

Received December 17, 2020, accepted December 31, 2020, date of publication January 5, 2021, date of current version January 15, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3049444

Involving Stakeholders in the Implementation of Microservice-Based Systems: A Case Study in an Ambient-Assisted Living System

GASTÓN MÁRQUEZ¹, CARLA TARASCO², HERNÁN ASTUDILLO¹, (Member, IEEE), VINCENT ZALC³, AND DAN ISTRATE³

¹Departamento de Informática, Universidad Técnica Federico Santa María, Valparaíso 2390302, Chile

²Escuela de Ingeniería Civil Informática, Universidad de Valparaíso, Valparaíso 2362905, Chile

³Laboratoire BMFI UMR 7338, Université de Technologie de Compiègne, 60203 Compiègne, France

Corresponding authors: Gastón Márquez (gaston.marquez@usm.cl) and Carla Taramasco (carla.taramasco@uv.cl)

This work was supported in part by the ECOS Sud grant BV2 (Well Being and Well Ageing), in part by FONDECYT Regular (Multimodal Machine Learning approach for detecting pathological activity patterns in elderlies) under Grant 1201787, and in part by ANID PIA/APOYO (CCTVal) under Grant AFB180002.


ABSTRACT Microservice-based systems promote agility and rapid business development. Some features, such as fast time-to-market, scalability and optimal response times, have encouraged stakeholders to get more involved in the development and implementation of microservices architectures in order to translate their business vision into the implementation of the architecture. Although some techniques allow the inclusion of the stakeholders' perspective in the design of microservice architectures, few proposals consider such perspectives in the selection and evaluation of technologies that implement microservice architectures. Indeed, the qualities that characterize microservice-based systems strongly depend on the suitable selection of technologies, such as application frameworks and platforms. This article proposes a collaborative technique that includes stakeholders and software architects in the selection and evaluation of application frameworks and platforms to implement microservice-based systems. We evaluated the technique in an industrial case of design and implementation of an Ambient-Assisted Living (AAL) system, which combines microservice architecture and Internet-of-Medical-Things (IoMT) sensors. The case results indicate that the proposed technique supported stakeholders in the pragmatic evaluation of alternative technological solutions. Additionally, it allowed the implementation of an AAL system that satisfies the quality specifications of stakeholders and end-users. This initial study suggests that actively including stakeholders in the implementation of microservice-based systems allows architects to make design decisions that better consider stakeholders viewpoints as well as managing their expectations.

INDEX TERMS Software architecture, microservice architecture, ambient-assisted living system, frameworks.

I. INTRODUCTION

People and organizations interested in a system are commonly defined as *stakeholders*. A stakeholder is anyone who has an interest in the success of the system, e.g., customers, end-users, developers, project managers, maintainers, and even those who market the system [1]. But, although all stakeholders share the same desire for the project success, they have different needs and viewpoints that they want the

system to guarantee, optimize, or deliver to a target audience. Depending on the context in which the system is involved, these needs encompass different types and levels of concerns. In this regard, some studies (such as [2] and [3]) have investigated the role of stakeholders regarding decision making in software development. These studies conclude that stakeholders significantly influence decisions in the early stages of the software life cycle, specifically in the processes of capturing and eliciting requirements. On the other hand, they also mention that stakeholders provide essential information regarding the constraints and qualities that the system

The associate editor coordinating the review of this manuscript and approving it for publication was Resul Das .

must satisfy. These aforementioned constraints and qualities are often represented by quality attributes requirements (QARs). QARs describe the properties and characteristics that the system must satisfy such as usability, security, performance, reliability, performance, among others [1].

Despite the significant influence that stakeholders have regarding the success in developing and implementing software, they reduce their participation as the software life cycle progresses. Most of the design and implementation decisions fall on experts' judgment (such as architects and technical leaders), which implies that stakeholders' viewpoints are barely considered [4]. The reasons why this scenario occurs can be varied; however, Schulenklöpper and Rommes [5] describe that one of the main causes of the limited participation of stakeholders is related to the lack of adequate tools the architects have to communicate with them.

Concerning the development and implementation of *microservice architectures*, the aforementioned scenario is not different. The microservices architectural style aims to develop and deploy software in small and distributed services, each running autonomously and communicating with each other, for example, through HTTP requests to their APIs [6]. Microservice architectures promote the creation of more adaptable and flexible IT infrastructures. Each service can be deployed and modified without affecting other services or functional aspects of the system [7].

Several studies have investigated the stakeholders' profiles and roles that participate in the development of microservice-based systems. In this regard, Haselböck *et al.* [8] identify five stakeholders profiles, which are software architect, developer, application engineering, quality assurer, and manager. Rademacher *et al.* [9], on the other hand, identify three stakeholder roles, which are domain expert, service developer, and service operator. Descriptions of current microservice-based projects (such as [10]–[12]) also show that stakeholders participation is high in the modeling, design and analysis development phases, but low in the implementation phase, where technical experts and architects often make decisions. Hence, one of the main issues we know about this regard is a lack of techniques to involve stakeholders in decision-making about selecting technological tools (such as frameworks and platforms) to develop and implement microservice-based systems. Moreover, the evidence describes that evaluating technologies is led mainly by technical experts and architects; still, it is unclear how they map the QARs defined by the stakeholders into the implementation of microservice-based systems.

This article introduces a technique to include stakeholders in decision-making for the implementation of microservice-based systems. The technique extends μ Azimet, a technique that uses architectural knowledge to support architects in the analysis, evaluation, and comparison of technologies (represented by framework and platforms) to satisfy QARs in microservice-based systems [13]. Our proposal aims to change the procedure of μ Azimet input selection. μ Azimet inputs are *properties* that characterize quality

attributes. Therefore, instead of an architect selecting these properties, we propose to extend the selection of properties through a collaborative process that includes stakeholders. Likewise, the inputs selected in this collaborative instance are used as evaluation criteria to analyze solutions to implement microservice-based systems. We evaluated our proposal in an Ambient Assisted Living system that uses Internet-of-Medical-Things (IoMT) and microservice architecture. The contributions of our study are as follows:

- A collaborative technique that allows stakeholders to participate in architecture decisions by prioritizing quality attribute-based properties representing QARs.
- A mechanism that enables stakeholders and architects to evaluate frameworks, platforms and tools in order to satisfy QARs in microservice-based systems.

This article is organized as follows: Section II contextualizes the problem addressed in our research; Section III describes the related work; Section IV introduces μ Azimet; Section V describes our technique; Section VI details the case study; Section VII discusses the threats to validity; and Section VIII concludes the article and describes future work.

II. PROBLEM STATEMENT

Stakeholder participation in software development is a critical factor in achieving the expectations that software must satisfy [1]. In this regard, Hujainah *et al.* [14] argue that stakeholders play an important role in prioritizing software requirements during the requirement elicitation process. Although the prioritization is often influenced by project costs, deadlines, and government policies, usually stakeholders prioritize requirements based on their own technical knowledge. This experience that each stakeholder has positively influences the prioritization of requirements, but prioritization affects not only requirements but also other software development disciplines. Ernst *et al.* [15] mention that stakeholders participation is not only essential to develop software, but also to design its architecture, define the architecture roadmap, and execute the roadmap. Indeed, stakeholders participation in the prioritization of software requirements not only supports software architects but also affects the characteristics of the software product [1].

Although there are several techniques to involve stakeholders in prioritizing software requirements (such as [16] and [17]), it is still hard to involve them in the decision-making of software architectures. Schulenklöpper and Rommes [5] attribute this gap to the communication problems between architects and stakeholders. More precisely, the authors state that “the architect’s tools for communicating with stakeholders are blunt and often unsuitable”.

Regarding service-oriented systems, Shekhovtsov *et al.* [18] describe two observations about stakeholders and the implementation of service-oriented architectures:

- It is difficult for stakeholders without IT experience to express their expectations on the quality of a system under development if they cannot experience it in the appropriate context; without such experience, they are

forced to be rough in their opinions (e.g., “the service must be reliable”).

- There are not established processes and tools for quality-related interaction between business stakeholders and IT people.

If we extrapolate these observations to microservice-based systems, there is not much difference. As the market of tools, platforms and frameworks grows, new technical knowledge makes it challenging to include stakeholders in the decision-making of microservice architectures, as well as in the prioritization of requirements. Stakeholders, therefore, reduce their participation in the decision-making concerning the implementation of microservice-based systems. This lack of participation implies that the stakeholders’ viewpoints have no influence on the implementation of critical business microservice components (such as APIs and business services) compromising not only the stakeholders’ expectations about the system but also the functionalities that the system offers to clients and end-users [19].

III. RELATED WORK

This section introduces the studies that have addressed the evaluation and selection of software components (including technologies such as frameworks and platforms) in software architecture and microservices. Additionally, we describe studies that have examined the role of stakeholders in the evaluation of software components.

A. SOFTWARE COMPONENTS SELECTION AND EVALUATION

Astudillo *et al.* [20] propose an approach to systematic generation, evaluation, and comparison of component assemblies, using potentially incomplete, imprecise, and changing descriptions of requirements and components. The authors propose using architectural policies and mechanisms obtained from distributed systems to propose a technique that generates alternatives based on software components to implement architectures. Like μ Azimet, the authors’ approach focuses on generating systematic solutions. Nevertheless, this technique does not consider stakeholders and is not focused on microservice architectures.

Lenarduzzi and Sievi-Korte [21] present an approach based on three steps to support the developers in selecting alternative components in case the component is not working anymore, or future updates bring mismatches in versioning, which forces to select others alternative. The approach considers the following steps to select component alternatives: (i) identification of components; (ii) exploration survey; and (iii) the use of a component adoption model. However, this approach it does not focus on the design of microservices architecture. The approach supports developers in evaluating and selecting components, but does not address stakeholders’ needs in such components selection.

Ernst *et al.* [22] describe emerging research related to component selection using high-level quality attribute indicators, project health measures, and a context-specific

aggregation function for producing a single yes/no decision for integrators. The authors’ proposal involves several aspects to evaluate components according to the context. Nevertheless, the authors briefly discuss the importance of design decisions in selection. In addition, although they include stakeholders in their proposal, they do not discuss explicit mechanisms to include them in the component selection.

Cervantes *et al.* [23] investigate the criteria used by practicing software architects in selecting security frameworks. They also propose how information associated with some of the important criteria to architects can be obtained manually or in an automated way from online sources, such as GitHub. More precisely, the authors performed a study of which criteria are useful to architects in selecting application frameworks. The study also allows understanding which criteria are critical to practitioners and how data associated with some of these criteria can be gathered from online sources. The technique described by the authors does not explicitly discuss how to incorporate stakeholders’ viewpoints in selecting frameworks and technology.

B. STAKEHOLDERS AND DECISION MAKING

Aurum and Wohlin [2] introduce the decision making roles that stakeholder should play in Requirements Engineering, and argued that they must be involved in strategic decisions. Although the authors mention the importance and relevance of stakeholders in the whole process of Requirement Engineering with respect to decision making, the authors do not address the importance of stakeholders in the design and implementation of QARs. On the other hand, the authors mention that a key factor for software development success is to track the decisions made in the Requirement Engineering process. Still, they do not discuss how these decisions influence the design and implementation of software.

Petersen *et al.* [3] report a survey that classified stakeholders as 1) decision initiators, 2) decision “preparators”, or 3) decision makers. The authors also mention that often, decisions on software implementation are made by technical experts. More precisely, the authors mention that decisions made by expert stakeholders (such as software managers and software developers) are more suitable for evaluating CSOs (Component Sourcing Option). Although this study does analyze the importance of stakeholders in component evaluation, the authors do not use collaborative techniques to evaluate CSOs.

