Using General-Purpose Instead of Domain-Specific Middleware Platforms for the Creation of an Ambient Assisted Living System

Kristin Aleksandrova [1]

[1] *Sofia University St. Kliment Ohridski, 15 Tsar Osvoboditel Blvd., Sofia, 1504, Bulgaria*

Abstract
With the extended life expectancy, we have seen an increase in the load put on each country's healthcare system. This has increased funding for technologies that could enable the autonomous living of elderly or disabled people. Various such technologies exist, and they can be summarized under the term of Ambient Assisted Living (AAL) Systems. The development of such systems is currently done by private healthcare facilities or research teams. It does not aim for commercial availability. There are several domain specific middleware platforms like universAAL, but in this work, we argue that general-purpose middleware platforms, like OpenRemote can be just as effective in the creation of an AAL system and at times even more cost effective. To illustrate that a prototype has been developed, that would be further extended to confirm or reject our hypothesis.

Keywords
Ambient Assisted Living, elderly care, smart home

1. Introduction

One of the challenges for many countries is the delivery of healthcare services, as currently personal care, nursing homes and hospitals prove to be both expensive and unable to handle the estimated number of people in the upcoming years. This is putting more focus on coming up with ways for elderly people to live by themselves, with minimised assistance from caretakers, family, or doctors. This is where Ambient Assisted Living (AAL) Systems [1] appears. Their sole purpose is to improve the independence and quality of life of people in need of assistance, whether that is in a nursing home or in their own home environment. AAL systems have shown immense potential to improve the quality of life of not just elderly people, but also of people with disabilities.

Information Systems & Grid Technologies: Fifteenth International Conference ISGT'2022, May 27–28, 2022, Sofia, Bulgaria
EMAIL: kristinia@uni-sofia.bg (K. Aleksandrova)

Looking at the current state of the AAL domain, certain system designs seem to stand out, as they find applications in the resolution of a variety of problems, posed by an AAL system [2]. Those designs implement data retention policies and allow the further analysis of stored data. This allows the growth of a system based on the end-user requirements, as in this specific area the desired software may not be what was initially designed. In turn, this poses the question of data harmonization and integration between various sensors, acting as data sources, control and input devices and notification receiving devices, all of which could be cloud or on-premises. Not to mention that for many systems it is crucial that the data sources are flexible. Meaning in one case we can use sources like room temperature and light levels and in another case. We would use motion detection sensors to track movement in the house. In both situations, we would then be able to apply the same principles, like predefined and custom rules, based on the source data, and machine learning algorithms looking for patterns.

This type of architecture is best defined and realized with the help of a middleware. Usually packaged as a platform [3], this is the central piece when connecting devices and scenarios to create a holistic AAL system. One thing to note here is that there are AAL specific middleware platforms and open-source ones that hold no domain knowledge for the needs of an elderly person. In this work, we will look at both options and consider the best approach for the creation of an AAL system. It is based also on the cost-functionality trade off.

2. Selecting middleware

As we already mentioned there is a significant differentiation between domain aware AAL middleware platforms and their open-source general-purpose counterparts. In this section, we will take two prominent examples of each type, UniversAAL and OpenRemote and illustrate parts of a much larger analysis on an optimal middleware, to be adopted for the creation of an AAL system.

2.1. Reference scenario

The foundation of AAL systems is the core idea of assisting elderly people in their daily lives, in turn silently and unobtrusively increasing their quality of life, independence and the visibility of their physical and mental state to their families of caretakers [4]. That on its own is proving to be a very broad mission statement, and in the real world, the vast difference in the application and target user groups of AAL systems illustrates that. Therefore, it is necessary that we take the time and define the target persona for the prototype that would be the result of this work. In addition, we need to define the exact challenges said persona will have and potential ways we will introduce to resolve them.

To start off, let us define an elderly person's autonomy. With age, many daily tasks prove to be a challenge or a life-threatening situation. For example, the lead cause of concern for elderly people living alone is undetected falls [5]. As the consequences can be dire and it is crucial, that a fall is recognized as soon as possible and the relevant authorities are notified, so we can mitigate the consequences. Therefore, even when we talk about elderly people living independently, we are taking into account that either there may be one or several persons, direct family members, neighbors or government provided help that are interested in said person's wellbeing and therefore regularly check up on them. The occurrence of this visitation varies from individual to individual but would generally be within once a day to once a week. In this work, we often refer to this concerned individual, as a caretaker.

