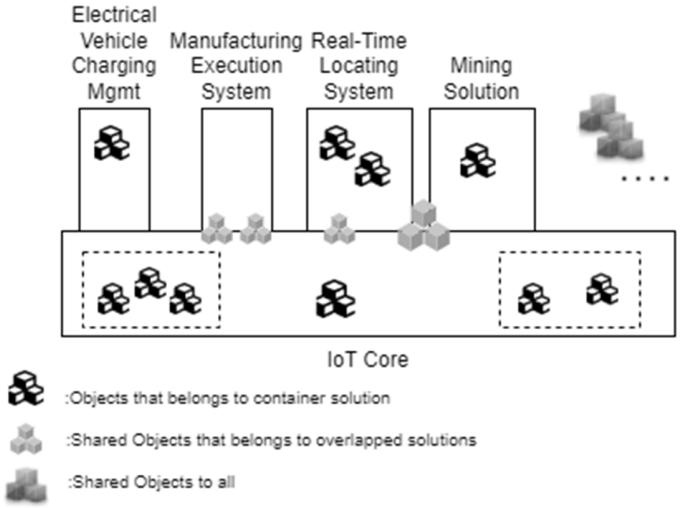

Fig. 3 Object distribution over core and other solution space

This object belongings to some solutions are called namely **"Solution Ownership"** in this study (a.k.a **"Application Ownership**). The final scenario, but not the least, any object that does not belong to that specific solution (a.k.a **Application**) cannot be seen even by a fully authorized user in that solution.

Solution Ownership is the most generalized and coarse way to manage the object ownerships. We need a fine-tuned way that prevents from unauthorized access to objects within the same solution according to user, user's organization, roles, grant levels etc. In this part, there will be more details which is the authorization backbone of the product.

Hereinafter, object word will be called as **asset** where they are the beings, objects in IoT world. This ownership is so called **"Asset Ownership".** So, every asset may have some owners other than solutions. Before diving deeply into this, let us just discuss about users, organizations and roles, permissions which are the decomposed pieces of roles and features creating solutions and what customers pay for.

Users are the ones who are managed by IdentityServer and authenticated to use IoT product and surrounding solutions. Each user should belong to a child organization or a zone, and thus, to a tenant.

Role is a key that has access to data with respect to their permission levels. A role is combined with user and organization

so that it defines that the role will provide an access to that specific user over selected organization(s). It contains various type of permissions.

EndpointItemPermission is responsible to validate a user to reach that specific type of service endpoint. It is generally used in microservice endpoints. (e.g. Asset_Read, Asset_Delete, Vacuum_Read). If no such permission found for user, service generates 403 Forbidden HTTP result as defined in the RFC document [11] for related request.

MenuItemPermission is responsible to render menu in a UI project. This permission includes route information, menu icon info, order number, or child menus if exist etc. This permission type is used to define the menu for redirecting to a page in accordance with which menu is clicked. (e.g. Rule Management, Asset, Asset Type menu items). If a user does not have this permission, then they will not be able to see that menu in UI.

UIItemPermission is responsible for components, buttons, sections, or any kind of UI Items for displaying/not displaying to the corresponding user. Any UI will need to take these permissions into consider in order to decide what to show or not. (e.g. Gauge Chart Widget, Update Profile Picture Button, A Create button for an object).

Permission Groups consist of meaningful set of permissions to manage them easily and intuitively. For example, an "Asset User Permission Group" may contain Asset Read and Create endpoint permissions and also AssetType Read endpoint permission and Asset MenuItem Permission and also "Create an Asset Button" UIItemPermission. So, if the permissions are atoms, permission groups are molecules. So, there should not be inconsistent atoms within a molecule.

Policy corresponds to an authorization area, also known as Organization. Organizations are the things that have strict boundaries by containing some assets or users. So, policy is another nomenclature of organization.

Feature is the smallest struct within a solution that presents a capability. It contains bunch of permission groups in order to manage authorization for users.

Feature Set is basically a set of features.

Solution, also known as Application, is set of feature sets. Thus, if some feature sets are combined, it will correspond to a solution.