Badampudi *et al.* [24] discuss the decisions of software components selection from several origins, such as in-house development vs COTS (Components off-the-shelf), OSS (Open Source Software), and outsourcing; and proposed a decision-making process-line to select software asset origins. The authors emphasize the importance of stakeholders in their proposal, but they do not use collaborative techniques to select and evaluate COTS.

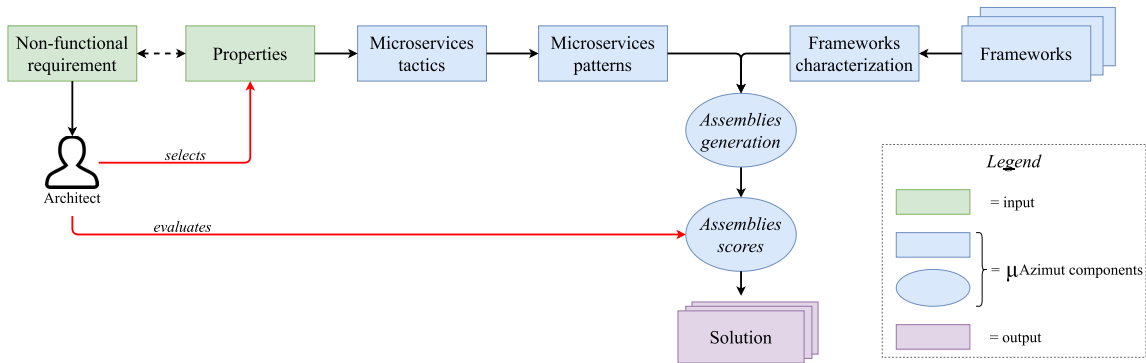


FIGURE 1. The μ Azimut approach.

C. SUMMARY

Several studies have investigated how to select software components, frameworks and platforms to design software architectures. Similarly, other studies have investigated the role and importance of stakeholders in several phases of software development. Despite this interest, to the best of our knowledge, few studies have discussed the role of stakeholders in decision-making on systems implementation. On the other hand, except for [21], few studies have discussed how to include stakeholders' expectations and viewpoints in the implementation of microservice-based systems. Therefore, our research attempts to complement the existing body of knowledge about the stakeholders and the implementation of microservice-based systems by introducing a collaborative technique that involves stakeholders in selecting properties (emerged from the description of quality attributes) to evaluate implementation solutions created from frameworks assemblies generated by μ Azimut.

IV. μ Azimut AT A GLANCE

μ Azimut [13] (see Figure 1) is a technique that uses architectural knowledge (represented in multi-dimensional catalogs of *properties*, *architectural tactics*, *architectural patterns*, and *frameworks* characterizations) to support architects in the analysis, evaluation, and comparison of *framework assemblies* to satisfy QARs in microservice-based systems. We define a framework assembly as a finite set of frameworks and platforms that satisfy one or more QARs. μ Azimut deals with information imperfection using imprecise "characterization" of architectural tactics, architectural patterns, frameworks, and platforms. In the following sections, we describe each component of μ Azimut.

A. INPUT

μ Azimut uses *properties* which characterize quality attributes of microservice architectures (See Figure 2). The primary purpose of properties is to encapsulate specific characteristics that represent quality attributes in microservice-based systems. Furthermore, properties allow the architect to focus on particular perspectives of an

Availability

- ☐ High detection of failed host
- ☐ Intermittently asynchronous data transmission
- ☐ Regular snapshots
- ☐ Efficient duration of timeouts periods
- ☐ High isolation
- ☐ Effective load balancing

Scalability

- ☐ Effective technical duplication
- ☐ High functional decomposition
- ☐ Effective data partitioning

Interoperability

- ☐ High cooperation among components
- ☐ High coordinated orchestration among components

Security

- ☐ High level of individuals, groups, or systems authorization
- ☐ High-security authentication
- ☐ Effective credentials management
- ☐ Effective access control

Alpha ☐

Beta ☐

Calculate framework assemblies

FIGURE 2. μ Azimut input. Each property is represented as an option. "Alpha" and "Beta" are parameters which establish the level of credibility required by each QAR for the tactics/patterns (α) and patterns/frameworks (β) relationship.

QAR rather than analyzing it as a whole (we describe the properties in Appendix I). For this first version of μ Azimut, we have defined properties for four quality attributes, which are availability, scalability, interoperability and security. However, we plan to include more quality attributes in future versions of the technique.

B. PROCESSING

The key μ Azimut elements are as follows:

- *Architectural tactics* for microservices ("microservices tactics" in Figure 1) that address the properties that architects select for each QAR. Architectural tactics are design decisions that influence the achievement of a quality attribute response [1].
- *Architectural patterns* for microservices ("microservice patterns" in Figure 1) [25], [26] related to microservice tactics. Architectural patterns represent systematic solutions to recurring architectural problems [1]. A given architectural pattern may be related to architectural tactics for the same quality concern or architectural tactics across several quality concerns; similarly, a given

architectural tactic may be related by several architectural patterns.

- **Frameworks and platforms** that translate functionalities to satisfy quality attributes. Frameworks are reusable elements of software that provide generic functionalities focused on solving recurrent issues [27]. On the other hand, platforms¹ are frameworks that provide a complete set of functionalities to implement an application in a particular domain [1]

To obtain frameworks assemblies, μ Azimuth uses *catalogs*² that store architects' knowledge about microservices patterns, microservices tactics, frameworks, as well as rules of satisfaction among them. Catalogs are key to reusing information about improving the quality of knowledge concerning design spaces and frameworks and to support the architect in the process of exploring these design spaces.

In practical deployment contexts, the catalog preparators might not know or not be certain whether a microservice pattern supports a certain microservice tactic. Therefore, the catalogs use *credibility degrees* to represent the imprecise information between microservices tactics, microservices patterns, frameworks. The values corresponding to credibility degrees are as follow 1 (*supports*), 0.6 (*probably supports*), 0.3 (*possibly does not support*), and 0 (*does not support*).

The number of frameworks in the assemblies depends on the level of credibility considered by the architect for an QAR. Two variables, α and β represent these levels. α represents the credibility level for the microservices tactics/microservices patterns catalog and β represents the credibility level for the microservices patterns/frameworks catalog. These variables are represented by values ranging from 0 to 1. For some critical QARs, α and β will have a value of 1. However, for others (non-critical QARs), they may have values lower than 1.

C. OUTPUT

μ Azimuth uses credibility degrees to intersect the dimensions of microservices tactics/microservices patterns and microservices patterns/frameworks catalogs in order to compute a *support score* that ranks framework assemblies based on the importance that the architect determines for each QAR (see Figure 3). The support score counts dimensions in favor of the statement “the framework assembly f_w satisfies the tactic t ”.

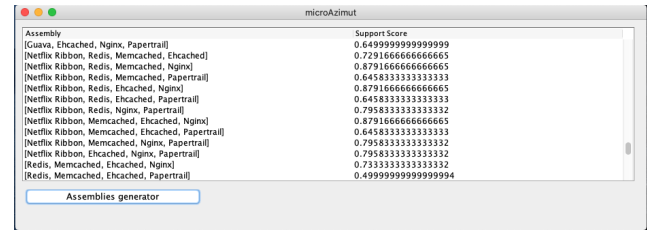
V. STAKEHOLDERS AND μ Azimuth

To incorporate stakeholders in the evaluation of implementation solutions, we modify μ Azimuth making the properties selection as collaborative, i.e., different perspectives of stakeholders are considered to select properties through consensus (see Figure 4).

Consensual selection is supported by TaSPeR [28], a consensus-building technique that allows practitioners to

¹From now, when we refer to frameworks, we also consider platforms.

²Further detail of the catalogs can be found in <https://github.com/gmarquez87/microAzimuth>



Assembly	Support Score
[Guava, Ehcache, Nginx, Papertrail]	0.6499999999999999
[Netflix Ribbon, Redis, Memcached, Ehcache]	0.7291666666666666
[Netflix Ribbon, Redis, Memcached, Nginx]	0.8791666666666666
[Netflix Ribbon, Redis, Memcached, Papertrail]	0.6458333333333333
[Netflix Ribbon, Redis, Ehcache, Nginx]	0.8791666666666666
[Netflix Ribbon, Redis, Ehcache, Papertrail]	0.6458333333333333
[Netflix Ribbon, Redis, Nginx, Papertrail]	0.7958333333333332
[Netflix Ribbon, Memcached, Ehcache, Nginx]	0.8791666666666666
[Netflix Ribbon, Memcached, Ehcache, Papertrail]	0.7958333333333332
[Netflix Ribbon, Memcached, Nginx, Papertrail]	0.7958333333333332
[Netflix Ribbon, Ehcache, Nginx, Papertrail]	0.7958333333333332
[Redis, Memcached, Ehcache, Nginx]	0.7333333333333332
[Redis, Memcached, Ehcache, Papertrail]	0.4999999999999999

FIGURE 3. μ Azimuth output.

identify, argue for, and choose among architectural design decisions according to objectives and priorities. TaSPeR extends the Planning Poker technique [29] to allow a group to agree on what design decisions should be used to develop a software architecture. The technique adapts and combines the concepts related to architectural tactics and consensus-based techniques to include stakeholders in the software design's decision-making processes. We replaced *architectural tactics* with *properties*, i.e., instead of stakeholders discussing architectural tactics, they discuss properties.

The academic literature describes several techniques and methods for prioritizing QARs. Dabbagh and Lee [30] propose an approach to prioritize quality attributes under two approaches: (1) based on the importance of customers, and (2) applying an eliminatory approach to ensure consistency in the list of prioritized quality attributes. Additionally, Dabbagh *et al.* [31] conduct an empirical study to evaluate functional and non-functional requirements (including QARs). The empirical study results show that an integrated prioritization approach technique obtains better results than the other techniques used. On the other hand, Thakurta [32] proposes a negotiation algorithm in order to facilitate the selection of quality attributes through the satisfaction of business objectives. Gupta and Gupta [33] also describe a prioritization technique based on the collaborative dependence of requirements, which takes into consideration multiple criteria to obtain individual preferences.

Considering the techniques and methods proposed to prioritize quality attributes, we followed the steps suggested by McGee *et al.* [34]. The authors consider software architecture aspects as prioritization variables for prioritizing quality attributes in order to create a properties prioritization process (see Figure 5).

A. STAKEHOLDERS IDENTIFICATION

This step aims to identify the main stakeholders in the system. Since different stakeholders profiles surround the software life cycle, this guideline recommends identifying only those stakeholders that significantly impact the software architecture decisions. The study conducted by Pacheco and Garcia [35] describe that there is a reduced set of techniques for identifying stakeholders and these are not structured. Therefore, the authors propose the following practices to identify the most relevant stakeholders in software projects:

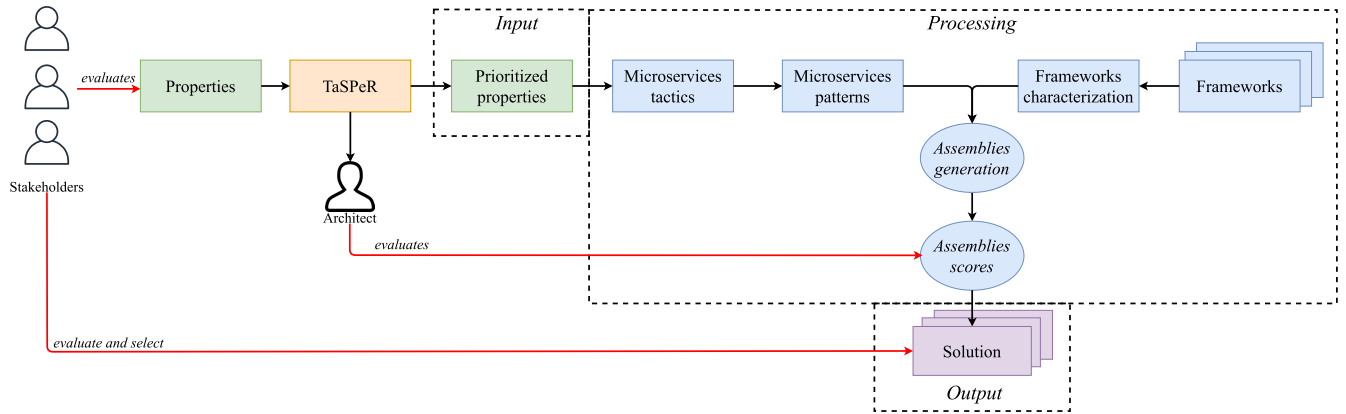


FIGURE 4. The extended μ Azimut approach.

