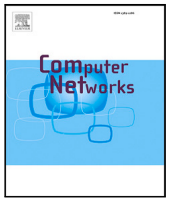
Computer Networks 251 (2024) 110632

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



Definition and implementation of the Cloud Infrastructure for the integration of the Human Digital Twin in the Social Internet of Things ☆

Roberto Girau [c,*], Matteo Anedda [a], Roberta Presta [d], Silvia Corpino [a], Pietro Ruiu [b], Mauro Fadda [b], Chan-Tong Lam [e], Daniele Giusto [a]

[a] DIEE (UdR CNIT) University of Cagliari, Cagliari, 09123, Italy
[b] University of Sassari, Sassari, 07100, Italy
[c] DISI University of Bologna, Cesena, 47521, Italy
[d] Scienza Nuova Research Centre, University Suor Orsola Benincasa, 80135 Naples, Italy
[e] Faculty of Applied Sciences, Macao Polytechnic University, Macao Special Administrative Region of China

ARTICLE INFO

Keywords:
Social IoT
Heterogeneous networks
Digital twins

ABSTRACT

With the integration of virtualization technologies, the Internet of Things (IoT) is expanding its capabilities and quickly becoming a complex ecosystem of networked devices. The Social Internet of Things (SIoT), where intelligent things include social properties that improve functioning and user engagement, is the result of this progress. The SIoT still has issues with scalability, data management, and user-centric operations, despite tremendous progress. In order to overcome these obstacles, a strong architecture is needed that can handle the enormous number of IoT devices while simultaneously streamlining the user interface.

This study provides a unique architecture for the IoT that uses containerization to efficiently deploy and manage services while integrating Virtual Users (VUs) and Social Virtual Objects (SVOs) into a scalable Cloud/Edge infrastructure. These innovative aspects collectively advance previous works presented in literature and focused on novel SIoT architectures and implementations, by addressing key challenges in scalability, efficiency, and automation within the SIoT. The proposed method presents an extensible, modular architecture that lets VUs self-manage IoT services, making user administration easier and improving system security and scalability. Important parts of the design include a host controller for container orchestration, a deployer for automated service deployment, and user clusters for aggregating VUs, SVOs, and apps to provide secured and efficient data sharing. We show through experimental assessment that the architecture can manage high-volume installations and operating needs, exceeding the conventional platform based on Google App Engine in terms of system overhead and deployment timeframes. The obtained results highlight how our suggested architecture, which provides an easy-to-use, scalable, and secure foundation for IoT deployments, has the potential to advance the SIoT landscape.

1. Introduction

The Internet of Things (IoT) [1] represents a transformative technological paradigm where everyday objects are connected to the internet, enabling them to send and receive data. These devices, integral to the IoT, are context-sensitive and can be recognized, detected, and remotely controlled. A key aspect in this domain is the integration of virtualization technologies [2], which enable sensing, communicating, acting, interacting, and exchanging knowledge in the Cloud, reflecting the informational content from their real-world counterparts.

This integration allows Internet-connected devices to collect information about the physical world and transfer it to the virtual realm, facilitating a wide range of applications and services. However, the rapidly growing number of Internet-connected objects and the consequent surge in data volume necessitate a redesign of future systems to effectively process and store this burgeoning information.

☆ This work has received financial support under the National Recovery and Resilience Plan (NRRP), Mission 4, Component 2, Investment 1.1, Call for tender No. 1409 published on 14.9.2022 by the Italian Ministry of University and Research (MUR), funded by the European Union – NextGenerationEU– Project Title "METATwin - Metaverse & Human Digital Twin: digital identity, Biometrics and Privacy in the future virtual worlds", – CUP J53D23015030001- Grant Assignment Decree No. 0001382 adopted on 01/09/2023 by the Italian Ministry of University and Research (MUR).

* Corresponding author.
E-mail address: roberto.girau@unibo.it (R. Girau).

https://doi.org/10.1016/j.comnet.2024.110632
Received 6 April 2024; Received in revised form 14 June 2024; Accepted 3 July 2024
Available online 8 July 2024

While virtualization provides essential scalability and flexibility for numerous IoT services, recent developments indicate that this approach alone is insufficient. Incorporating social networking concepts into IoT offers new strategies to address the challenges of the IoT ecosystem [3]. This has led to the emergence of the Social Internet of Things (SIoT), a network of intelligent objects and services with social characteristics.