Considering these definitions, a customer pays for a solution regarding the feature sets consisting of. For example, a customer wants to enable Electric Vehicles Charging Management System (EVCMS) whereas a study explains very well about the usage and adaptation of this solution for the sake of more efficient battery usage [12] in their Core IoT Platform as a vertical solution. This solution contains two feature sets namely "Charging Tracker" and "Reporting" as in Fig. 4. Each feature set also includes some features. For example, they want to communicate with Core IoT via OCPP (Open Charge Point Protocol) which is an EVCMS-specific communication protocol where the protocol usage in this domain is well-defined in a study [13], but they do not want to enable any voltage surge protection software, because they already handle this problem in charging unit as is. Moreover, they do not want any reporting features, so Reporting feature set will not be enabled for this customer.

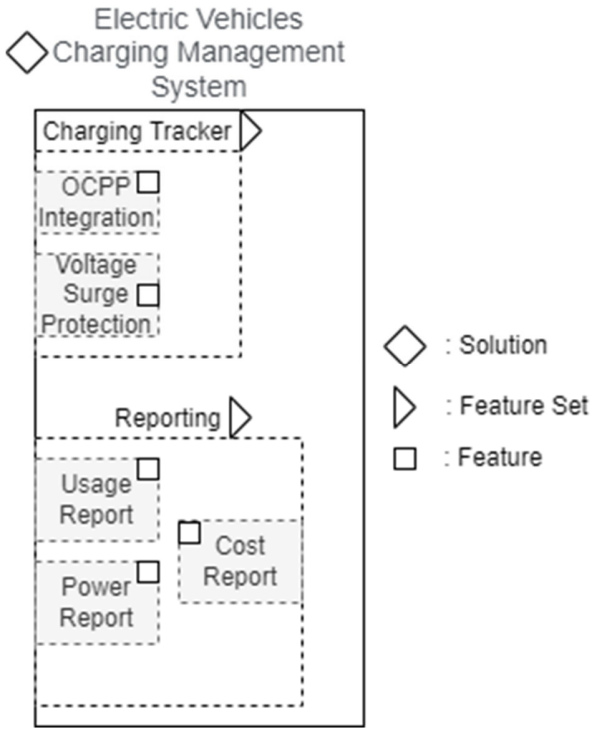

Fig. 4 Solution to Feature Decomposition for a EVCMS solutions

If a feature is not enabled, then there will be no permission groups, thus no permission will be granted to that customer (tenant). So that, this customer will not be authorized somehow to any endpoints or UI things which corresponds to a specific feature.

Now let us refer to the zones introduced in the previous section. Zones are the structures which buy solutions. So, in a tenant with multiple zones, each zone can prefer different type of solutions, feature sets or features and any organization that is connected to zone at the top can only use them. All the zones are isolated from each other in terms of purchased solutions. However, this isolation is not a hard line, but a soft line. As described previously, each user belongs to one and only one child organization within zone. From that point on, roles come to the stage and show themselves to make users grant any kind of authorization. Zones are analogous to closed-door rooms, and users are the residents inside these rooms whereas the roles are the keys for these rooms. Thus, anyone from Zone Hotel Z Suites – City X can access to any feature from Hotel Z Garden – City Z, thanks to roles. Fig. 5 illustrates a sample scenario where Company A purchased a set of solutions in accordance with zones.

Roles are not only simple door keys, but also defines how to access. Now, we deep dive into the role definition. Roles are only associated with one zone and one solution that is purchased by zone. It contains feature sets and thus features and thus permission groups and thus eventually Endpoint, UI Item and Menu permissions in it. We simply grant privileges to permission groups in terms of Read, Create, Update or Delete. If we grant Read privilege for a permission group, then we grant accordingly any permissions that resides in that permission group with a given access level.

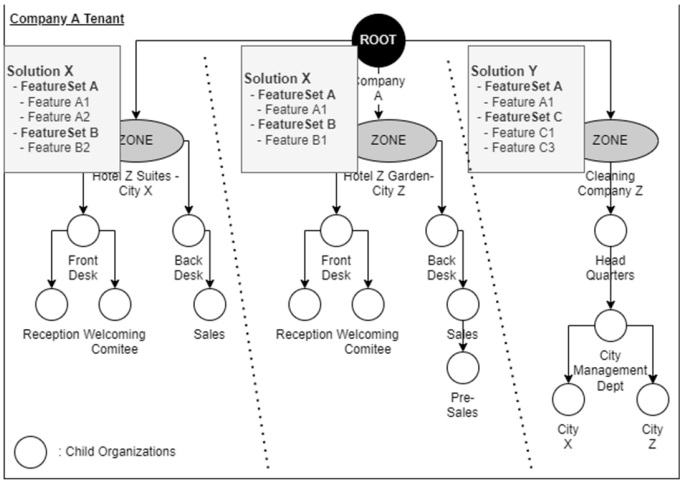

Fig. 5 Illustration of Hotel Tenant with Purchased Solutions and enabled feature sets, and features.