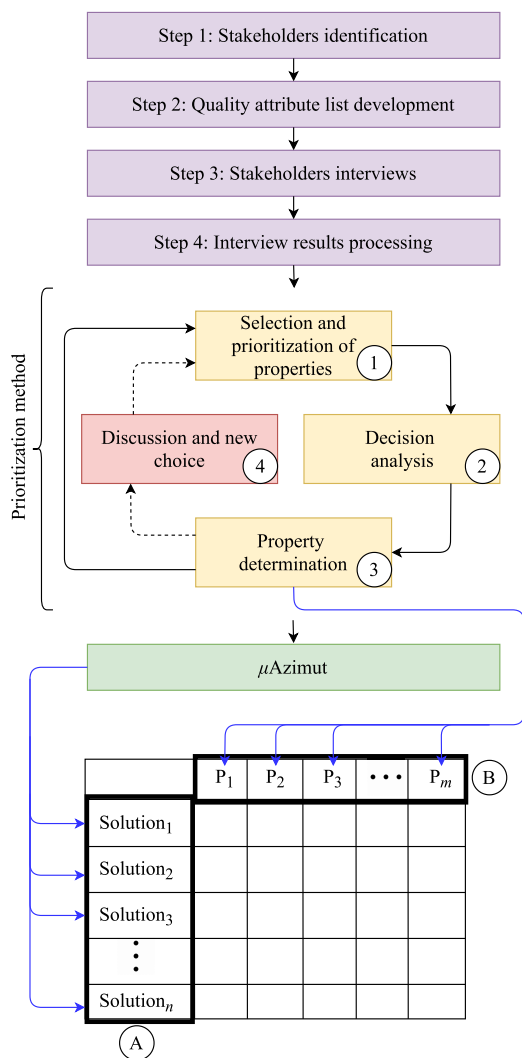


FIGURE 5. Steps to prioritize properties.

- 1) *Identify and consult all likely sources of requirements*: This practice suggests selecting stakeholders based on their experience and knowledge.

This type of stakeholders must have a broad understanding of the domain.

- 2) *Identify user classes and their characteristics*: This practice proposes evaluating stakeholders in terms of risk and cost, since the project stakeholders have different system privileges.
- 3) *Identify and consult with the stakeholders of the system*: This practice suggests identifying stakeholders through a description of the project requirements. For each requirement, the reasons why one or more stakeholders are relevant to address that requirement should be described.

For our proposal, we use all three practices to identify stakeholders.

B. QUALITY ATTRIBUTE LIST DEVELOPMENT

Once the stakeholders have been identified, it is recommended to create a list of those quality attributes that directly affect the architecture. For this, taxonomies representing software quality, such as ISO/IEC 25010,³ can be used. Additionally, we use Utility Trees to organize quality attributes. The Utility Tree is a technique that allows for prioritizing the quality attributes to evaluate the suitability of a candidate architecture against the requirements [36]. The advantage of using Utility Trees in μ Azimut points to the quick identification of the critical quality attributes for stakeholders.

C. STAKEHOLDERS INTERVIEWS

The objective of this step is to interview stakeholders in order to identify the corresponding QARs which are relevant to the project. The purpose of the interview is to obtain (i) the stakeholders' interests in the system, (ii) which activities are the most important in the organization and (iii) which roles the stakeholders will have in the system.

³<https://iso25000.com/index.php/en/iso-25000-standards/iso-25010>

D. INTERVIEW RESULTS PROCESSING

This step aims to create a quantitative method for prioritizing information and data obtained from interviews. We defined that each stakeholder must prioritize properties using the letters H (high), M (medium), and L (low). The prioritization is based on two criteria (i) business goals and (ii) the maintainability and performance of the system. The business goals define the actions to be taken to fulfill the organization mission and vision. According to Bass *et al.* [1], some business goals may lead to quality attributes (which lead to architectures), or lead directly to architectural decisions, or lead to non-architectural solutions (see Figure 6).

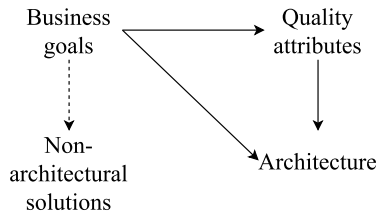


FIGURE 6. Relationship between business goals and architecture.

On the other hand, maintainability and performance belong to the set of the most relevant systemic properties in microservice-based systems [7]. Maintainability is the probability of performing a successful repair action within a given time, and performance measures how effective is a software system with respect to time constraints and resources allocation.

E. PRIORITIZATION METHOD

Once the properties are prioritized, the following steps should be executed in order to achieve stakeholders' consensus:

- *Selection and prioritization of properties* ①: In this step, the properties and the mechanisms to prioritize them are presented (see Appendix I).
- *Decision analysis* ②: Each stakeholder argues their choice and prioritization. A moderator⁴ records the rationale manifested by stakeholders.
- *Property determination* ③: If one or more properties are selected by all stakeholders, it becomes a selected property (o properties).
- *Discussion and new choice* ④ (optional): If there is no consensus on a property or a tie, stakeholders can argue their rationale and try to make a new common choice. If they still do not reach an agreement, the property is rejected.
- Repeat ①, ②, ③ and ④ until no QAR remains to be analyzed.

Finally, the architect should create a comparison matrix using the assemblies generated by μ Azimut (A) and the priorities selected by the stakeholders (B). This matrix aims to enable architects to show the satisfaction level of the priorities

⁴The moderator is the lead architect, project manager, or anyone capable of making architectural decisions on a project.

in each solution. The satisfaction level is classified using the following labels: Yes (Y), Partially (P), and No (N). Using this matrix, stakeholders can quickly decide which solution is best.

F. LIMITATIONS

The current version of μ Azimut uses two catalogs with 26 microservices tactics, 18 microservices patterns and 36 frameworks. Although the market for microservices frameworks is wide, μ Azimut only uses frameworks in whose documentation it is possible to identify both microservices patterns and microservices tactics. This constraint eventually limits the analysis capacity of architects and stakeholders to satisfy only certain QARs. With the intention of further increasing the capacity of μ Azimut, we are extending the scope of frameworks through a collaborative platform that allows gathering and describing frameworks that practitioners actually use to develop microservice-based systems.

On the other hand, μ Azimut uses four quality attributes (availability, scalability, interoperability, and security). Therefore, we are currently defining inclusion criteria to extend the range of quality attributes in the technique.

G. ILLUSTRATIVE EXAMPLE

Let us consider an architect evaluating the design and implementation of an IoT microservice architecture, with four QARs corresponding to availability, scalability, interoperability, and security. We also assume that stakeholders have already been identified.

The first procedure is the prioritization of the properties that the microservice-based system should satisfy. Therefore, the architect uses TaSPeR to involve the stakeholders in evaluating all properties and prioritizing them. Table 1 describes the selected properties.

TABLE 1. Properties for the illustrative example.

QA	ID	Property
Availability	A1	High detection of failed host
	A2	High isolation
	A3	Effective load balancing
	A4	Periodic heartbeat signal
Scalability	S1	Effective technical duplication
	S2	High functional decomposition
Interoperability	I1	High coordinated orchestration among components
Security	SC1	Strong level of individuals authorization
	SC2	Effective credentials management

Since the properties described in Table 1 are critical for stakeholders, then $\alpha = 1$ and $\beta = 1$. Once the properties are selected, μ Azimut proceeds to calculate the support scores in order to obtain a list of framework assemblies that satisfy the properties selected by the stakeholders. For simplicity of this example, Table 2 describes the first three framework assemblies generated by μ Azimut.

Once the assemblies for each QAR are generated, the architects proceed to join them and obtain a final list of solutions.

TABLE 2. Ranked framework assemblies.

Ranking	Availability	Scalability	Interoperability	Security
1	$A_{av}=[\text{Netflix Eureka, Netflix Zuul, Netflix Ribbon}]$	$A_{sc}=[\text{Kubernetes, Docker}]$	$A_{inter}=[\text{Mosquitto}]$	$A_{sec}=[\text{OAuth, Netflix Zuul}]$
2	$A_{av}=[\text{Netflix Eureka, Netflix Zuul}]$	$A_{sc}=[\text{Kubernetes}]$	$A_{inter}=[\text{Mosquitto}]$	$A_{sec}=[\text{Netflix Zuul}]$
3	$A_{av}=[\text{Netflix Eureka, Netflix Ribbon}]$	$A_{sc}=[\text{Docker}]$	$A_{inter}=[\text{RabbitMQ}]$	$A_{sec}=[\text{OAuth}]$

TABLE 3. Comparison matrix.

Solutions	A1	A2	A3	A4	S1	S2	I1	SC1	SC2
$S_1=[\text{Netflix Eureka, Netflix Zuul, Netflix Ribbon, Kubernetes, Docker, Mosquitto, OAuth, Netflix Zuul}]$	Y	Y	Y	Y	P	Y	Y	Y	P
$S_2=[\text{Netflix Eureka, Netflix Zuul, Kubernetes, Mosquitto, Netflix Zuul}]$	Y	N	Y	Y	P	Y	Y	P	P
$S_3=[\text{Netflix Eureka, Netflix Ribbon, Docker, RabbitMQ, OAuth}]$	Y	P	Y	Y	P	Y	P	Y	P

For each solution, the architect must evaluate the level of satisfaction of the properties defined by the stakeholders (see Table 3). With this in mind, the architect argues the pros and cons of each solution and thus selects, together with the stakeholders, the most appropriate solution.

VI. CASE STUDY: AMBIENT ASSISTED LIVING SYSTEM FOR MONITORING ELDERLY PATIENTS

This section reports a case study where we used μAzimut and stakeholders to evaluate implementation solutions of an Ambient Assisted Living (AAL) system, which includes an IoMT environment and a microservice-based architecture. This case study is exploratory, i.e., we will evaluate a technique (μAzimut) in a real context in order to seek new insights about stakeholders and microservice architectures. For this purpose, we used the guidelines described by Runeson and Höst [37] to conduct case studies in Software Engineering research.

A. CONTEXT

In the last few years, the population of adults over 60 years has presented a global expansion. A significant group of older adults is forced to live alone in their homes, implying an increased likelihood of suffering distress situations related to home accidents. In this regard, falls are especially relevant to patients and health systems because approximately one-third of adults older than 65 that live in a community suffer a fall each year [38].

To improve patients' quality of life, sensor-based technologies are an alternative to address elderly patient care challenges. Modern technologies, such as smart devices, allow collecting health-related sensor data (such as physiological, actimetric and others) to investigate the permanent waking states of elderly patients in order to generate conscious support ecosystems for patients and analyze the variables captured by devices to alert the fall of patients [39].

In this case study, we addressed the design and implementation of a system that allows the use of several technologies to study and estimate falls of elderly patients in their houses. The project stakeholders requested the design of an AAL system using an IoMT environment to visualize a technological solution for monitoring elderly patients with several

types of devices (not only sensors). Additionally, to facilitate the integration of devices and other systems, stakeholders required a microservice-based system.