SIoT, akin to human social networks, enhances the functionalities of smart devices with relational capabilities, presenting a viable solution for managing and exploiting the anticipated multitude of heterogeneous devices. As our interactions with these devices increase, it becomes crucial to alleviate the management burden on users. This is where the concept of the VU [4] comes into play, autonomously integrating the user into the digital world and assuming most of their operational responsibilities, based on continuous learning of user behavior and automated processes.

Despite the potential of the VU concept within the SIoT paradigm, there is a lack of robust implementations. This work aims to present, describe, and test an architecture that aligns with the VU requirements and offers a modular, extendable Cloud infrastructure for robust, rapid, and secure communication at all levels.

This research is characterized by several innovative aspects:

1. **Cloud Infrastructure for VU Concept:** This research introduces the first cloud infrastructure solution that fully meets the fundamental requirements of the Virtual User (VU) concept. This entails designing a robust, scalable, and efficient cloud-based system capable of supporting the extensive and dynamic interactions typical within a VU, thereby facilitating seamless connectivity and resource management.
2. **Containerization-based Virtualization in SIoT:** The research leverages advanced virtualization technologies through a containerization-based approach. Containers offer lightweight, efficient, and portable execution environments that enhance the deployment and management of IoT applications. This method ensures better resource utilization, faster startup times, and improved scalability compared to traditional virtualization techniques.
3. **Automated Service Setup Processes:** A key innovative aspect is the automation of service setup processes within the SIoT. This is achieved by the functional elements of the architecture, which manage and streamline the configuration, deployment, and maintenance of services. Automation reduces the manual effort required, minimizes errors, and accelerates the deployment of IoT services, thus enhancing operational efficiency.

How can we improve the SIoT for user-centric applications so that users do not need to continuously update their profiles, considering the wide variety of existing devices? What specific benefits would implementing an HDT (Hypermedia-Driven Thing) solution in conjunction with SIoT bring? The main scientific aspect of this research lies in the integration and novel application of cloud infrastructure and virtualization technologies within the SIoT framework. These innovative aspects collectively advance the field by addressing key challenges in scalability, efficiency, and automation within the SIoT, paving the way for more effective and manageable IoT ecosystems. The paper is structured as follows: The first section reviews the state of the art, introducing and analyzing the concepts of VU and SIoT. The second section details the proposed Cloud infrastructure, addressing the needs and gaps identified in the literature. This includes the functional modules, communication interfaces, and key operational processes like Cloud service distribution. The third section evaluates the infrastructure's effectiveness through experimental results, demonstrating its performance capabilities and potential for future developments. The paper concludes with a summary of findings and implications.

2. Background and key concepts

In the "always connected" paradigm, society increasingly relies on numerous internet-accessible objects, generating vast data requiring efficient storage, processing, and interpretation. Cloud computing emerges as a key technology, offering solutions for managing sensors, data storage devices, analytical tools, artificial intelligence, and management platforms, particularly through device virtualization.

The Virtual Objects (VOs) [5] represent digital counterparts of physical devices in the Cloud, introducing functionalities beyond real-world objects' capabilities, such as service discovery, complex application creation, energy management efficiency, and support for multilingual object communications.

These VOs serve as abstractions, acting as an intermediary layer between physical devices and the applications or services interacting with them. By dynamically updating to reflect changes in the state, configuration, or location of their physical counterparts, VOs enable real-time adaptation and seamless integration. Standardized protocols promote interoperability, ensuring effective communication and collaboration between diverse IoT devices and platforms. Integrated into cloud and edge computing environments, VOs facilitate efficient processing, storage, and management of data within the IoT ecosystem. They also enhance user experience by allowing users, applications, or other devices to interact with VOs as if directly engaging with the physical devices. This concept supports scalability, enabling the virtual representation of the IoT environment to adapt to an increasing number of connected devices. In essence, VOs play a pivotal role in simplifying complexity, providing a standardized interface, and streamlining data management and analysis within the evolving landscape of the Internet of Things.

Recent studies [6] suggest applying social networking concepts to manage heterogeneous devices effectively. The SIoT concept [7], formalized in [8], envisions a social network where nodes (objects) autonomously establish social relationships based on owner-defined rules. This model promises scalability and reliability by correlating informational data with social relationships, enhancing service search, selection, and composition within IoT communities.

Pioneering architectures like Atzori et al.'s three-tier model [9] comprise Object, Component, and Application Layers. This logic is echoed in subsequent proposals [10,11], adapting SIoT to various domains like IoV [12] and E-health [13]. Lysis, a Cloud platform based on [9], is particularly relevant, integrating Virtual Objects (VOs) into Social Virtual Objects (SVOs) [14].