Roles are standalone objects, and one can assign roles to users by associating organizations as in Fig. 6. To instantiate this description let us say we have Role A with some CRUD (Create, Read, Update, Delete) privileges and we want to assign this Role A to User B to be used in Organization 1 in Zone X and another Organization namely 2 in Zone Y. Additionally, we want the user to use this role wherever he is. So, if this user changes his organization, this role will automatically be valid in that organization. Here we see that roles are only usable with organization association by users, and they use roles as a key to access that closed room.

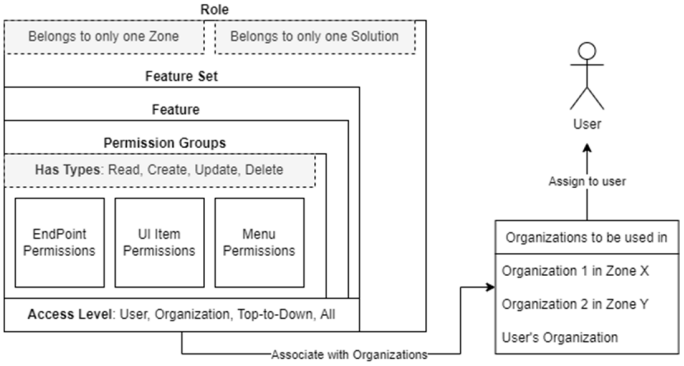

Fig. 6 Role decomposition and depiction of how they are assigned to users

Now, we consider ownerships defined previously. Each object can have an owner by a user or organization or entire customer (tenant). If an object does not have any owner or customer type of ownership, then there will be no role rules applied, anyone can access those objects if they are granted with authorization for managing that object space. For example, if a user is granted with "Car Read" privilege, then he will be reading that "Car" no matter how he gets the roles, or where his organization is.

Another role limiter and fine-tuner approach is "Access Levels" for Permission Groups. This approach sets the access rights for organizations and records that is owned by someone.

- **User Level:** If we create a role and grant "Car Read" privilege with User level, then the user that is assigned with this role can only access his own records, meaning that, the cars that he individually owned. Thus, he cannot see any other cars that is owned by other users or organizations.
- **Organization Level:** If we create a role and grant "Car Read" privilege with Organization level, then the user that is assigned with this role can only access the Cars owned by organization or users in this organization that is addressed by Role-Organization association.
- **Organization and its children (top-to-down):** If we create a role and grant "Car Read" privilege with this level, then the user that is assigned with this role can only access the Cars owned by organization or users in this organization and propagate down to its child organizations that is addressed by Role-Organization association. To exemplify, "a manager can see any salaries of any employee that is in all the branches below his branch".
- **All Organizations in the zone:** Barely seen from the definition, this level of privilege will make the user to see all the records that belongs to any organization under the zone which the role is defined for.

Some other notion in this approach is User Types. There are three types of users.

- **Superadmin:** One and only user that can rule the whole company which is exempted from any roles and privileges. This user can access and do any operation against any objects, can create users, roles, organizations and more. It is assumed that only a single superadmin should exist within tenant space to provide some security and control.
- **Admin:** Zone admins can rule the whole zone in a tenant as superadmin does, but only a specific zone. They can also create any roles, organizations, users in the given zone. They require roles to access other zones. Any user can be assigned with an admin role by superadmins at any time to rule a zone.
- **Normal User:** Simple, basic users other than admins. They need roles to move inside the organization hierarchy, otherwise they do not have any access capability.

The last but not the least notion in this approach is Organization Types. There are two types of organizations.

- **Normal Organization:** The ordinary organization type.
- **Isolated Organization:** A niche but useful organizations that presents a sandbox within zones. They are tightly sealed organizations that no usual roles can access with any kind of propagating privilege levels (top-to-down and all organizations level). In order to access them, the role should be associated especially to that isolated organization. An Isolated organization can only be created under a normal organization. No normal organization can be created under an isolated organization.