Although AAL systems must satisfy several quality attributes (such as usability, privacy, reliability, performance, among others) and, at the same time, each of these quality attributes has different levels of complexity [40], in this case study we focused on addressing QARs related to quality attributes of μAzimut (availability, scalability, interoperability and security).

B. OBJECTIVES AND RESEARCH QUESTIONS

This case study aims to evaluate the solutions generated by μAzimut based on the properties selected by the project's stakeholders. Consequently, the main research question of the case study is:

Can μAzimut help stakeholders to evaluate and select frameworks to implement microservice-based systems?

This research question aims to evaluate the support of μAzimut for stakeholders in evaluating solutions to implement the AAL system.

C. DATA COLLECTION

Although μAzimut has an initial core of multidimensional catalogs created from previous experiences, for this case study, we investigated other studies (such as [41]–[44]) to gather frameworks that have been used in AAL and IoMT/IoT-based systems. Using inclusion and exclusion criteria defined by μAzimut to include new frameworks, we updated the catalogs and calibrated the corresponding credibility degrees.

D. PREPARATION

Before evaluating implementation solutions, we created the system design using the Attribute-Driven (ADD) method [45]. ADD is a systematic step-by-step method for designing the software architecture of a software-intensive systems.⁵

The ADD inputs are as follows: functional requirements (the functionalities that the system must provide); design

⁵A complete practical example of ADD can be consulted in [46]

TABLE 4. Steps of the ADD method and a brief description of the actions that we executed in each step.

Step	Description	Executed actions
1)	Get information about the context	We conducted an As-Is/To-Be analysis to obtain an initial diagnosis of the current and desired scenarios of the AAL system and the main design constraints.
2)	Select system elements that will be addressed	In brainstorming sessions, we identified potential microservices that could provide the architectural and monitoring capabilities to elderly patients demanded by stakeholders and patients.
3)	Rank stakeholders' requirements by impact on the architecture	We identified the stakeholders' main concerns regarding the capabilities of the architecture and the capabilities of the network.
4)	Select the most important elements that will appear in the architecture	We defined and described design decisions related to IoMT-based and AAL-based systems. Subsequently, we analyzed each design decision and defined variables to measure them. Finally, we identified the architectural drivers that the system should have.
5)	Define software elements responsibilities	In this step, we analyze each identified microservice and define each microservice's responsibilities in the architecture.
6)	Define the services and properties required by the design software elements	We represented the relation among microservices architecture components using the Richardson's notation [47] (component name, description, and dependencies). Dependencies can either "invoke" (operations implemented by other services) or "subscribe" (to messages, including events, produced by other services).
7)	Verify that the elements decomposition satisfies the functional requirements	For each microservice in the architecture, we verified if these satisfy stakeholders concerns, QARs, and design constraints.

constraints (the decisions that must be incorporated into the final version of the system design); and QARs. The output of ADD is a system design in terms of roles, responsibilities, properties, and relationships among software elements.

Table 4 describes the steps of ADD and the actions we execute in each step. Additionally, Table 5 details the rationale of the quality attributes addressed in the system and Figure 7 illustrates the high-level design of the AAL system. The AAL system has five microservices that manage the main activities required for monitoring patients in homes (details of each microservice can be found in Appendix II). Additionally, the system considers a middleware layer that connects medical devices. The microservices also share data between them. For example, the microservice *Historical events* invokes information of the microservice *Devices management* and subscribes data to the microservice *Patient management*.

On the other hand, the Appendix III describes the AAL system's microservices main architectural drivers. This information is relevant for stakeholders to prioritize properties.

E. EVALUATING IMPLEMENTATION SOLUTIONS

Before executing the steps to prioritize properties, step 3) of the ADD method (see Table 4) enabled us to select the stakeholders that will participate in the system's design. In this step of ADD, we take advantage of executing the first 4 steps of the μ Azimet extension (see Figure 5). As a result, from 8 project stakeholders, we identified 3 stakeholders who will prioritize properties. Stakeholder interests in the system are described as follows:

- **Stakeholder 1:** Main project leader and lead researcher. His interest aims to expand the system's capacity to other medical devices in order to obtain more data to train fall prevention algorithms. On the other hand, he is interested in increasing the system's capacity to expand the number of houses that will use it.
- **Stakeholder 2:** Lead researcher. He is in charge of managing medical devices and data processing. This stakeholder is interested in the integrity and availability of the project data.

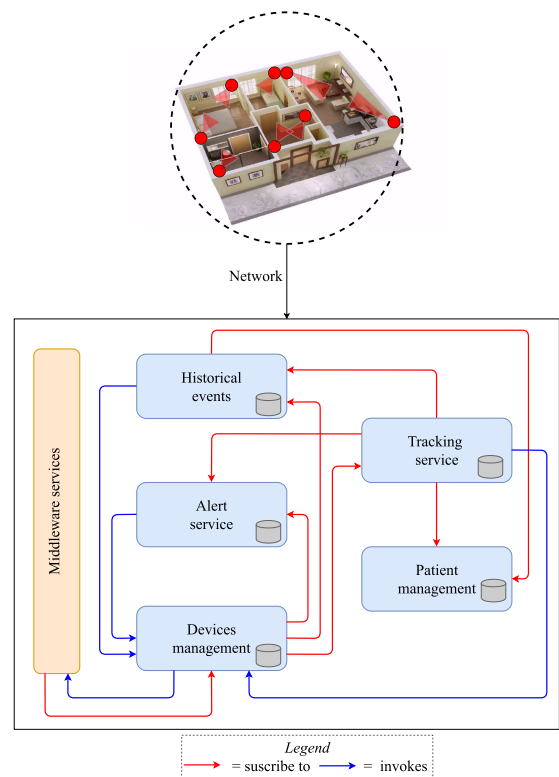


FIGURE 7. AAL system design and their corresponding microservices. Red and blue lines illustrate "subscribe to" and "invoke" messages, respectively.

- **Stakeholder 3:** Ph.D. student who will use the system outputs to conduct quantitative studies of elderly patient falls.

Once the stakeholders have been identified, we proceed to execute the prioritization process. In the following points, we describe the main activities conducted in each step. Additionally, Table 6 summarizes the selected properties.

- **Selection and prioritization of properties:** In this step, we schedule a meeting where we meet the 3 stakeholders and a architect from our research team. The aforementioned architect acts as a moderator. Then, we present

TABLE 5. Summary of the quality attributes description.

ID	QA	Rationale
QA1	Availability	One of the essential features of the system is services must have high availability. Specifically, services related to sensor data must have a low unavailable time rate. This is required because the data sent by sensors are highly sensitive and critical. Therefore, any time interval in which the data is not captured can compromise the patient's care and life.
QA2	Scalability	As the number of sensors and/or devices increases, the system must continue providing functionality within pre-established parameters. The fact that the system is scalable means that it must handle the new demand of data every time the number of devices in the house or houses is increased.
QA3	Interoperability	Another concern is the ability to interoperate with other systems to exchange information. This attribute will be addressed in a middleware layer, which allows exchanging information between the different devices that enable taking care of the patient.
QA4	Security	The data controlled by the system is sensitive. The system must satisfy confidentiality and privacy policies. Since the system will use different devices and they will be connected to a network, it is imminent that cyber-security defense mechanisms must be applied.

TABLE 6. Key properties selected by stakeholders.

Id	Property	QA	Summary of stakeholders' rationale
P1	Quick broken state recovery	QA1	Since the system depends mainly on the operation of the devices that capture the patient's information, services related to obtaining data of the devices must have a fast recovery in case of failures.
P2	High control of failure propagation	QA1	The system manages the care of elderly patients. Any scenario where services fail compromises the patient's life. Therefore, this property is critical to creating contingency plans to recover from failures in critical services.
P3	High service monitoring visibility	QA1	This property is essential for stakeholders because they want to know the service's status at all times. Furthermore, since the system will use devices and sensors in more homes, stakeholders need to control each service's status.
P4	Periodic heartbeat signal	QA1	This property supports the previous property (P3). Heartbeat allows checking the status of each service periodically.
P5	Effective technical duplication	QA2	Stakeholders are clear in requesting that the system should be scalable in the sense that if the devices increase, the capacity must be maintained.
P6	High level of individuals, groups, or systems authorization	QA4	It is necessary to control users to grant them limited access to the system resources without exposing their credentials.
P7	High cooperation among components	QA3	The system demands the capability of exchange information among services and devices (such as sensors) to use data that has been exchanged for research and patient monitoring purposes.
P8	High coordinated orchestration among components	QA3	This property points to a control mechanism to coordinate, manage and sequence the invocation of particular components.

and explain to the stakeholders the properties and the mechanism to prioritize them.

- **Decision analysis:** In a one-hour session, each stakeholder argues his or her choice and decision on priorities. In this step, stakeholders were also allowed to use any media (blogs, forums, specialized websites, among others) to argue their decision. Additionally, the moderator captures the most important decisions discussed among stakeholders.
- **Property determination:** From 12 selected properties, 8 were prioritized with the high assignment (H) in the two prioritization criteria by all stakeholders; therefore, they are automatically considered as selected properties. On the other hand, 4 properties were prioritized with different prioritization levels by two stakeholders; one stakeholder did not consider these properties in their analysis.
- **Discussion and new choice:** Since there was controversy on 4 properties, the stakeholders who selected these properties argued their choice and the rationale for prioritization in a one-hour session. At the end of the session, stakeholders chose not to consider these 4 properties. Despite the fact that these properties addressed some important business objectives, their implementation and maintenance were considered very time-consuming. Therefore, they considered leaving these properties for the next phase of the project.

With the properties already selected, we proceed to execute μ Azmut. The project architect (the moderator) then

TABLE 7. Assemblies generated by μ Azmut and their corresponding frameworks.

Assemblies	Solution A	Solution B	Solution C
Availability	Netflix Eureka	Netflix Eureka	Netflix Eureka
	Netflix Zuul	Netflix Zuul	Netflix Hystrix
	Netflix Hystrix	Apache Kafka	Apache Kafka
	Apache Kafka	Zipkin	Zipkin
	Zipkin	MongoDB	Redis
Scalability	MongoDB		
	Netflix Eureka	Netflix Eureka	Netflix Eureka
	Netflix Zuul	Netflix Zuul	Netflix Zuul
	Docker	Docker	Docker
	Kubernetes	Kubernetes	Kubernetes
	MongoDB	MongoDB	Redis
Inter.	Mosquitto	RabbitMQ	RabbitMQ
	Apache Kafka	Apache Kafka	Apache Kafka
	Apache Zookeeper	Apache Zookeeper	RabbitMQ
Sec.	Mosquitto	RabbitMQ	
	OAuth2.0	OAuth2.0	OAuth2.0
		Netflix Eureka	Netflix Eureka

takes the μ Azmut assemblies (the architect considered α and β as 1), define solutions using the assemblies, and makes high-level diagrams representing implementation solutions. Table 7 describes the solutions defined by the architect and their corresponding frameworks. It is important to note that some frameworks (e.g., Netflix Eureka) satisfy one or more quality attributes.

Subsequently, the next day, we gathered the stakeholders to evaluate the solutions. For each solution, stakeholders use the selected priorities as criteria to evaluate each solution in the comparison matrix (see Table 8).