Lysis is a cloud platform utilizing the PaaS model, enabling easy programming and reusability. It implements VOs and SVOs, with SVOs being central, equipped with interfaces for secure connections with Real World Objects (RWOs) and standardized communication procedures.

SVOs, detailed in [14,15], possess functionalities influenced by their associated RWOs. They include a Social Enabler (SE) module for managing social relationships, forming friendships based on static owner information, location, and encounter frequency. Types of social relationships include Social Object Relationship (SOR), Ownership Object Relationship (OOR), Parental Object Relationship (POR), Co-location Object Relationship (CLOR), and Co-work Object Relationship (CWOR). SVOs interact autonomously with community members, enhancing system scalability.

Girau et al. [4] introduced the VU concept, a distributed agent in the Cloud representing IoT users' virtual counterparts. The VU aims to relieve users from tedious IoT service management tasks, acting on their behalf and fully exploiting IoT potential aligned with user profiles and interests. The VU provides a unified interface, maintains an updated user context, builds user profiles, manages credentials, and creates personalized Quality of Experience (QoE) models. It supports intelligent mobility in edge-computing environments. Key components include Profile, Owner Control, ID&A Management, QoE, Social Network (SN) Management, Context Management, and Migration Management.

These components enable the VU to autonomously manage interactions with IoT services and applications, representing user interests and preferences in the Cloud.

In summary, the integration of Cloud computing, SIoT, and virtualization technologies like VOs and SVOs, along with the concept of VU, represents a significant advancement in managing the increasing complexity and potential of IoT environments.

3. State of the art

This section briefly describes the initial implementation proposals that have utilized the concept of the VU in recent years.

Girau et al. [4] propose a three-level implementation scheme to integrate the VU into an IoT–cloud system. The base level involves a database for storing context descriptors and the user's DNA, a numerical matrix containing unique user characteristics. The component level includes tools for profiling, owner control, ID and authorization management, Social Network management, context management, and migration management. The top layer comprises the user interface and interfaces with other IoT components: VO, CVO, and apps.

Functionally, the VU profiles are created and updated based on user monitoring by ICT objects, which convert physical characteristics into digital descriptors. This process relieves users from manual intervention, as the knowledge obtained is stored and processed for each user by their respective VU, ensuring flexibility and privacy.

As key processes in VU Implementation we can mention:

1. Continuous data collection at a certain sampling frequency.
2. Calculation of profile portions associated with classifier clusters after sufficient data collection.
3. Detection of preferred settings related to clusters and storage in the DNA.
4. Transmission of settings to the user's VOs.
5. Daily definition of information to add to the profile based on collected data.

Another work integrates the VU with Social IoT, using a new Chatbot integrated with Telegram [16] and compliant to the Lysis platform. This aims to improve user experience and platform efficiency by simplifying interaction through text messages. The VU in this setup interacts with all architecture modules, including Lysis, creating a social network among virtual objects to facilitate interaction. The chatbot system acts as a back-end service for communication between users and the VU. Requests are generated by web apps or messaging services, processed by an AI-based module, and then directed to the necessary function. This architecture simplifies user-platform-SVO interaction, especially for challenging actions like setup and configuration.

However, these initial models and implementations do not yet provide a comprehensive architectural and functional solution for the VU and its associated modules. The development and experimentation process, which will be the subject of the following sections, aims to address these gaps.

3.1. Digital Twin in the IoT

The concept of the Digital Twin (DT) has its roots in the manufacturing sector [17], where it is defined as a virtual representation that encompasses the entire lifecycle of a product. This includes design, prototyping, testing, and production phases. The physical and digital/virtual counterparts are closely linked, allowing for comprehensive design, experimentation, and analysis of the real object's physical characteristics at any stage of its lifecycle. The goal is to obtain any information from the DT that could be obtained from inspecting the physical product [18].

Minerva et al. [19], building on the definition from [20], describe the DT as: *"A complete software representation of a single physical object, including its properties, conditions, and behavior in real life, through models and data. A DT is a set of realistic models that can simulate the behavior of an object in a distributed environment, representing and reflecting its physical twin throughout its entire lifecycle"*.

This definition implies that a DT is intricately connected to its physical counterpart, containing all necessary information to complete, characterize, and document its history. The concept of DT has evolved, with its definition and applications expanding significantly [19].