Concerning what a caretaker would need to be monitoring, there are some straightforward questions and metrics to be taken into account:

- *Does the monitored person take their medication based on a pre-defined, approved schedule?* – In many cases the person can be confused, whether they have taken the needed dosage of their prescribed medication; there are two potential downfalls of this. In one case, the person decides not to take any more medication and therefore disrupts the medication effect, which in dementia patients, for example, could intensify the severity of their symptoms and in turn lead to more confusion about medication and general activities. This is a vicious cycle to that needs to be identified and interrupted by the caretaker, as they notice the increasing symptomatic. Alternatively, the other case would be that the person decides to take their medications again, to be on the safe side. Unfortunately, we cannot be sure that this would happen only once per missed dose. An elderly person could overdose on the prescribed medication within a few days or in mild cases over the course of several months. This could have severe consequences, again it would be the caretaker's responsibility to identify and address the issue, as no one else has a similar level of visibility of the mental and physical state of the elderly person.
- *Is the monitored person following a regular meal plan?* – This question poses similar concerns as the previous one, the person could be confused, whether or not they have eaten, what was in their fridge prior to meal-time and now. Naturally, this means similar repercussions to the previous concern. The elderly person can skip meals, causing weight loss and in turn, their moods, energy and immune system will be affected, or this could lead to overeating with the corresponding weight gain that could lead to complications in existing illness or lead to new ones. One example risk group would be elderly diabetics.
- *In case they are cooking, are all heat-generating appliances stopped after use?* – No further elaboration is needed here, as this could cause burn injuries, if the elderly person is distracted or in the worst-case scenario a home fire, that could put them, their family and neighbors at risk.

- *Are they sleeping an adequate amount of time?* – One common concern of elderly people is the lack of sleep. In most cases, this is a natural response to age and the different lifestyle, they have now adopted. If an AAL system is capable of monitoring their times of falling asleep and waking up, these concerns can be mitigated and most importantly, in case a real issue exists the system would not be biased and would be able to recognize it.
- *General monitoring of activities of daily living and more…*

These concerns, that can be easily addressed and improved with the AAL system, we have in mind are especially relevant for lightweight cases of dementia patients [6]. This also why in this prototype, they will be the main target audience and recipient of the AAL system monitoring and assistance functionalities.

2.2. Definition of comparison criteria and applying it to AAL candidates

Looking at the reference scenario, we can naturally derive the following criteria, many of which are also identified in similar research [7], [8], [9]:

1. Customization – measures the platform provided functionalities for creating a custom AAL system and supporting custom created devices.
2. Extensibility – measures the extensibility of an AAL system, created on said platform, regarding the addition of new supported scenarios and new devises and sensors.
3. Support of notification channels – the AAL system needs mechanisms to communicate with the users and especially the person's caretaker.
4. Security and Identity management – the AAL system has a multitude of personas related to each elderly person, we need to be able to model those personas with their access levels.
5. Data Persistence – in our prototype we aim to provide additional insights in a person's wellbeing based on derived insights from historical data, this requires sensor data to be securely collected and stored.
6. Scheduling and automation – it is important that when certain criteria are met it is possible to automate a response from the system.
7. User-defined rules and triggers – to have a fully functioning system, we need to be able to provide user-defined rules describing the person, being monitored, as everyone has their own habits and behaviors.
8. Data processing and ML algorithms training – to be able to derive insights on historical data, we require the system to support said data processing and model training.
9. Cost optimization – the overall solution must not be too expensive to install and maintain, including the types of devices it works on or we risk no future adoption.

In the next subsections, we will consider how the two prominent candidates abide by said criteria and come to a decision on which to base the target prototype.

2.2.1. UniversAAL

UniversAAL [10] is an open-source IoT platform that originally was developed as part of an EU-funded research project. As shown by the name, this middleware was designed to assist the development of AAL applications and in turn the growth of the field [11]. As such, it is currently published under Apache GPL license. UniversAAL is written as a distributed middleware in Java using the OSGi. It provides a skeleton of modules and concepts, upon which you can build a productive system. With universAAL it is possible to model a variety of scenarios and ensure their interconnectivity, including a variety of different user interfaces, which also ensures a high extensibility, as also required by our second criteria. All of this is possible due to the multi-level conceptualization that is implemented in universAAL. This in turn makes the process for building applications dependent on the progress made in understanding the foundational concepts in the platform. The ramp-up process on this platform is not a quick and simple one, and it is worth to mention, that without previous knowledge on the platform, any prototype's initial version will be delayed.