The architect described each solution and its frameworks, focusing on its advantages and disadvantages. For example,

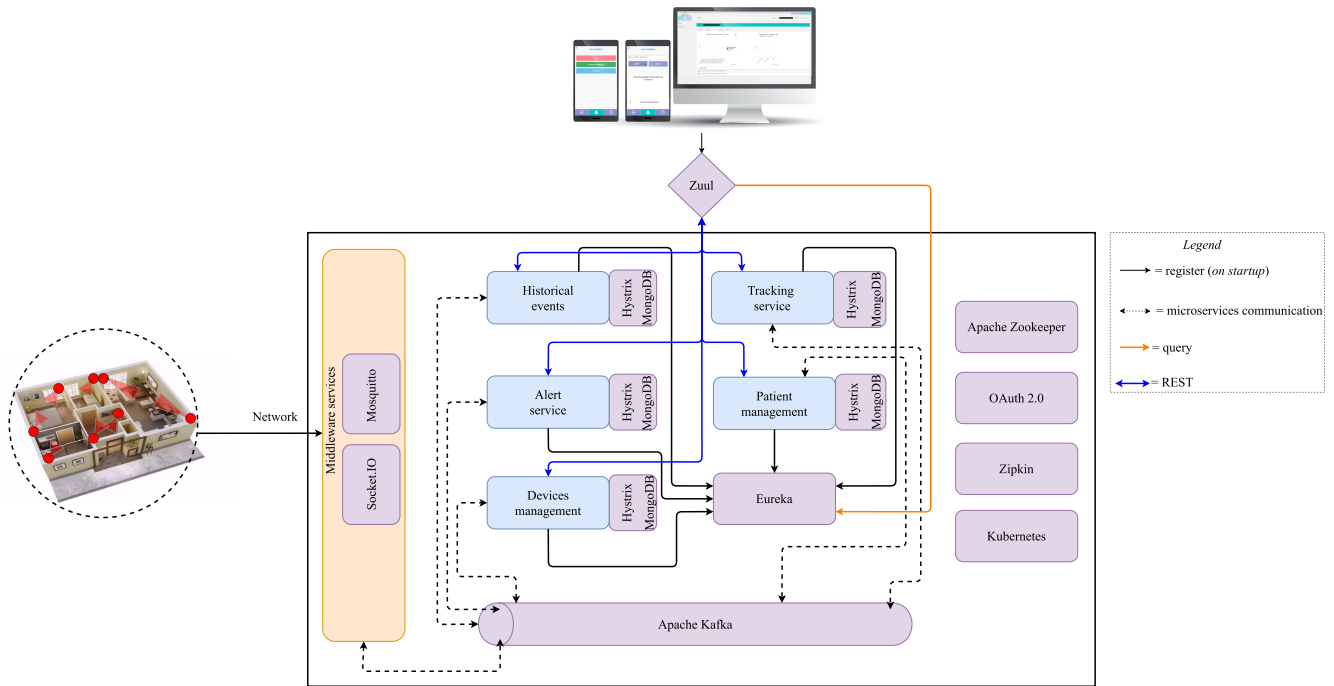


FIGURE 8. The AAL system architecture. The figure depicts the microservices architecture with its corresponding frameworks and platforms selected by the stakeholders.

TABLE 8. Comparison matrix of the solutions created by the architect.

	P1	P2	P3	P4	P5	P6	P7	P8
Solution A	P	Y	Y	P	Y	P	Y	Y
Solution B	Y	Y	P	P	P	Y	Y	P
Solution C	Y	Y	Y	P	P	Y	P	P

let us consider the following assembly [Apache Zookeeper, RabbitMQ]. The (dis)advantages of this assembly are as follows:

- Apache Zookeeper has good session timeout support, but deploying it has high resource consumption (hardware/software).
- RabbitMQ simplifies implementing asynchronous communication among microservices, but if the connection is lost it can affect the number of long-lived connections.

Comparing advantages and disadvantages of each framework allowed stakeholders to contrasting the properties with their concerns. The stakeholders' decisions for each solution are as follows:

- **Solution A:** There was unanimous agreement. Although the solution partially satisfies the properties P1, P4 and P6 (MongoDB compromises health check status functionalities and specific security mechanisms by using Netflix Eureka and Netflix Hystrix at the same time⁶), stakeholders were satisfied with the architect's proposal. Moreover, this option was the preferred one because the architect argued that these frameworks allow easy and flexible maintenance of the system. On the other hand, stakeholders preferred Netflix Zuul as API Gateway and Netflix Eureka to monitor the services. Additionally,

stakeholders appreciated that the first draft had an interoperability layer to manage both sensors data.

- **Solution B:** Stakeholders were not in agreement about the acceptance of the proposed implementation. The combination of frameworks seemed to satisfy most concerns, especially regarding availability. The draft was rejected because they investigated RabbitMQ thoroughly and realized that some issues reported by developers described that this framework has bugs regarding the integration with particular medical devices. Furthermore, given the importance of patient data, stakeholders were not satisfied that solution B partially addresses P3 because this property is very critical to them.
- **Solution C:** This solution was a candidate as a final solution. But there was a debate about sacrificing the P7 property. Although this solution is much more flexible than Solution A, it partially addresses cooperation between medical devices. On the other hand, stakeholders accepted Netflix Eureka and Netflix Hystrix as infrastructure services, but didn't accept Redis⁷ as a database. Despite its potential advantages, stakeholders decided that this database had too many issues raised in other open microservice projects, and rejected it.

F. RESULTS

Table 9 and Figure 8 summarize the frameworks selected (solution A) and the AAL system overview, respectively. The main features of the system are as follows:

⁷Redis: in-memory database engine (based on hash tables) that can also be used as a persistent database.

⁶<https://github.com/spring-cloud/spring-cloud-netflix/issues/1780>

TABLE 9. Frameworks and platforms selected for the case study.

Name	URL	Description	QA	Property
Netflix Eureka	https://github.com/Netflix/eureka	Service registry for resilient mid-tier load balancing and failover.	QA1, QA2	P2, P5
Netflix Zuul	https://github.com/Netflix/zuul	Gateway service that provides dynamic routing, monitoring, resiliency, security, and other features.	QA1, QA2	P3, P5
Netflix Hystrix	https://github.com/Netflix/Hystrix	Latency and fault tolerance library designed to isolate points of access to distributed systems.	QA1	P2
Apache Kafka	https://kafka.apache.org	Open-source stream-processing software platform.	QA1, QA3	P4, P7
Docker	https://www.docker.com	Automates the deployment of applications within software containers.	QA2	P5
Apache Zookeeper	https://zookeeper.apache.org	Server which enables highly reliable distributed coordination.	QA3	P8
Socket.IO	https://socket.io	Bi-directional communication library between web clients and servers.	QA2	P5
OAuth 2.0	https://oauth.net/2/	The industry-standard protocol for authorization.	QA4	P6
Zipkin	https://zipkin.io	Distributed tracing system. It helps gather timing data needed to troubleshoot latency problems.	QA1	P4
Kubernetes	https://kubernetes.io	System for automating deployment, scaling, and management of containerized application.	QA2	P5
MongoDB	https://www.mongodb.com	Database that provides high performance, high availability, and automatic scaling.	QA1, QA2	P4, P5
Mosquitto	https://mosquitto.org	Message broker that implements the MQTT protocol versions 5.0, 3.1.1 and 3.1	QA2, QA3	P5, P7

- The system uses Netflix Zuul to route user requests.
- Netflix Zuul runs REST calls in all microservices.
- Each microservice has MongoDB as database manager and Netflix Hystrix as the access point manager for each microservice.
- Apache Kafka acts as a streaming processor between the middleware layer and the microservices.
- Netflix Eureka manages the microservices registry and communicates with Netflix Zuul to manage microservices calls.
- Apache Zookeeper, OAuth 2.0, Zipkin and Kubernetes services provide other properties desirable by stakeholders, such as security, monitoring and scalability.
- The middleware layer that manages in-home medical devices is supported by Socket.IO and Mosquitto.

To illustrate how the properties selected by the stakeholders are addressed by solution A, let us consider one of the most critical functionalities of the AAL system, which is *real-time monitoring elderly patients in their homes* (see Figure 9). This functionality is related to properties P3, P4, P7 and P8.

To provide real-time patient monitoring, the data incoming from sensors processed by *Device management* and *Historical events* must be kept highly available. Mosquitto and Socket.IO capture data from 1D and 2D sensors and send them to the system via Apache Kafka. This system uses heartbeats to monitor the microservices' status and also connects them through data pipelines. Concerning the 2D sensor, data capture is trivial. Nevertheless, regarding the 1D sensor, the data capture requires 4 Omron D6T-8L-06 sensors per room to enable sensing most of the patient's bodies. Then, the captured data is stored in structured files, more precisely, in 32×24 matrices (2D sensor) and 1×33 arrays (4 1D sensors measurement + timestamp).

Since patient monitoring must be done in runtime, the status of all microservices must be monitored. Netflix Eureka checks that both *Device management* and *Historical events* be active and records the data (see Figure 10). On the other

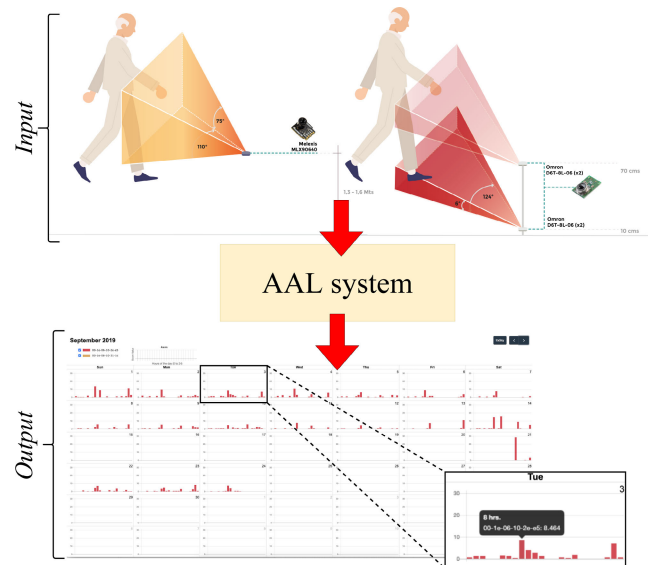


FIGURE 9. Real-time patient monitoring. The input for this functionality is the data captured by the sensors. The output is a real-time graph showing the peaks of patient movements in time intervals. The bars indicate patient movement. The absence of these may indicate (i) the patient is resting, (ii) the patient is sleeping or (iii) the patient has fallen.

hand, Netflix Hystrix can monitor the microservices' status and the number of incoming requests in real-time.

Finally, *Historical events* computes *presence scores* [48] on the data to identify behavior patterns of patients in the room.

G. DISCUSSION

The results obtained in the case study suggest that μ Azimet helps to reduce the difficulty for stakeholders to evaluate frameworks. Indeed, stakeholders agreed that the technique not only helps to reduce the space of solutions that must be analyzed to satisfy QARs but also allows people, who are not familiar with technologies, frameworks or software

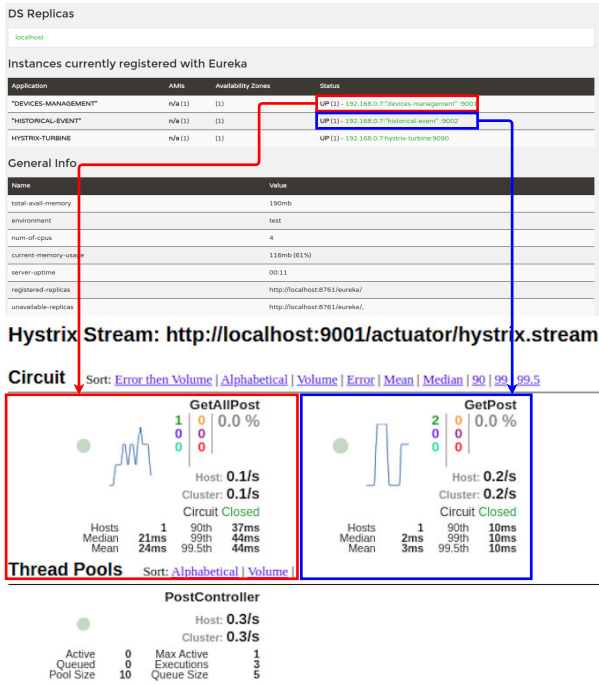


FIGURE 10. Microservices status monitoring. The red line corresponds to the device management microservice and the blue one to the historical events microservice.

architectures, to infer and visualize potential advantages and disadvantages of a certain architecture. In critical systems, like the AAL system under study, the success or failure of QARs satisfaction can compromise not only the system itself but also people’s health. In fact, some studies, such as [49], state that considering the perspectives of stakeholders in AAL systems, (e.g., Telemonitoring systems) can address frequent problems in emergency services. In this case study, for example, stakeholders were emphatic in highlighting that availability is a critical attribute for them because if sensors data are not available, it can compromise a patient’s life. For this reason, the effort to include stakeholders in the evaluation of frameworks to implement the AAL system turned out to be quite beneficial because (i) we were able to make, explain and show design and implementation decisions and (ii) we were able to identify early risks that may arise in the development and deployment of the system. We hence were able to early manage stakeholder expectations by clarifying what the AAL system can and cannot do.