In large systems, a DT should represent and behave like the real object, adapting to changing conditions over time. The connection between the real object and its virtual twin becomes crucial, especially post-production, where physical data must be continuously fed to the DT. The advent of advanced communication capabilities has enabled DTs to connect with their physical counterparts via the Internet or other protocols, facilitating the creation of specific networks. Additionally, the DT can store and analyze the history of the object's behavior, aiding in the improvement of design, production, and operation for future iterations [21]. For large systems, distributing virtual systems across different computing environments (edge, fog, cloud computing) is essential for simulation, stress testing, and error detection [22].

Given these characteristics and advantages, the DT concept has garnered significant interest in IoT, Industrial IoT, and Cyber–Physical Systems (CPSs). There is a strong demand for adaptive system solutions, and DTs could play a key role in developing products and controlling processes [23–25]. [26] illustrates a general framework that incorporates many capabilities and functions identified in academic and industrial proposals.

[27] proposes a DT-assisted resource allocation framework for Vehicular Edge Computing (VEC) to enhance network performance by enabling collaborative computing among edge nodes. It integrates DT technology to create virtual replicas of network nodes, improving real-time condition monitoring and resource allocation. The framework employs Deep Reinforcement Learning (DRL) for decision-making, specifically using an Advantage Actor–Critic (A2C) algorithm to solve the optimization problem. Simulation results demonstrate the framework's effectiveness in reducing task completion delay and increasing VEC system computation rates compared to benchmark approaches.

The authors in [28] discuss the integration of Adaptive Digital Twins (ADT) with Vehicular Edge Computing (VEC) to address challenges like high mobility and dynamic environments in vehicular networks. It proposes a three-layer ADT-VEC framework, utilizing Deep Reinforcement Learning (DRL) for efficient resource allocation and task offloading, aiming to minimize latency. The framework demonstrates effectiveness through simulations, offering a novel approach to enhancing VEC networks with adaptive and intelligent management capabilities.

SOL, a service offloading method using deep reinforcement learning, for Digital Twinning-Empowered Internet of Vehicles (IoV) in Edge Computing has been introduced in [29]. The aim is to enhance the quality of service (QoS) in the IoV by efficiently managing service offloading in edge computing environments. SOL employs Deep Q-Network (DQN) to make optimized offloading decisions, addressing the challenge of excessive service requests that could potentially overload edge computing devices (ECDs). Through real-world datasets, the effectiveness and adaptability of SOL in various environments are demonstrated, showcasing its potential to improve IoV services by leveraging edge computing.

Zhang et al. [30] introduce an innovative framework that integrates DT technology and Artificial Intelligence to enhance vehicular edge computing networks. This approach aims to optimize edge service management by revealing potential service matches among vehicles and simplifying management complexities. The framework employs a gravity model for vehicle aggregation in the DT environment and a multi-agent learning algorithm for efficient resource scheduling in physical networks. The performance of these schemes, validated through real traffic datasets, demonstrates improved efficiency and lower costs compared to traditional methods.

[31] introduces a novel framework for reducing offloading latency in Digital Twin Edge Networks (DITEN) within 6G environments. It leverages DTs of edge servers to improve mobile offloading decisions, minimizing latency while considering service migration costs due to user mobility. The framework employs Deep Reinforcement Learning, specifically an Actor–Critic approach, to dynamically optimize offloading decisions. Simulation results demonstrate the effectiveness of this approach in reducing offloading latency, failure rates, and service migration rates, enhancing overall network performance.

Wang et al. [32] present a cooperative computing framework for DT enabled 6G Industrial IoT, focusing on End–Edge–Cloud (EEC) collaborative computing. It proposes a DT-assisted scheme for optimizing device association, offloading modes, bandwidth allocation, and task split ratio to minimize system costs, considering task latency and energy consumption. A Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithm is employed for collaborative computation and resource allocation, demonstrating significant improvements in task success rates, reduced latency, and energy consumption through simulations.

In [33], the authors investigate the strategic positioning of social digital twins (SDTs) within edge computing environments to leverage the advancements in Beyond 5G (B5G) IoT networks. This research introduces the concept of SDTs, which are sophisticated digital replicas that not only mimic the physical and operational characteristics of devices or entities but also incorporate social interactions and behaviors, enhancing the IoT ecosystem's intelligence and responsiveness. By deploying these SDTs at the network edge, the study addresses critical challenges in B5G networks, such as latency, data congestion, and privacy concerns, by enabling more localized and context-aware data processing and decision-making.