IV. IMPLEMENTATION DETAILS AND DEMONSTRATIONS

In this section, we provide some visual implementations and demonstrations for the sake of better understanding, and describe

some scenarios which demonstrate objects and users and their behaviors when a role is given.

Fig. 7 A role that manages the door objects in hotel that is managed via Door Automation Solution which is defined to Hotel Z Suites – City X zone.

First things first, we create a role as in Fig. 7, and define the privileges as in Fig. 8. If we grant Read role, then the user will be granted any permission groups in terms of read type that is contained by Feature Door Command (such as Door_Read, DoorType_Read permissions in permission groups).

Fig. 8 Door Management Feature set with Door Command Feature that contains some permission groups (Door_C,R,U,D; DoorType_C,R,U,D) for the given role.

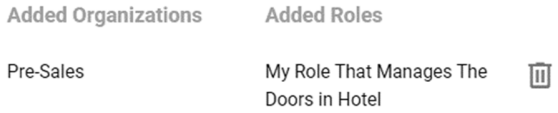

Fig. 9 The role is assigned to the user with Pre-Sales organization association

After the role is created, it is associated to organizations. An example UI is shown in Fig. 9 where the aforementioned role is associated with Pre-Sales organization and assigned to the user. Multiple organizations can be assigned to a single role of a user. The more association role has, the more accessibility role will have.

Fig. 10 depicts a scenario where User A is in Sales organization. The role is now granted to User A as if he is in Pre-Sales organization, although he is in Sales organization. There is a door which belongs to Pre-Sales organization, but owned by User A which is a very niche scenario, but possible. As the User A owns that door, even though that door is also contained by Pre-sales; and the User A is granted with User Level privilege, the User A can execute the given operation, that is Read operation. The user surely cannot modify, delete or create a door object as no privilege granted except for read (in Fig. 8). To conclude, no matter where the object is in terms of organization, if that object is owned by a user and that user granted with user-level privilege, then that object is accessible by mentioned user.

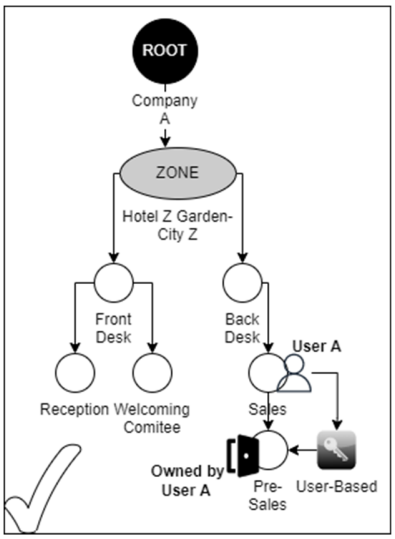

Fig. 10 User A is assigned with a role to Pre-Sales that includes User-Level Read privilege for a Door record that is owned by User A, but in Pre-Sales Organization.

Now we discuss the objects that is owned only by organization (no user ownership) which is the most coincided scenario in IoT world. As in the Fig. 11, User A that belongs to Sales organization is assigned with a role to Pre-Sales that includes Organization-Level Read privilege for a Door record that is owned by Pre-Sales Organization. So, this user can benefit that role to read any doors in Pre-Sales, and the door is in Pre-Sales as well, thus, read operation is successful.

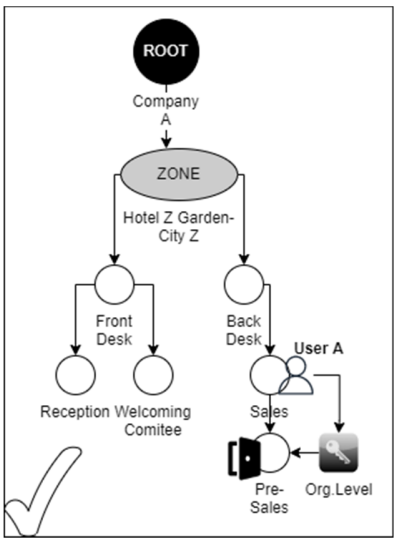

Fig. 11 User A is assigned with a role to Pre-Sales that includes Organization-Level Read privilege for a Door record that is owned by Pre-Sales Organization.