Stakeholders also highlighted that μ Azimet is quite useful because, instead of selecting frameworks by “trial and error”, μ Azimet generates frameworks assemblies supported by architectural constructs (microservices patterns and microservices tactics). Consequently, we realized that using architectural patterns and architectural tactics allowed us to have a precise response to any stakeholder inquiry regarding why we use a specific framework rather than another.

Another interesting aspect to discuss is the prioritization of properties. What difference would it have made if an architect

with expertise in microservices and AAL systems had prioritized properties? A probable answer to that question would be “none”. Nevertheless, the significant difference we detected after analyzing the results and interviewing the stakeholders points to understanding the stakeholders’ views. A collaborative selection among stakeholders helps the architect to understand better how different stakeholders *observe* the system. Therefore, considering stakeholder opinions allows the architect to make architectural decisions by judging and balancing different views of the system.

H. LESSONS LEARNED

In this case study, we could identify that, in general, stakeholders had many expectations regarding the use of a microservices architecture for the AAL system. These expectations arise from experiences of other researchers in the use of microservices architecture in different projects related to IoMT. However, stakeholders realized that technology selection seriously compromises the capabilities of microservice-based systems. The use of the properties to evaluate the solutions was critical for the stakeholders because it made it easier for them to evaluate solutions without necessarily being experts in microservice development technology. Therefore, we realized that if stakeholders have an instrument to facilitate their analysis and understanding of microservices architectures, it is possible to increase their participation in the decision-making process of implementing microservice-based systems.

Schulenklopper and Rommes [5] describe three suggestions for bridging the communication gap between stakeholders and IT professionals. The first suggestion is to ignore the differences between stakeholders and IT professionals and use technical models and languages to explain a technological solution. The second suggestion is to become a polyglot and learn the language of the stakeholders. The third suggestion is to create a common language between stakeholders and IT professionals to communicate technical solutions. Our case study experience shows that creating a common language between stakeholders and architects could help evaluate microservice-based systems technology solutions (the third suggestion). This *common language* corresponds to the μ Azimet properties. From a clinical point of view, creating this common language also allowed us to understand the catastrophic consequences of making bad architectural decisions in systems that manage elderly patients’ data.

On the other hand, some challenges emerge when considering stakeholders in the implementation of microservices architectures. The main challenge is related to managing expectations regarding the qualities that microservices architecture should satisfy. Due to each stakeholder has a particular perspective and interest in the system, the analysis and trade-offs of the qualities may not easily lead to agreements.

Another challenge that we detected in the case study is time management when there are no agreements between stakeholders. In the discussion of certain qualities attributes, the time used to reach an agreement is quite long. The reasons

for the increased discussion time can be varied. However, we realized that the analysis of architecture trade-offs takes quite some time. Each stakeholder has a particular interest in the system. In those moments, when trade-offs had to be discussed, some stakeholders did not want to compromise their interests. Consequently, the moderator's role is important in monitoring these cases and making a final determination if there is no agreement among stakeholders.

VII. THREATS TO VALIDITY

In this section, we proceed to describe the threats to the validity of our research. We used the validity threats described by Wohlin *et al.* [50].

A. INTERNAL VALIDITY

The threats to internal validity refer to the factors that could negatively affect the analysis process and data collection. The main threats detected and their mitigation are as follows:

- *Stakeholder analysis capacity*: This threat emerges when stakeholders react differently as time passes. To mitigate this threat, we analyzed the implementation solutions proposals in a single session, with a 10-minute break in between.
- *Instrumentation*: This threat is related to the impact of using incorrect artifacts in the case study. This threat was mitigated through the research of other AAL systems. The frameworks used to develop the systems were analyzed and then they were incorporated into μ Azimut. Additionally, the μ Azimut catalogs were updated based on the analysis of AAL systems.

B. EXTERNAL VALIDITY

The threats to external validity threats limit the ability to generalize the study results to other setting. The main threat we detect is *selecting subjects that are not representative* of the population we want to generalize to, i.e., wrong people participate in the case study. To mitigate this threat, we defined four steps to identify stakeholders. Furthermore, we followed the practices suggested by McGee *et al.* [34] and Pacheco and Garcia [35] to characterize and evaluate stakeholders. These guidelines allow us to identify suitable stakeholders to execute the case study.

C. CONSTRUCT VALIDITY

The threats to construct validity are related to the study's generalizability to the theory behind it. We identified as the main threat the *inadequate explanation of constructs*, i.e., the little description of μ Azimut and the case study's objectives. We explained to the stakeholders the main concepts related to microservice architectures, AAL and IoMT systems to mitigate this threat. Likewise, we also showed them the functionalities of some of the frameworks that satisfy the project's needs. Additionally, we explained all the properties of μ Azimut and showed them examples of these.

D. CONCLUSION VALIDITY

Conclusion validity is concerned about issues that may affect the ability to draw the correct conclusion about the relation between the case study and the results. We identified the *quality attributes addressed by μ Azimut* as a threat. The version of μ Azimut we used for the case study considers four quality attributes, which are availability, scalability, interoperability and security. Although this number of quality attributes is limited, they belong to the group of quality attributes critical for AAL systems. Therefore, we know that we cannot generalize the results obtained in the case study because we show the solution of a part of the AAL system. Still, the promising results we obtained with these quality attributes allow us to infer that we will have more successful results in the future. Furthermore, we are currently increasing the number of quality attributes in μ Azimut in order to obtain more significant and replicable results.

VIII. CONCLUSION

In this article, we have present a collaborative technique to include stakeholders in the implementation of microservice-based systems. Our technique considers the extension of the μ Azimut technique. This technique focuses on support architects in the analysis, evaluation, and comparison of framework assemblies to satisfy QARs in microservice-based systems. Our proposal points to extend the input of the μ Azimut technique in order to allow stakeholders' consensus decisions. Additionally, we defined a set of steps to (i) identify and evaluate suitable stakeholders to use our technique and (ii) prioritize properties collaboratively.

We evaluate our proposal in a case study, where stakeholders participated in selecting frameworks and platforms to implement an Ambient Assisted Living (AAL) system composed of microservice architecture and an IoMT environment. The results suggest that the stakeholders felt comfortable evaluating implementation solutions because the properties they selected were used as evaluation criteria for the solutions proposed by the project architect. Our proposal was also useful to manage the stakeholders' expectations regarding the implementation of the AAL system, i.e., the technique allowed the architect to manifest the functionalities and restrictions that the AAL system will have in the early stages of the project development.

To further our research, we plan to expand the capacity of μ Azimut, including more quality attributes. On the other hand, we are developing a platform to identify frameworks used by microservices developers in order to enrich the μ Azimut catalogs. We also want to include properties and technological tools related to DevOps and microservices architectures [51] in μ Azimut. Finally, we will evaluate including Multiple-Criteria Decision-Making methods (MCDM) (such as Analytic Hierarchy Process (AHP) [52]) in our technique.

TABLE 10. High-availability properties.

Id	Name	Description
P1	<i>High detection of failed host</i>	This property defines the capability of detect failed hosts in order to a load balancer can stop requests to them.
P2	<i>Intermittently asynchronous data transmission</i>	Communication property where a message sender does not wait for a response.
P3	<i>Regular snapshots</i>	Property related to recovering the system from planned host maintenance owing to hardware upgrade, soft reboot, among others.
P4	<i>Efficient duration of timeouts periods</i>	Property that prevents remote procedure calls from waiting indefinitely for a response.
P5	<i>High isolation</i>	The property where each microservices is its own encapsulated application.
P6	<i>Effective load balancing</i>	Efficiently distributing incoming network traffic among groups of backend servers.
P7	<i>Quick broken state recovery</i>	Capability to restart states when they are broken for a more extended period.
P8	<i>High control of failure propagation</i>	Property which indicates the capability of isolate failures through a good definition of service boundaries.
P9	<i>High service monitoring visibility</i>	Property that allows visibility into the health of the microservice architecture.
P10	<i>Periodic heartbeat signal</i>	Property related to the periodic signal to check the status of services.
P11	<i>Low application restarting</i>	This property refers to the low rate of restart services when a failure occurs.
P12	<i>Efficient resources consumption</i>	Property related to the impact on the resources consumption (hardware/software) of a system.

TABLE 11. Interoperability properties.

Id	Name	Description
I1	<i>High cooperation among components</i>	The system demands the capability of exchange information among services and devices (such as sensors) to use data that has been exchanged for research and patient monitoring purposes.
I2	<i>High coordinated orchestration among components</i>	This property points to a control mechanism to coordinate, manage and sequence the invocation of particular components (which could be ignorant of each other).

TABLE 12. Security properties.

Id	Name	Description
C1	<i>High level of individuals, groups, or systems authorization</i>	Capacity of control users to grant them limited access to platforms systems resources without having to expose their credentials.
C2	<i>High-security authentication</i>	Capacity of verifying the identity of a person or device.
C3	<i>Effective credentials management</i>	Property related to the management of credentials, making them available to less or high privileged users for authentication to other systems without giving them access to the credentials themselves.
C4	<i>Effective access control</i>	Property that allows verifying if an entity requesting access to a resource has the necessary rights to do so.

APPENDIX I. PROPERTIES

In this section, we describe the properties used by μ Azimet. Tables 10, 11, 12 and 13 describe the properties of Availability, Interoperability, Security and Scalability, respectively.

APPENDIX II. MICROSERVICES DESCRIPTION

In this appendix, we describe the system's microservices.

APPENDIX III. ARCHITECTURAL DRIVERS

In this section, we describe the architectural drivers used by stakeholders as criteria for evaluating properties.

TABLE 13. Scalability properties.

Id	Name	Description
S1	<i>Effective technical duplication</i>	This property focuses on the need to execute multiple identical copies of an application behind a load balancer, to improve its capacity and availability.
S2	<i>High functional decomposition</i>	This property focuses on separating services and data along noun or verb boundaries, allowing segmentation of teams and ownership of code and data.
S3	<i>Effective data partitioning</i>	This property focuses on grouping related items.

TABLE 14. Tracking microservice description.

Name	Tracking service
Description	The primary purpose of this service is to provide information to investigate the activities of elderly patients in the rooms.
Dependencies	
Invokes	Subscribes to
<ul style="list-style-type: none"> Devices Management 	<ul style="list-style-type: none"> Patient Management Historical events Alert service

TABLE 15. Middleware microservice description.