In case of DT of a person, namely Human Digital Twin(HDT), it consists of the replication of the individual human body, in its entirety or in a subset, in virtual space, simultaneously reflecting its physical and psychological state in real-time [34].

To the best of our knowledge, there are currently no existing works specifically addressing HDT in the context of the SIoT. To address the gaps in the current state of the art, we have critically analyzed recent advancements in the HDT within IoT.

A HDT can be employed to monitor the health status and well-being of a person, provide personalized support for health and well-being, and to monitor and improve people's safety, for example, by preventing dangerous situations, such as a driving accident, or in case of emergency, the HDT can automatically send an emergency call to the relevant emergency services [35].

In [36], the authors delve into the sophisticated utilization of DT technology to encapsulate human components within the scheduling processes of cyber–physical production systems. This study presents an approach where DTs serve as dynamic human representations, facilitating more informed and adaptive scheduling decisions. By embedding human factors into the DT, the research highlights how the intricate interactions between human operators and the production environment can significantly influence scheduling efficiency and system adaptability.

The study [37] explores an innovative integration of DT technology with deep reinforcement learning to enhance human activity recognition systems in the context of mobile edge computing. The research introduces a cost-effective framework that employs DTs as virtual replicas of physical entities, enabling simulations and predictions of human activities. By leveraging the computational capabilities of edge devices, this approach not only addresses the latency and privacy concerns inherent in cloud-based systems but also significantly reduces the system's overall expenses. The employment of deep reinforcement learning further refines the activity recognition process, adapting dynamically to new data and environments, thus ensuring high fidelity in the recognition outcomes.

3.2. Virtual User and Human Digital Twin: A comparison

The Virtual User (VU) foreseen in [4] is more like a virtual assistant, a copilot that interact with multiple IoT devices, applications and the user herself. In this view, we can highlight their main differences:

- Purpose and Functionality:
  - HDTs: These are complex simulations that represent a virtual counterpart of a human being, integrating multidimensional data to replicate and predict the physical, biological, and behavioral aspects of the individual. They are primarily used in healthcare, personalized medicine, and performance optimization, among other fields.
  - VUs: These are AI-powered software agents designed to assist users in tasks, typically through voice or text interactions. Their functions include setting reminders, answering questions, controlling smart devices, and facilitating user interaction with technology.
- Data and Complexity:
  - HDTs: They require comprehensive, high-fidelity data encompassing genetic information, medical history, lifestyle, environmental factors, and possibly real-time physiological data. The complexity of a HDT is substantial, as it aims to accurately model and predict human physiology and behavior.
  - VUs: While they may personalize responses based on user data and preferences, the depth and scope of data integration are generally less complex than that of HDTs. Virtual assistants primarily rely on user input, preferences, and interaction history.
- Interactivity:
  - HDTs: The interaction with HDTs is usually more analytical and data-driven, often used by healthcare professionals, researchers, or the individuals themselves to make informed decisions about health, training, or lifestyle adjustments.
  - VUs: These are designed for interactive, user-friendly experiences, allowing lay users to easily communicate with digital devices and services. The interaction is more about facilitating everyday tasks and accessing information.
- Application Areas:
  - HDTs: Their applications are highly specialized, focusing on areas like personalized healthcare, biomechanics, pharmaceuticals, and complex system simulations in which individual-specific outcomes are crucial.
  - VUs: They have a broad range of applications in consumer electronics, smart homes, customer service, and general productivity tools, aimed at a wide consumer base.
- Development and Implementation:
  - HDTs: The development involves interdisciplinary expertise, including medicine, biotechnology, data science, and computational modeling, to achieve a high degree of accuracy and predictive capability.
  - VUs: : Development focuses on natural language processing, user experience design, and integration with various platforms and services, aiming for accessibility and ease of use.

In summary, while both HDTs and VUs are innovative digital technologies, they differ significantly in their purpose, complexity, interactivity, application areas, and development focus. For these reasons, even if tempted by considering a VU as a HDT, in this work we consider

them distinct concepts. Nonetheless, a VU should exploit a HDT in order to predict the human behavior and preferences in the different scenarios of application. To this, the VU will involve a HDT as a building part of its architecture.

3.3. Considerations

The analysis of the state of the art reveals that virtualization technologies are extensively used in the ICT sector [38]. Their ability to simplify the management of numerous physical entities connected to the Internet and participating in the IoT community is invaluable. These technologies facilitate the handling of the continuous stream of data and information produced and several solutions can be found in literature.