Fig. 12 shows a similar scenario with very slight difference, that is, role is associated with Back-Desk organization rather than Pre-sales in the previous case. In this case, User A can only see the doors in Back-Desk, however the door is still in Pre-sales organization. To sum up, there will be no read operation allowed.

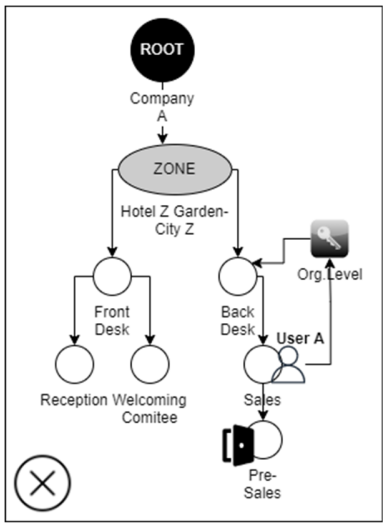

Fig. 12 User A is assigned with a role to Back Desk that includes Organization-Level Read privilege for a Door record that is owned by Pre-Sales Organization.

A slightly different but an interesting scenario is illustrated in Fig. 13. User A is now granted with the same role, but this time the door is owned by the same organization that he is in. Even though they are in the same organization, the user will not be able to see that door as the role is associated to Back-Desk. What matter is how the role is and assigned, not where the user is.

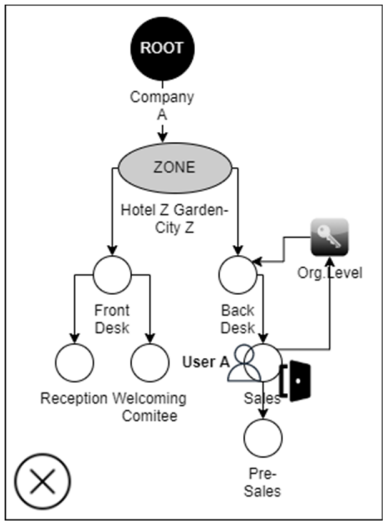

Fig. 13 User A is assigned with a role to Back Desk that includes Organization-Level Read privilege for a Door record that is owned by Sales Organization.

Now, we consider a scenario where the access level of the role is increased to “Organization and Its children (Top to down)” as in Fig. 14. This role traverses from the organization that is given to the below organizations by drilling down. In Fig. 14, viewable organizations are shown by light gray. This role grants user to view any doors in Back desk, Sales and including the door in Pre-Sales.

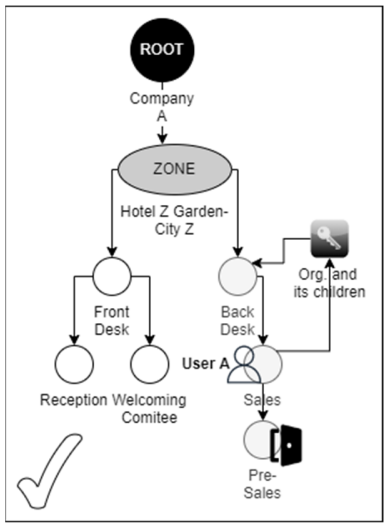

Fig. 14 User A is assigned with a role to Back Desk that includes Organization and Its Children level Read privilege for a Door record that is owned by Pre-Sales Organization.

In the scenario shown in Fig. 15, the access level of the role is increased to “All Organizations”. Now, all child organizations and zone organization can be read by User A in terms of door object. So, regardless of where the door is in the mentioned organizations, user can read that door object thanks to highest level of privilege.

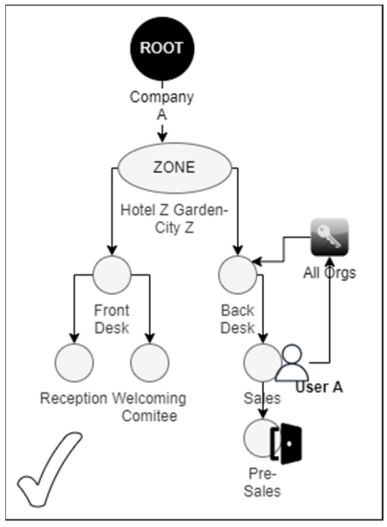

Fig. 15 User A is assigned with a role to Back Desk that includes All Organizations level Read privilege for a Door record that is owned by Pre-Sales Organization.