Additionally, the number of supported devices is extended regularly, and the most prominent communication protocols are utilized. In that sense, the cost of adding and removing devices in an already developed system is very low. As with other systems that offer this level of device supportability, it is possible to create and use custom devices for our system. The only requirement for their seamless integration into universAAL is the support of one of the communication standards KNX, ZigBee or FS20 [12]. Technically the middleware does not communicate directly with the devices with specific protocols; instead, we utilize the Context Bus. As part of the application, we develop Context Providers, which gather information about devices state. Either this is done directly or as applications that combine contextual information to derive a custom reading. That input is used by Context Publishers, which are applications that send data over the Context Bus, transformed as a Context Event, to fit the ontological models in place. For example, upon turning on the kitchen light, the Context Provider, which is measuring the brightness in the room will create a Context Event with the <subject, predicate, object> format, in this case <KitchenLight, hasValue, 100>. This event would then be published via the Context Publisher to the Context Bus and then consumed in the main application. Understanding these types of concepts and how to implement properly them, is one of the main hurdles in the prototype development, based on this middleware, as other options while not as powerful in the implementation, have a simpler structure for development and device onboarding.

UniversAAL reuses most of the underlying systems security, provided by Java and OSGi, and builds additional functionality to ensure confidentiality and integrity as a primary focus [13. User sessions can be two types, the classic sense of a device bound session or location-bound sessions, that are valid for a predefined location for the user and includes all devices there. It is also possible do model specific user roles and permissions for restricted access to certain functionalities, as well as consent management, the latter is crucial if we would like to create a prototype abiding by European Union's regulations for data protection [14], that operates on the premise of actively given consent by its users. Additionally, the universAAL platform offers encryption for secure service-to-service communication and node authentication. This is achieved by grouping services in uSpaces and provisioning a key per service to allow secure communication with the other services in the group.

Data persistency in universAAL is both restrictive and promising for the development of machine learning models on top of. This is because, as we illustrated before, data is stored as Context Events, or in other words as instances of the ontology model. If we go for any ontology-based algorithm that is perfect as there is no data processing required and the data access and storage is optimized for this type of algorithm. However, if we would like to rely alternative algorithms, like neural networks, it will be out of the question to transform large volumes of data, as we have neither the time, the space, nor the processing power for such a costly operation. Alternatively, we can modify each Context provider, to record his or her data not only as a Context Event, but also as an entry in a database. This defeats the purpose and optimization of the universAAL platform, not to mention it introduces double maintenance of data. It transposes to all operations. It could lead to inconsistencies. At present, we possess no information on the performance of each type of model. In addition, do not process what would be a preferable option. In case, we decide on an ontology-based approach universAAL would be the most likely choice.

One of the main benefits of choosing universAAL undoubtedly is its focus specifically on assisting the creation of AAL applications. This includes the already created ontologies, describing daily activities, devices, rooms, etc. with their relationships. Considering our goal of creating a machine-learning algorithm, that aims to enhance an AAL system with insights and potential rules, triggering actions based on those insights, those ontologies could be immensely helpful. Of course, with the AAL use case in mind the security aspect of the platform covers precisely what we defined with the added bonus of authorization options for people who find difficulty relying on basic authentication, namely the location-based authorization for devices. However, we cannot overlook the effort needed to understand all the concepts required developing an application based on the universal. This is coupled with the volume of the development. It is a significant part in the high cost for installation and maintenance. A working prototype would take more time and expertise then the previous options to be developed and customized across different users.

2.2.2. OpenRemote

OpenRemote[2] is an open source IoT platform that prioritizes offering a simplified approach to connecting different sensors and devices in a network, that then can be managed via mobile or web applications. The platform is developed following a micro service-like architecture, with each component being hosted in its own Docker container. There is a main manager application, which is the central component that ties the platform together, the database is PostgreSQL and security is handled by Keycloak[3].