Name	Middleware service
Description	These services usually provide an abstraction layer for devices. Thus, they are able to ensure data transfer between such services and therefore achieve interoperability
Dependencies	
Invokes	Subscribes to
<ul style="list-style-type: none"> No services involved 	<ul style="list-style-type: none"> Devices Management

A. ARCHITECTURAL CAPABILITIES

Since the system is based on IoMT, this concept points to the collection of medical devices and applications that connect to healthcare systems through networks [53]. Therefore, the main system capabilities from the IoMT viewpoint are as follow:

- Provide real-time data to monitor the patient's health status:** This capacity is relevant since the data that the system will receive comes from patients. This means that this real-time communication must ensure the integrity of the data. Furthermore, security and privacy aspects must also be taken into account so that the patient can trust in the devices' functionalities.
- Integration with other devices:** IoMT extends systems' capacity to the integration of data from several medical devices using appropriated middleware services.

TABLE 16. Devices management microservice description.

Name	<i>Devices management</i>
Description	This service processes all the information captured by sensors in the room or house. In addition, this service is not limited to sensors; it can be extended to other devices.
Dependencies	
Invokes	Subscribes to
<ul style="list-style-type: none"> • Middleware service 	<ul style="list-style-type: none"> • Historical events • Alert service • Tracking system

TABLE 17. Historical events microservice description.

Name	<i>Historical events</i>
Description	This service intends to provide information and analysis regarding accidents or incidents in order to discover all the history of an accident so that it can be avoided. Furthermore, the idea of this service is to identify improvements aimed at preventing or mitigating possible accidents in elderly patients. This service also allows the identification of hazards in order to be evaluated and classified.
Dependencies	
Invokes	Subscribes to
<ul style="list-style-type: none"> • Devices Management 	<ul style="list-style-type: none"> • Patient Management

- *Alert family members or medical staff when there are abnormalities in the patient's daily behavior:* The system must have the ability to integrate all the necessary services to alert and inform family members or medical staff when in life-threatening circumstances. For this, procedures for checking the status of services in time batches (time to be defined) must be applied in order to monitor the current status of the services.

Interoperability, subscription, notification, and command execution should be performed in the middleware layer. This layer offers an abstraction to handle connected objects. This set of services is critical to be able to integrate different devices into the system.

The system should have an asynchronous event-driven web application server for presenting information and notifications in a client's application. It communicates by REST API.

B. NETWORK CAPABILITIES

The access protocol management module should implement an interface to handle the different communication protocols (such as IEEE 802.15. \times and 802.11).

TABLE 18. Patient management microservice description.

Name	<i>Patient management</i>
Description	This service manages the different profiles of elderly patients. It uses information from other health institutions that allow it to characterize the patient and his health conditions (for example, patients who, due to health conditions, regularly attend the bathroom
Dependencies	
Invokes	Subscribes to
<ul style="list-style-type: none"> • <i>Other sources of information</i> 	<ul style="list-style-type: none"> • Historical events • Tracking service

TABLE 19. Alert microservice description.

Name	<i>Alert service</i>
Description	This service aims to generate the corresponding alerts to warn the medical staff, family, or other interested parties. This service should eventually interoperate with other systems.
Dependencies	
Invokes	Subscribes to
<ul style="list-style-type: none"> • Devices Management 	<ul style="list-style-type: none"> • <i>Client or other alert systems</i>

The connectivity protocol module will use lightweight Publish/Subscribe messaging transport. Publish/subscribe messaging is a method of asynchronous service-to-service communication used in serverless and microservices architectures. In a Publish/Subscribe model, any message published to a topic is immediately received by all subscribers to the topic [54]. Publish/Subscribe messaging can be used to decouple applications in order to increase performance, reliability, and scalability.

REFERENCES

- [1] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 3rd ed. Glenview, IL, USA: Software Engineering Institute, 2013.
- [2] A. Aurum and C. Wohlin, "The fundamental nature of requirements engineering activities as a decision-making process," *Inf. Softw. Technol.*, vol. 45, no. 14, pp. 945–954, 2003, doi: [10.1016/S0950-5849\(03\)00096-X](https://doi.org/10.1016/S0950-5849(03)00096-X).
- [3] K. Petersen, D. Badampudi, S. M. A. Shah, K. Wnuk, T. Gorschek, E. Papatheocharous, J. Axelsson, S. Sentilles, I. Crnkovic, and A. Cicchetti, "Choosing component origins for software intensive systems: In-house, COTS, OSS or outsourcing?—A case survey," *IEEE Trans. Softw. Eng.*, vol. 44, no. 3, pp. 237–261, Mar. 2018, doi: [10.1109/TSE.2017.2677909](https://doi.org/10.1109/TSE.2017.2677909).
- [4] K. Smolander and T. Päiväranta, "Describing and communicating software architecture in practice: Observations on stakeholders and rationale," in *Proc. Int. Conf. Adv. Inf. Syst. Eng.*, 2002, pp. 117–133, doi: [10.1007/3-540-47961-9_11](https://doi.org/10.1007/3-540-47961-9_11).
- [5] J. Schulenklopper and E. Rommes, "Why they just Don't get it: Communicating about architecture with business stakeholders," *IEEE Softw.*, vol. 33, no. 3, pp. 13–19, May 2016, doi: [10.1109/MS.2016.67](https://doi.org/10.1109/MS.2016.67).

- [6] J. Lewis and M. Fowler. *Microservices. A Definition of This New Architectural Term*. Accessed: Nov. 9, 2020. [Online]. Available: <https://martinfowler.com/articles/microservices.html>
- [7] P. Di Francesco, P. Lago, and I. Malavolta, "Architecting with microservices: A systematic mapping study," *J. Syst. Softw.*, vol. 150, pp. 77–97, Apr. 2019, doi: [10.1016/j.jss.2019.01.001](https://doi.org/10.1016/j.jss.2019.01.001).
- [8] S. Haselböck, R. Weinreich, and G. Buchgeher, "Decision models for microservices: Design areas, stakeholders, use cases, and requirements," in *Proc. Eur. Conf. Softw. Archit.*, 2017, pp. 155–170, doi: [10.1007/978-3-319-65831-5_11](https://doi.org/10.1007/978-3-319-65831-5_11).
- [9] F. Rademacher, P. Sorgalla, J. Wizenty, S. Sachweh, and A. Zündorf, "Graphical and textual model-driven microservice development," in *Microservices*. Cham, Switzerland: Springer, 2020, pp. 147–179, doi: [10.1007/978-3-030-31646-4_7](https://doi.org/10.1007/978-3-030-31646-4_7).
- [10] A. Bucchiarone, N. Dragoni, S. Dustdar, S. T. Larsen, and M. Mazzara, "From monolithic to microservices: An experience report from the banking domain," *IEEE Softw.*, vol. 35, no. 3, pp. 50–55, May 2018.
- [11] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables DevOps: Migration to a cloud-native architecture," *IEEE Softw.*, vol. 33, no. 3, pp. 42–52, May 2016.
- [12] W. Luz, E. Agilar, M. C. de Oliveira, C. E. R. de Melo, G. Pinto, and R. Bonifácio, "An experience report on the adoption of microservices in three Brazilian government institutions," in *Proc. 32nd Brazilian Symp. Softw. Eng. (SBES)*, 2018, pp. 32–41, doi: [10.1145/3266237.3266262](https://doi.org/10.1145/3266237.3266262).
- [13] G. Marquez, Y. Lazo, and H. Astudillo, "Evaluating frameworks assemblies in microservices-based systems using imperfect information," in *Proc. IEEE Int. Conf. Softw. Archit. Companion (ICSA-C)*, Mar. 2020, pp. 250–257, doi: [10.1109/ICSA-C50368.2020.00049](https://doi.org/10.1109/ICSA-C50368.2020.00049).
- [14] F. Hujainah, R. B. A. Bakar, M. A. Abdulgaber, and K. Z. Zamli, "Software requirements prioritisation: A systematic literature review on significance, stakeholders, techniques and challenges," *IEEE Access*, vol. 6, pp. 71497–71523, 2018, doi: [10.1109/ACCESS.2018.2881755](https://doi.org/10.1109/ACCESS.2018.2881755).
- [15] N. Ernst, J. Klein, G. Mathew, and T. Menzies, "Using stakeholder preferences to make better architecture decisions," in *Proc. IEEE Int. Conf. Softw. Archit. Workshops (ICSAW)*, Apr. 2017, pp. 133–136, doi: [10.1109/ICSAW.2017.19](https://doi.org/10.1109/ICSAW.2017.19).
- [16] F. Hujainah, R. B. A. Bakar, and M. A. Abdulgaber, "StakeQP: A semi-automated stakeholder quantification and prioritisation technique for requirement selection in software system projects," *Decis. Support Syst.*, vol. 121, pp. 94–108, Jun. 2019, doi: [10.1016/j.dss.2019.04.009](https://doi.org/10.1016/j.dss.2019.04.009).
- [17] M. H. Yip and T. Juhola, "Stakeholder involvement in software system development—insights into the influence of product-service ratio," *Technol. Soc.*, vol. 43, pp. 105–114, Nov. 2015, doi: [10.1016/j.techsoc.2015.05.006](https://doi.org/10.1016/j.techsoc.2015.05.006).
- [18] V. A. Shekhovtsov, H. C. Mayr, and C. Kop, "Stakeholder involvement into quality definition and evaluation for service-oriented systems," in *Proc. 1st Int. Workshop User Eval. Softw. Eng. Researchers (USER)*, Jun. 2012, pp. 49–52.
- [19] M. Sultan, "Linking Stakeholders' viewpoint concerns and microservices-based architecture," 2020, *arXiv:2009.01702*. [Online]. Available: <http://arxiv.org/abs/2009.01702>
- [20] H. Astudillo, J. Pereira, and C. López, "Identifying 'interesting' component assemblies for NFRs using imperfect information," in *Software Architecture*. Berlin, Germany: Springer, 2006, pp. 204–211, doi: [10.1007/11966104_15](https://doi.org/10.1007/11966104_15).
- [21] V. Lenarduzzi and O. Sievi-Korte, "Software components selection in microservices-based systems," in *Proc. 19th Int. Conf. Agile Softw. Develop., Companion*, 2018, pp. 1–3, doi: [10.1145/3234152.3234154](https://doi.org/10.1145/3234152.3234154).
- [22] N. Ernst, R. Kazman, and P. Bianco, "Component comparison, evaluation, and selection: A continuous approach," in *Proc. IEEE Int. Conf. Softw. Archit. Companion (ICSA-C)*, Mar. 2019, pp. 87–90, doi: [10.1109/ICSA-C.2019.00023](https://doi.org/10.1109/ICSA-C.2019.00023).
- [23] H. Cervantes, R. Kazman, J. Ryoo, J. Cho, G. Cho, H. Kim, and J. Kang, "Data-driven selection of security application frameworks during architectural design," in *Proc. 52nd Hawaii Int. Conf. Syst. Sci.*, 2019, pp. 1–10.
- [24] D. Badampudi, K. Wnuk, C. Wohlin, U. Franke, D. Smite, and A. Cicchetti, "A decision-making process-line for selection of software asset origins and components," *J. Syst. Softw.*, vol. 135, pp. 88–104, Jan. 2018, doi: [10.1016/j.jss.2017.09.033](https://doi.org/10.1016/j.jss.2017.09.033).
- [25] D. Taibi, V. Lenarduzzi, and C. Pahl, "Architectural patterns for microservices: A systematic mapping study," in *Proc. 8th Int. Conf. Cloud Comput. Services Sci.*, 2018, pp. 221–232, doi: [10.5220/0006798302210232](https://doi.org/10.5220/0006798302210232).
- [26] G. Marquez, F. Osses, and H. Astudillo, "Review of architectural patterns and tactics for microservices in academic and industrial literature," *IEEE Latin Amer. Trans.*, vol. 16, no. 9, pp. 2321–2327, Sep. 2018, doi: [10.1109/TLA.2018.8789551](https://doi.org/10.1109/TLA.2018.8789551).
- [27] R. Kazman. *Rapid Software Composition by Assessing Untrusted Components*. Accessed: Nov. 1, 2020. [Online]. Available: https://insights.sei.cmu.edu/sei_blog/2018/11/rapid-software-composition-by-assessing-untrusted-components.html
- [28] F. Osses, G. Márquez, M. M. Villegas, C. Orellana, M. Visconti, and H. Astudillo, "Security tactics selection poker (TaSPeR) a card game to select security tactics to satisfy security requirements," in *Proc. 12th Eur. Conf. Softw. Archit., Companion*, 2018, p. 54, doi: [10.1145/3241403.3241459](https://doi.org/10.1145/3241403.3241459).
- [29] M. Cohn, *Agile Estimating and Planning*. London, U.K.: Pearson, 2005.
- [30] M. Dabbagh and S. P. Lee, "A consistent approach for prioritizing system quality attributes," in *Proc. 14th ACIS Int. Conf. Softw. Eng., Artif. Intell., Netw. Parallel/Distrib. Comput.*, Jul. 2013, pp. 317–322, doi: [10.1109/SNPD.2013.9](https://doi.org/10.1109/SNPD.2013.9).
- [31] M. Dabbagh, S. P. Lee, and R. M. Parizi, "Functional and non-functional requirements prioritization: Empirical evaluation of IPA, AHP-based, and HAM-based approaches," *Soft Comput.*, vol. 20, no. 11, pp. 4497–4520, Nov. 2016, doi: [10.1007/s00500-015-1760-z](https://doi.org/10.1007/s00500-015-1760-z).
- [32] R. Thakurta, "A framework for prioritization of quality requirements for inclusion in a software project," *Softw. Qual. J.*, vol. 21, no. 4, pp. 573–597, Dec. 2013, doi: [10.1007/s11219-012-9188-5](https://doi.org/10.1007/s11219-012-9188-5).
- [33] A. Gupta and C. Gupta, "Towards dependency based collaborative method for requirement prioritization," in *Proc. 11th Int. Conf. Contemp. Comput. (IC)*, Aug. 2018, pp. 1–3, doi: [10.1109/IC3.2018.8530542](https://doi.org/10.1109/IC3.2018.8530542).
- [34] R. A. McGee, U. Eklund, and M. Lundin, "Stakeholder identification and quality attribute prioritization for a global vehicle control system," in *Proc. 4th Eur. Conf. Softw. Archit. Companion (ECSA)*, 2010, pp. 43–48, doi: [10.1145/1842752.1842765](https://doi.org/10.1145/1842752.1842765).
- [35] C. Pacheco and I. Garcia, "A systematic literature review of stakeholder identification methods in requirements elicitation," *J. Syst. Softw.*, vol. 85, no. 9, pp. 2171–2181, Sep. 2012, doi: [10.1016/j.jss.2012.04.075](https://doi.org/10.1016/j.jss.2012.04.075).
- [36] R. Kazman, M. Klein, and P. Clements, "ATAM: Method for architecture evaluation," *Softw. Eng. Inst., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU/SEI-2000-TR-004, ADA382629*, 2000.
- [37] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Softw. Eng.*, vol. 40, no. 2, p. 131, 2008, doi: [10.1007/s10664-008-9102-8](https://doi.org/10.1007/s10664-008-9102-8).
- [38] D. Istrate, C. Taramasco, A. Fleury, N. Houmani, M. Hariz, and J. Boudy, "Well-being and ageing with chronic disease—The BV2 project," *Journées d'Etude sur la Télésanté*, pp. 1–5, May 2019.
- [39] C. Taramasco, T. Rodenas, F. Martinez, P. Fuentes, R. Munoz, R. Olivares, V. H. C. De Albuquerque, and J. Demongeot, "A novel monitoring system for fall detection in older people," *IEEE Access*, vol. 6, pp. 43563–43574, 2018, doi: [10.1109/ACCESS.2018.2861331](https://doi.org/10.1109/ACCESS.2018.2861331).
- [40] A. El Murabet, A. Abtoy, A. Touhafi, and A. Tahiri, "Ambient assisted living system's models and architectures: A survey of the state of the art," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 32, no. 1, pp. 1–10, Jan. 2020, doi: [10.1016/j.jksuci.2018.04.009](https://doi.org/10.1016/j.jksuci.2018.04.009).
- [41] H. K. Ngankam, H. Pigot, M. Parenteau, M. Lussier, A. Aboujaoudé, C. Laliberté, M. Couture, N. Bier, and S. Giroux, "An IoT architecture of microservices for ambient assisted living environments to promote aging in smart cities," in *Proc. Int. Conf. Smart Homes Health Telematics*, 2019, pp. 154–167, doi: [10.1007/978-3-030-32785-9_14](https://doi.org/10.1007/978-3-030-32785-9_14).
- [42] S. Ali, M. A. Jarwar, and I. Chong, "Design methodology of microservices to support predictive analytics for IoT applications," *Sensors*, vol. 18, no. 12, p. 4226, Dec. 2018, doi: [10.3390/s18124226](https://doi.org/10.3390/s18124226).
- [43] G. Cherradi, A. E. Bouziri, A. Boulmakoul, and K. Zeitouni, "Real-time microservices based environmental sensors system for Hazmat transportation networks monitoring," *Transp. Res. Procedia*, vol. 27, pp. 873–880, Jan. 2017, doi: [10.1016/j.trpro.2017.12.087](https://doi.org/10.1016/j.trpro.2017.12.087).
- [44] M. Taneja, N. Jalodia, J. Byabazaire, A. Davy, and C. Olariu, "SmartHerd management: A microservices-based fog computing-assisted IoT platform towards data-driven smart dairy farming," *Softw., Pract. Exper.*, vol. 49, no. 7, pp. 1055–1078, Jul. 2019, doi: [10.1002/spe.2704](https://doi.org/10.1002/spe.2704).
- [45] R. Wojcik, F. Bachmann, L. Bass, P. Clements, P. Merson, R. Nord, and B. Wood, "Attribute-driven design (ADD), version 2.0," *Softw. Eng. Inst., Carnegie-Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU/SEI-2006-TR-023*, 2006.
- [46] W. Wood, "A practical example of applying attribute-driven design (ADD), version 2.0," *Softw. Eng. Inst., Pittsburgh, PA, USA, Tech. Rep. CMU/SEI-2007-TR-005*, 2007.
- [47] C. Richardson, *Microservices Patterns*. New York, NY, USA: Manning, 2018.