We have to note that, a company may include multiple zones as described before. In Fig. 16, we just zoom out and show other zones in the company. User A uses the same role, but this time the door is owned by another Sales organization that is under Hotel Z Suites City X. Even though the read privilege is “All Organizations”, this only covers all organizations in the zone that is addressed by the role. Roles are inherited from single zone organization as in Fig 7. Therefore, the door is not accessible by User A.

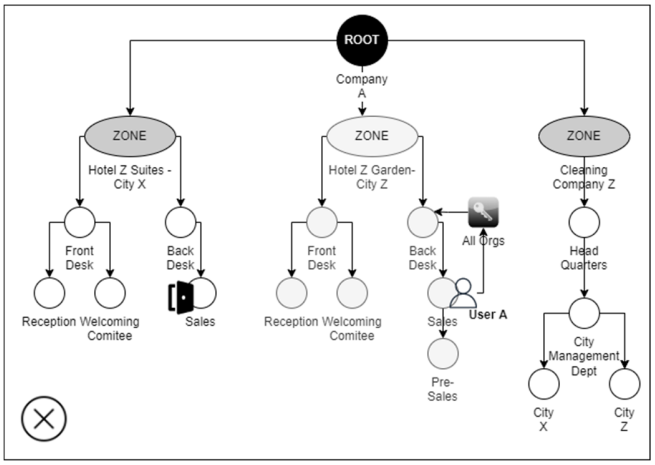

Fig. 16 User A is assigned with a role to Back Desk in Hotel Z Garden City Z that includes All Organizations level Read privilege for a Door record that is owned by Sales Organization in Hotel Z Suites City X.

Nevertheless, there is still an easy way to access that door simply granting another role that is inherited from Hotel Z Suites City X with Organization-Level Door_Read privilege as in Fig. 17.

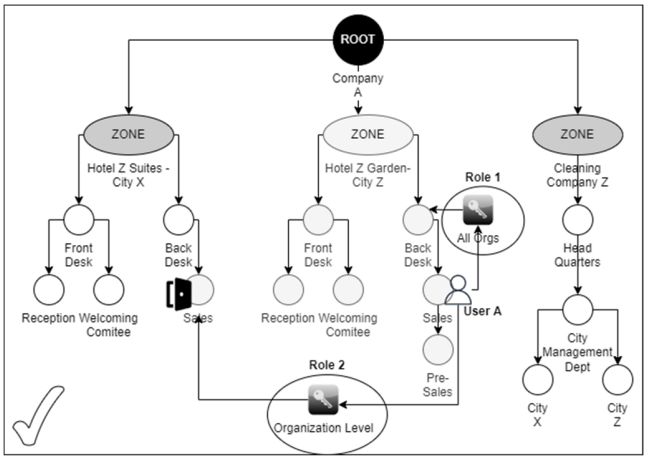

Fig. 17 User A is assigned with a role to Back Desk in Hotel Z Garden City Z that includes All Organizations level Read privilege and another role with Organization Level Read privilege for a Door record that is owned by Sales Organization in Hotel Z Suites City X.

Now, let us consider a scenario where the door is an object that presents in Door Automation solution and door automation solution is only purchased by Hotel Z Suites and Hotel Z Garden, not by Cleaning Company Z. In this case, the door object does not exist from the perspective of Cleaning Company Z. In other words, no door can be observed for Head Quarters under Cleaning Company Z zone, as well as the operations against door.

To make concise the approach of isolated organization, let us have an illustration to finalize this section. A critical door to security cabin should be isolated from any other roles that is assigned to users. Otherwise, propagating privilege levels (such as “All Organizations” or “Top-to-down” levels) enable access to them. But this door should be managed by only security users. So, creating security cabin as an isolated organization will prevent any propagated access by privilege levels. Thus, the door owned by security cabin organization is not accessible even with the highest privilege level namely “All Organizations”. In order to access to that door, User A should be assigned with a role associated directly with security cabin, not back-desk or sales or any other organization. Fig. 18 illustrates this approach. This niche scenario will work very well with such kind of approach.