The manager component also implements and handles the user interface. It is quite straightforward; there are four main screens, which highlight most of the provided functionality. The map shows the location of each device and is an interactive way to check the value of each one. Devices and sensors are referred to as assets and can be configured in the asset screen, where also different online agents can be configured to provide input to the system. There is also the possibility to create custom rules in several ways. Lastly, the insights tab shows a custom dashboard of metrics and analytics, aggregated to a desired level. Due to the distributed nature of the platform, in order to modify the application and add custom functionality, it is not needed to modify the source code. There is a convention defined, upon which by providing a few modifying files in a structured manner and customizing the Docker compose file, the application's behavior is modified and customized. This would significantly reduce the initial costs for development and the costs for customizing the application for each user, as we would need to show a customized location for example, and the costs for installation and distribution.

By the same logic, we can also handle extensibility. The easy way to provide modifications leads to the possibility of adding additional functionality as an additional container. Or by modeling and modifying the existing ones, for example according to the development guides provided by the platform, there are several alternative behaviors available, and the choice can be made in the docker compose file, that controls the provisioning of the containers. In addition, there are predefined device types, which allow a plug-and-play compatibility, as for everything else a wide range of protocols are supported, like Bluetooth Mesh, HTTP, KNX, LoRa, MQTT, SNMP, TCP, UDP, Velbus, Websocket, Z-Wave, etc.

We already mentioned the available functionality for creating user-provided rules. The main benefit in OpenRemote's approach is the usability of the solution by people with different technical background. There are human readable rules of the type when then, which use the preconfigured assets to create simple actions or notifications. This can be very easily used by the family or caretaker of the assisted person to quickly define the risky and potentially dangerous situations or behaviors and take measures to prevent them. Additionally, rules can be defined as a Flow

[2] https://openremote.io

[3] https://www.keycloak.org

model, which includes aggregating data from several assets, performing operations on it and as a result modifying the state of a different asset. For example, we can create a flow model, that takes the temperature in every room of the house and sends a notification to the caretaker, when the aggregated home temperature falls or raises with more than 3 degrees, as this could be pointing to a forgotten open window, or running kitchen appliance, depending on where those sensors are situated. Talking about notifications, the out-of-the-box supported method is notification via mail or by sending alerts to the connected mobile application, the latter of which would require that we expose some parts of the system to the internet.

Security in OpenRemote is based on Keycloak. It allows for multi-tenant authentication. While this was not in our initial criteria, this is a very interesting functionality to consider, as it would allow to provide one central server for the system for several households in close proximity, as this would reduce the cost of hardware for processing and in turn the cost of installation. In all cases, we are treating fully cloud applications as the least favorable option, due to the higher risk of comptonization of the data security and privacy. In addition, we have the standard TLS/SSL communication by a HAProxy-based reverse proxy. Lastly, as we defined in the criteria, it is also possible to create different roles, with the corresponding to the persona access levels.

As we already mentioned, the database that this solution is relying on is PostgreSQL. It is its own Docker container and the stored data is available in a dedicated volume. This makes the data backup, restore, and move very easy, as it relies on the underlying Docker functionality. The relational database allows us flexibility, when implementing different machine learning functionalities, as it is not restrictive towards the approach we could use. As for the algorithms themselves, a good option would be the creation of a separate Docker container in the same virtual network, to service the training and answer on predictions via API calls.

To summarize, OpenRemote is satisfying a good portion of the defined criteria and is showing a lot of potential for customization and extensibility. The costs for development and maintenance are significantly lower, than the previous options. The main downside now is the supportability of notification channels, as now we can send emails or notification, if we create a dedicated application for our system. One way to go around this issue would be to involve an additional software product that then dispatches the alert to the target notification channel based on the received email.

2.2.3. Conclusion – candidate selection

As a summary, we see that the universAAL platform relies much more on homegrown resolutions to different problems, like security. While tailored closer to the AAL domain model, the question of the supportability of those models arises; additionally we have always the concern if the current implementation follows the state-of-the-art of the field. One way to be reassured of that is to have a decent sized development community of the open-source solution that regularly uses and maintains the original repository. In this case, universAAL is at a disadvantage, as OpenRemote has much wider adoption, due to its open range of applications.