- [48] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "The KDD process for extracting useful knowledge from volumes of data," *Commun. ACM*, vol. 39, no. 11, pp. 27–34, Nov. 1996, doi: [10.1145/240455.240464](https://doi.org/10.1145/240455.240464).
- [49] H. Calderon-Gomez, L. Mendoza-Pitti, M. Vargas-Lombardo, J. M. Gomez-Pulido, J. L. Castillo-Sequera, J. Sanz-Moreno, and G. Sencion, "Telemonitoring system for infectious disease prediction in elderly people based on a novel microservice architecture," *IEEE Access*, vol. 8, pp. 118340–118354, 2020, doi: [10.1109/ACCESS.2020.3005638](https://doi.org/10.1109/ACCESS.2020.3005638).
- [50] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Heidelberg, Germany: Springer-Verlag, 2012.
- [51] M. Waseem, P. Liang, and M. Shahin, "A systematic mapping study on microservices architecture in DevOps," *J. Syst. Softw.*, vol. 170, Dec. 2020, Art. no. 110798, doi: [10.1016/j.jss.2020.110798](https://doi.org/10.1016/j.jss.2020.110798).
- [52] T. L. Saaty, "What is the analytic hierarchy process?" in *Mathematical Models for Decision Support* (NATO ASI Series), vol. 48, G. Mitra, H. J. Greenberg, F. A. Lootsma, M. J. Rijkaert, H. J. Zimmermann, Eds. Berlin, Germany: Springer, 1988, doi: [10.1007/978-3-642-83555-1_5](https://doi.org/10.1007/978-3-642-83555-1_5).
- [53] H. Magsi, A. H. Sodhro, F. A. Chachar, S. A. K. Abro, G. H. Sodhro, and S. Pirbhulal, "Evolution of 5G in Internet of medical things," in *Proc. Int. Conf. Comput., Math. Eng. Technol. (iCoMET)*, Mar. 2018, pp. 1–7, doi: [10.1109/ICOMET.2018.8346428](https://doi.org/10.1109/ICOMET.2018.8346428).
- [54] D. Garlan, S. Khersonsky, and J. S. Kim, "Model checking publish-subscribe systems," in *Proc. Int. Spin Workshop Modeling Checking Softw.*, 2003, pp. 166–180, doi: [10.1007/3-540-44829-2_11](https://doi.org/10.1007/3-540-44829-2_11).



architecture schools. He is working in the research fields of architectural tactics, patterns, microservice architectures, technical debt, and security in telehealth systems.

GASTÓN MÁRQUEZ is currently pursuing the Ph.D. degree in informatics engineering with Federico Santa María Technical University, Chile. He has worked in financial companies for five years. He was a Research Visitor with the Rochester Institute of Technology, Rochester, NY, USA, and the Université de Technologie de Compiègne, Compiègne, France. He has authored or coauthored in several international conferences and has participated in international software



Paulo, Brazil, and an academic with UTFSM since 2003, where he is responsible for UTFSM's Software Engineering academic activities. He is the Chair of the Doctorate in informatics engineering, UTFSM, and the Co-Chair of BPM Center, UTFSM. He is a Principal Investigator of the Toeska Research and Development Team, which conducts teaching, research, and technology transfer in software architecture, semantic software systems, and software process improvement. His main research and development interests are identification, recovery, and reuse of architectural decisions and architectural knowledge (especially architectural tactics). He is currently the Executive Secretary of CLEI and a member of IFIP TC2 (Software Engineering) and the ACM, IASA, and INCOSE.

HERNÁN ASTUDILLO (Member, IEEE) received the Ph.D. degree in information and computer science from Georgia Institute of Technology, in 1995. He was an Informatics Engineer with the Universidad Técnica Federico Santa María (UTFSM), Santiago, Chile, 1988. He is currently a Professor of informatics with UTFSM. He has been a Lead Applications Architect with MCI Systemhouse and Financial Systems Architects, USA, a Visiting Professor with the Universidade de Sao



VINCENT ZALC worked on MRI acquisition and biomedical image processing for ten years. He then joined the eBioMed Chair, Connected Biomedical Tools. He has been an Engineer with the Biomechanics and Bioengineering Laboratory, UMR 7338, UTC, since 2007. He is working on multi sensors acquisition and signal processing.



along with scientific divulging. Her main academic interests are health, including mHealth, ambient-assisted living for elderly persons, e-health, telemedicine and telerehabilitation, and supervision of chronic diseases, and complex social systems, including dynamic networks, socio-semantic networks, and analysis of trajectories both individual and collective, among others.

CARLA TARAMASCO received the B.Eng. degree in computer engineering from the Universidad de Valparaíso, Chile, in 2001, the M.Sc. degree in cognitive science from the École Normale Supérieure in 2006, and the Ph.D. degree (*summa cum laude*) from the École Polytechnique, France, in 2011. She is currently a Researcher and a Professor with the Department of Computer Science, Universidad de Valparaíso. She currently teaches both at the undergraduate and graduate levels,



DAN ISTRATE is currently a Professor with the Biomechanics and Bioengineering Laboratory, UMR 7338, UTC, and is also in charge of the eBioMed Chair, Connected Biomedical Tools. He is working on applications for the supervision of elderly people at home using different connected sensors, functional rehabilitation, and monitoring of pregnant women for predict premature delivery. He is a scientific expert of several national and European calls.

...