Another useful tip about isolated organization is that they can present propagation property within each other. So, if we connect another security sub-cabin organization for security cabin, then any privilege with top-to-down to the main cabin can also infiltrates to that sub-cabin thanks to propagation property.

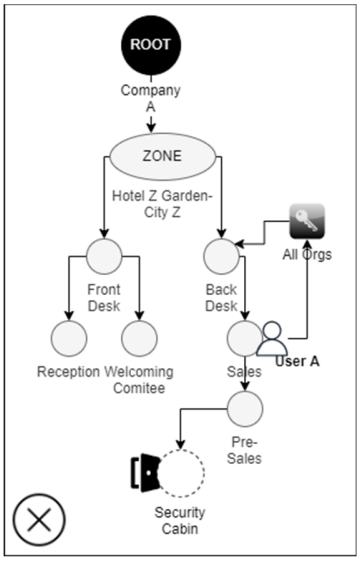

Fig. 18 User A is assigned with a role to Back Desk that includes All Organizations level Read privilege for a Door record that is owned by Security Cabin Organization.

These architectural designs can fit into user interfaces as given in Fig. 19. User SU is authorized to Solution X, Y and Z that are granted by roles no matter which zone it comes from. If user switches to Solution X, all the menus and features that are related to Solution X will appear in the page, and any requests (HTTP or else) originated from that solution page to microservices will carry a header namely “solution-id” which points the originated solution. So that, services can decide what to do with that request accordingly by authenticating or authorizing.

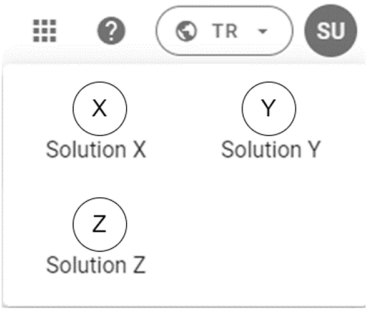

Fig. 19 User SU which is authorized to Solution X-Y-Z somehow with roles can see and switch to any solution from the User Interface

Not only UI, but also other services, APIs should include solution-id in each request regardless of what protocol is (HTTP or else). So that, regarding API can authorize the requestor with

the given solution. Fig. 20 depicts how third-party APIs can communicate with an API that serves for a solution.

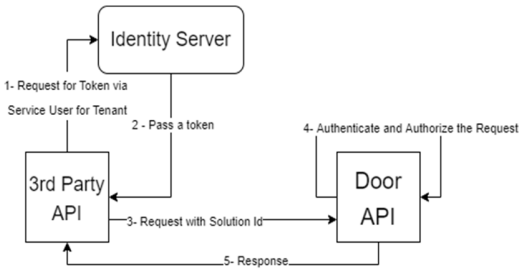

Fig. 20 3rd Party API authorizes itself to communicate with Door API in order to retrieve the doors that service user can.

If a third-party API is trusted and not required to be authorized with Service User and bypass all permission checks that are processed in the 4th step and solution ownerships, a "maximum-privilege" token can work for this purpose. Any API that uses maximum-privilege token can retrieve any objects that are in any solution and in any tenant in a single request. This is useful for some scenarios which also provides ease of usage and boosted performance.

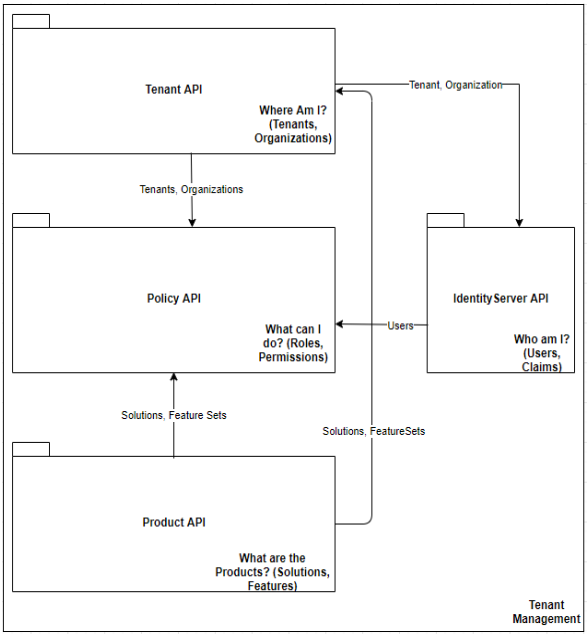

Fig. 21 API space and their responsibility diagram for a Multi-Tenant Management in IoT