Additionally, the entry barrier of universAAL is quite high, with the range of homegrown concepts and abstractions, it takes significant time to familiarize yourself with the concepts and start the development of a solution. In turn, this means high cost and effort for the initial implementation, any further extensions, the maintenance and supportability of the end solution and the long-term approach for the solution. On the other hand, OpenRemote has a simplified approach to development and distribution, essentially running as a docker container and reducing all regular maintenance operations for the system availability, backups and OS supportability, to operations on Docker containers.

Both universAAL and OpenRemote have their significant advantages and costs optimizations and are acceptable candidates for the creation of an AAL system. However, we cannot ignore the high effort and costs, required by universAAL, as the goal of this work is the creation of a prototype, in an optimal trade-off between functionality and costs. In that sense, OpenRemote fully facilitates the desired scenario and provides opportunities for further development and extension, at a reasonable trade-off. Therefore, in this work we will build a prototype based on OpenRemote that would exemplify the functionality and potential of this type of systems.

3. Prototype development

As we already mentioned, OpenRemote is easily extendable, as it is Docker based. To start, we can pull and run the system as Docker containers. There are four in total. Afterwards, a significant level of customization can be done via the Docker-compose file that contains the information for the container provisioning. Modifications are declared, according to the following folder structure:

```
deployment
|-- manager
| |-- app
| | +-- images
| | |-- manager_config.json
|-- map
| |-- mapdata.mbtiles
| |-- mapsettings.json
```

Where, the manager_config.json contains all the modifications to the styling of the UI, all colors, fonts and logos can be changed via this file, and images are stored in the adjacent images folder. Additionally, the styling can be changes based on the realm in which we are currently working, in turn creating higher customization between users of the same instance of the system.

The map folder holds the mbtiles-file, that is the map of the city or area we would like to visualize and work against, the maps can be served by a separate map container, when they are derived from raster images, for example a home's floor plan and is defined in a different manner. The mapsettings.json contains information about the provided map, such as the levels of zoom we would like to allow, the center point of the map, etc. Important to note is that similarly to the styling, we can have a different central point for each realm, which allows us to define a different household for each realm and focus on it in the starting screen, based on the logged in user.

3.1. Architecture

OpenRemote's architecture is based around the so-called Manager, that is a headless Java application, that captures the current state of the system. The specific scenario, in our case the various system capabilities for monitoring and assistance of elderly people, are modeled via the manager and the different assets with their attributes. This allows the initial modeling of a system that manages a home and later-on extensions in the direction of wandering prevention or location-based functionalities around a smart-aware city. Devices are connected to the manager via Agents, which is the combined name for the interface to service protocols, external APIs, and custom solutions (Figure 1).

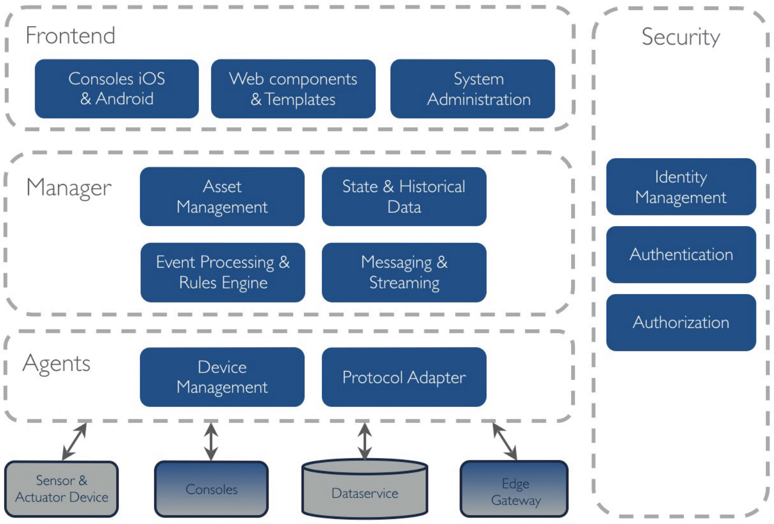


Figure 1: OpenRemote architecture, part of the official OpenRemote documentation

The frontend handles the creation and deployment of user interfaces, both mobile and web-based ones. These interfaces are referred to as consoles, and different functionality and rules can be defined, based on the console, as out-of-the-box geofencing and push notifications are implemented. OpenRemote supports multitenancy and the frontend reflects that by supporting multi-tenant dashboards and control panels.