The last but not the least, we briefly discuss API-Responsibility map shown in Fig. 21. In this map, different APIs are defined. Inbound arrows depict that given information is passed into inbounded API from outbounding API. Product API is responsible from solutions and features. Policy API is responsible from roles and permissions but aware of many objects (such as users, organizations and solutions) from other APIs. This approach simply derives from Domain Driven Design [14] namely shared objects in Bounded Contexts which is not a related topic to this paper. Tenant API is responsible from tenants and organizations, and Identity Server API is responsible from users.

V. CONCLUSION

In this study, we propose a robust, but flexible tenant management approach in terms of authorization and authentication. There are plenty of IoT solutions that can leverage the capabilities of traceability that IoT presents. Every solution that mounts over the IoT core brings more and more objects to be managed. It is really challenging to manage huge number of objects within a tenant in a secure way. Our approach significantly eases the management of these objects by defining roles, access levels, ownerships, and permissions in a very appropriate way. We believe that such a tenant management will mitigate significant amount of struggling and challenging operations against an IoT ecosystem. The proposed approach may work with the domains other than IoT as well, since it is designed as generic as possible. As a future work, this approach will be applied to other domains such as customer relationship management or inventory monitoring system.

ACKNOWLEDGMENT

This work is supported by Koç Digital R&D Center.

REFERENCES

[1] Bezemer, C. P., & Zaidman, A. (2010, September). Multi-tenant SaaS applications: maintenance dream or nightmare?. In Proceedings of the joint ercim workshop on software evolution (evol) and international workshop on principles of software evolution (iwpse) (pp. 88-92).

[2] Kalra, S., & Prabhakar, T. V. (2018, February). Towards dynamic tenant management for microservice based multi-tenant saas applications. In Proceedings of the 11th Innovations in Software Engineering Conference (pp. 1-5).

[3] Mace, J., Bodik, P., Fonseca, R., & Musuvathi, M. (2015). Retro: Targeted resource management in multi-tenant distributed systems. In 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15) (pp. 589-603).

[4] Tang, B., Li, Q., & Sandhu, R. (2013, July). A multi-tenant RBAC model for collaborative cloud services. In 2013 eleventh annual conference on privacy, security and trust (pp. 229-238). IEEE.

[5] Bien, N. H., & Thu, T. D. (2014, April). Hierarchical multi-tenant pattern. In 2014 International Conference on Computing, Management and Telecommunications (ComManTel) (pp. 157-164). IEEE.

[6] Hamilton, H., & Alasti, H. (2017). Controlled Intelligent Agents' Security Model for Multi-Tenant Cloud Computing Infrastructures. International Journal of Grid and High Performance Computing (IJGHPC), 9(1), 1-13)

[7] Levchenko, A., & Taratukhin, V. (2021, January). Reference business processes-based method for multi-tenant saas architecture deployment and adaptation. In 2021 28th Conference of Open Innovations Association (FRUCT) (pp. 264-270). IEEE.

[8] Bellu, R. (2018). Microsoft Dynamics 365 for dummies. John Wiley & Sons.

[9] McCLELLAN, M. (2001, June). Introduction to manufacturing execution systems. In MES Conference & Exposition, Baltimore, Maryland (pp. 1-7).

[10] Halawa, F., Dauod, H., Lee, I. G., Li, Y., Yoon, S. W., & Chung, S. H. (2020). Introduction of a real time location system to enhance the warehouse safety and operational efficiency. International Journal of Production Economics, 224, 107541).

[11] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (1999). Hypertext transfer protocol--HTTP/1.1 (No. rfc2616).

[12] Liu, K., Li, K., Peng, Q., & Zhang, C. (2019). A brief review on key technologies in the battery management system of electric vehicles. Frontiers of mechanical engineering, 14(1), 47-64.

[13] Alcaraz, C., Lopez, J., & Wolthusen, S. (2017). OCPP protocol: Security threats and challenges. IEEE Transactions on Smart Grid, 8(5), 2452-2459).

[14] Evans, E., & Evans, E. J. (2004). Domain-driven design: tackling complexity in the heart of software. Addison-Wesley Professional.