On the more hands-on side, OpenRemote is distributed as a group of Docker containers. One is the authorization container, named openremote_keycloack and all authorization requests are handled by it, as expected there is a separate container openremote_postgresql, where the database is. The corresponding volume in Docker holds all the data and we can perform back-ups on a volume level. The openremote_proxy, as suggested by the name handles the proxy and the openremote_manager is responsible for the user interface of the system. All containers are created in an isolated network, and communication between them is restricted.

3.2. Functionality

One functionality that we have adopted from the OpenRemote system is the concept of realms. This is the human-readable visualization of multitenancy. Each realm is essentially a different tenant that can have its own style, custom map, assets, and users. As such, this prototype suggests one local instance of the OpenRemote AAL-based system that could serve a whole neighborhood. This reduces the computational power needed for additional data processing of the data and allows an anonymized analysis of the similar behaviors of people, as well as the causes and effects of each behavior.

Currently the prototype is equipped with a custom map of Sofia. There are several sources, which can provide a current vector map of a city with a free to use license and one of them is Google Maps. In this case, that is the source of the MBTILES map, which we are using. For example, in the test realm called Ivan's home, we are monitoring the household of Ivan and the specific use-cases he has – monitoring medication intake and kitchen electrical appliances (Figure 2).

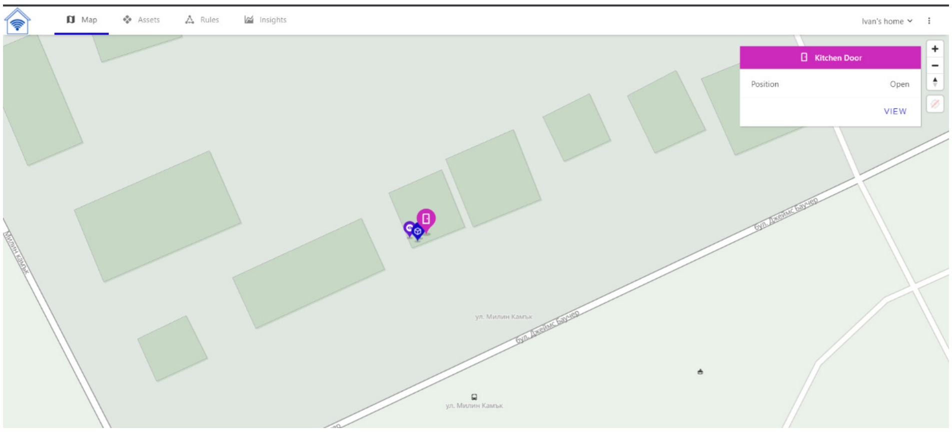
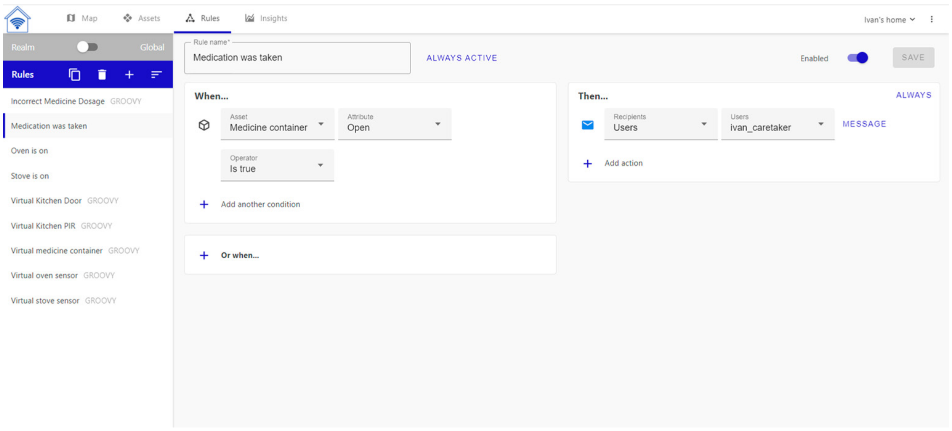


Figure 2: Sample realm – Ivan's home in the first version of the prototype

To implement the two mentioned use cases, several assets need to be modeled. To measure the medicine intake, we assume that the correct dosage of the medication is measured and stored in a dedicated medication box by Ivan's caretaker. This is usually necessary on a weekly basis, afterwards twice a day, as prescribed Ivan opens the box and takes the next dose of medication. In this case, we have a sensor on the box that gives us information when it is opened. The naïve resolution of the problem would be to send a notification each time the box is opened. This would require Ivan's caretaker to manually count and monitor the number of times Ivan has taken his medication and the timeframe between doses (Figure 3). Instead, in the prototype, we have created a groovy script, that monitors the box's sensor, and measures, how often is the box opened and does that fit to the prescribed medicine schedule. In case that is not true Ivan's caretaker is notified. Additionally, the opening of the box for refiling, must not conflict with the current calculation of taken dosages. To go around this, the current designed process requests that the caretaker disables the Groovy rule doing the calculations for the duration of the refilling and re-enables it afterwards.



Incorrect Medicine Dosage GROOVY
Medication was taken
Oven is on
Stove is on
Virtual Kitchen Door GROOVY
Virtual Kitchen PIR GROOVY
Virtual medicine container GROOVY
Virtual oven sensor GROOVY
Virtual stove sensor GROOVY

Rule name* Medication was taken
Enabled
Asset: Medicine container
Attribute: Open
Operator: Is true
Recipients: Users
Users: ivan_caretaker

Figure 3: Rules for medicine dosage in the realm Ivan's home

As also seen in Figure 3, in the current version of the prototype, all sensors are virtual; this is again achieved via groovy scripts. This also allowed the modeling of the second scenario – monitoring unattained kitchen appliances. This is far from a trivial question and to illustrate that let us consider the following example. Ivan goes into the kitchen around lunchtime and turns the oven on; he prepares the meal and mind and after 20 minutes moves the dish into the oven. Afterwards, he leaves the kitchen and the dish to cook for an hour. Compare this scenario with the following: Ivan goes into the kitchen around lunchtime and turns the oven on, he changes his mind meanwhile and prepares something that requires no heat, and as such, he forgets the oven on and leaves the kitchen. Recognizing the difference between those two cases and similar ones requires detailed sensor data, analyzing where Ivan is relative to the kitchen, the time that has been spent and most notably for the given example – has the oven door been opened. Now consider the same two examples with the stove: there is no door there to monitor, and in most cases, we have older appliances in elderly people homes, that do not recognize if the stove is in use and the question of additional sensors like heat, presence or pressure raises the cost of the solution. In those cases, the cheapest approach is to interact with the user, i.e. to ask if he is using the stove; is he forgotten it; etc.

4. Conclusion and future work

In conclusion, the first OpenRemote prototype was developed with a specific target persona in mind – elderly people that exhibit early stages of dementia and as such have manageable symptoms, which allow them to continue to live independently with some support by a caretaker. Without much development effort, we were able to onboard several relevant scenarios for AAL systems. This is promising for the future development and growth of the system. It definitely confirms our initial hypothesis, that we do not need a domain specific AAL middleware platform to be able to create quickly a system that would cover a wide range of scenarios in elderly care.

As mentioned in the previous section, currently the defined scenarios are simulated with virtual sensors. The next step several homes from the CASAS[4] dataset will be used to model different realms and homes with closer to real-life sensors, this would also open the possibility to use the analytics dashboards, that come with OpenRemote and start looking for data anomalies and potential leads for behaviors that can be recognized with the help of an algorithm.

Similarly, as we already defined in section 2.1, where we described the reference scenario, when talking about the daily life of an elderly person, there are much more scenarios to be considered. Also many behaviors to be modeled, not to mention each one differs from person to person and to truly grasp the potential of such a prototype it is necessary that we extend and showcase as many additional scenarios as possible.

Now the model differentiates between three types of users – the system administrator, the elderly person and their caretaker. However, in the AAL domain there are many different personas, including medical personnel, family members, friends, neighbors, caretakers that may or may not have medical qualifications, etc. Each one of them interacts with the system in a different manner and needs access to different resources. Obviously medical data is restricted to only the responsible for the elderly person doctor and/or nurse. OpenRemote allows the modeling of all those different personas and the restriction of read and write access for each asset in the system. Naturally, this would be the next step in the prototype evolution.

Lastly, the original idea behind the prototype creation was to employ machine-learning algorithms on the gathered data from the different devices and sensors and to establish a baseline behavior for the individual, so that the system can recognize outliers in the person's behavior and provide timely notifications to the person's caretakers. To do so we will create an additional Docker container. It will act as a server, where the models are trained. Each new sensor input, we would like to evaluate will be sent via an API to the server, where it will be evaluated and the response will be returned to the OpenRemote manager where a custom agent will record the prediction and if needed evoke the necessary notification channels or reaction protocols.

[4] http://casas.wsu.edu/datasets

5. Acknowledgements

Theresearchofthisauthorwassupportedby ProjectBG05M20P001-1.002-0011 “Centre of competence MIRACle – Mechatronics, Innovation, Robotics, Automation, Clean technologies” (https://miraclebg.com)[5].

6. References

[1] Queirós, Alexandra & Santos, Milton & Dias, Ana & Rocha, Nelson. (2019). Ambient Assisted Living: Systems Characterization. 10.1007/978-3-319-91226-4_3.

[2] Byrne, Caroline & Collier, Rem & O’Hare, Gregory. (2018). A Review and Classification of Assisted Living Systems. Information. 9. 182. 10.3390/info9070182.

[3] Toutsop, Otily & Kornegay, Kevin & Smith, Edmund. (2021). A Comparative Analyses of Current IoT Middleware Platforms. 413–420. 10.1109/FiCloud49777.2021.00067.

[4] Zhang, Shuai & Nugent, Chris & Lundström, Jens & Sheng, Min. (2018). Ambient Assisted Living for Improvement of Health and Quality of Life—A Special Issue of the Journal of Informatics. Informatics. 5. 4. 10.3390/informatics5010004.

[5] Yazar, Ahmet & Erden, Fatih & Cetin, A. (2014). Multi-sensor ambient assisted living system for fall detection.

[6] YILMAZ, Özgün. (2019). An ambient assisted living system for dementia patients. Turkish Journal of Electrical Engineering & Computer Sciences. 27. 2361-2378. 10.3906/elk-1806-124.

[7] Zentek, Tom & Yumusak, Can & Reichelt, Christian & Rashid, Asarnusch. (2014). Which AAL Middleware Matches My Requirements? An Analysis of Current Middleware Systems and a Framework for Decision-Support. 10.1007/978-3-319-11866-6_9.

[8] Amina El murabet, Anouar Abtoy, Abdellah Touhafi, Abderahim Tahiri, Ambient Assisted living system’s models and architectures: A survey of the state of the art, Journal of King Saud University – Computer and Information Sciences, Volume 32, Issue 1, 2020, Pages 1–10, ISSN 1319-1578, https://doi.org/10.1016/j.jksuci.2018.04.009.

[9] Phull, Rajjeet & Liscano, Ramiro & Mihailidis, Alex. (2016). Comparative Analysis of Prominent Middleware Platforms in the Domain of Ambient Assisted Living (AAL) for an Older Adults with Dementia (OAwD) Scenario. Procedia Computer Science. 83. 537-544. 10.1016/j.procs.2016.04.252.

[5] https://miraclebg.com

[10] Ram, Roni & Furfari, Francesco & Girolami, Michele & Ibáñez, Gema & Lazaro, Juan & Mayer, Christopher & Prazak-Aram, Barbara & Zentek, Tom & Tecnológicas, Soluciones & Salud, La & Bienestar, El. (2013). universAAL: Provisioning Platform for AAL Services. Advances in Intelligent Systems and Computing. 219. 10.1007/978-3-319-00566-9_14.

[11] Gorman, J. & Mikalsen, Marius & Stav, Erlend & Walderhaug, Ståle. (2010). universAAL – European Commission collaborative research and development to develop an open architecture and platform for Ambient Assisted Living (AAL). Gerontechnology. 9. 183–184. 10.4017/gt.2010.09.02.167.00.

[12] Jonas, Karl & Vogl, Bastian & Rademacher, Michael. (2017). Security Mechanisms of wireless Building Automation Systems. 10.18418/978-3-96043-044-5.

[13] Alejandro Medrano. (2017). Security Overview. https://github.com/universAAL/platform/wiki/Security-Overview Last accessed 07.04.2022.

[14] EU General Data Protection Regulation (GDPR): Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation), OJ 2016 L 119/1.

[15] OpenRemote. (2021) Official Documentation. https://github.com/openremote/openremote/wiki Last accessed 07.04.2022.