[39] proposed an architecture for virtual objects management in hyperconnected things network to facilitate the management tasks. In [40], the authors had the objective to bring cloud-computing services much closer to the end-users replacing physical devices with their Virtual counterparts.

However, it has become evident that the benefits derived from using virtual counterparts, often consisting of simple digital representations, are becoming insufficient. This necessitates an extension of the functionalities offered so far.

Among the solutions proposed in the literature, the application of human social networking processes and rules to the network of intelligent devices, used in everyday life and various contexts like industrial and medical, is particularly intriguing. The introduction of the SIoT lays the groundwork for developing key elements for the proposed architectural solution: the SVO, the Lysis platform, and the VU. While the VU emerges as a general concept, it seamlessly integrates into the SIoT paradigm.

In [41], the authors proposed a framework for SIoT objects to enable virtual representation of real world objects analyzing their relationship semantically to compose new services by combining VOs and named composite VOs. Compared with Lysis, the architectural scheme is not complete but limited to SVO definition.

The ability to virtualize the user, replacing them in most configuration and data/device manipulation activities, and activating learning processes based on past decisions and actions, is significantly enhanced when extended through varied relationships with other VUs and objects. This approach optimizes service search, leveraging the knowledge of the owner's VU, the VUs of connected friends, and the information collected by SVOs.

Implementing the VU requires innovative technologies and techniques such as user profiling, artificial intelligence, machine learning, and classification algorithms. This presents a challenge, especially within the SIoT context. An architecture that includes the VU must provide a distributed, scalable environment that is modular and versatile, capable of managing additional elements and protocols.

Security is another critical aspect. The modules constituting the functional parts in the Cloud exchange sensitive information and are vulnerable to cyber attacks. As system complexity and functionalities increase, so does the risk of security breaches. This opens avenues for new research proposals, including risk analysis specifically applied to the VU and SVOs, and the development of encryption algorithms for securing communications within the VU network.

Further advancements could include defining a complete semantics specific to the VU, enabling clear and immediate understanding of exchanged information within its network. Additionally, legislative gaps regarding the governance of VU communities need addressing. The autonomous nature of the VU, its ability to react and adapt, while maintaining its identity and interacting with other agents, raises important legal questions about the responsibility for the activities of the VU.

In Table 1, a comparison with the state of the art concerning the SIoT is provided. This table highlights the main innovative contributions of the present work in comparison to existing studies. The comparison was conducted by selecting several works from the literature that address SIoT, offering insights into possible architectures and implementations. The table is structured to illustrate how the present work advances beyond existing research by focusing on key aspects such as virtualization technologies adopted, cloud/edge based architecture, scalability, automation and the integration with VU. Each selected work is analyzed based on these criteria to demonstrate the novel contributions of the current study. Table 1 provides a clear, comparative overview, showcasing the innovative contributions and contextualizing them within the broader landscape of SIoT research.

Table 1
Related Works — Comparison.

References	Topic	SIoT SotA	Improvements
[4,5] [8,10] [14,15]	Infrastructure	Cloud/Edge -based	Enhanced robustness, scalability, and efficiency of cloud-based system
[4,5] [10,14] [15,16] [40]	Virtualization Technologies	VM Based	Advanced virtualization technologies through a containerization based approach
[4,11] [16]	VU concept	N/A	Fully integration with VU concept
[3,16]	Automation	N/A	Automation of service setup processes

3.4. Strategic requirements

When developing and deploying the VU in the SIoT, several strategic requirements must be considered.

A key requirement is a versatile infrastructure that supports the migration of functional modules within the architecture, particularly with an eye towards edge computing [42–44]. Edge computing aims to bring services closer to the data source (the customer), offering low latencies.

Research on container migration strategies in edge computing is gaining momentum, which is crucial for the VU concept realized through virtualizations and containers [45–47].

In the analysis of the current implementations of the SIoT we faced some challenges:

- The first version of Lysis [14] utilized Google App Engine (GAE) PaaS for container deployment. While GAE offers a user-friendly environment and key APIs, it has limitations, such as restricted language support, limited control over the infrastructure, and poor performance for high computational demand applications [48].
- Alternatives like AppScale, while not limited to Google's proprietary Cloud, present similar limitations and are not economically feasible for applications requiring substantial resources.

Past works present significant shortcomings from the point of view of structural requirements and the choices made for the distribution of the infrastructure itself. In particular, some strategies have been identified to overcome the gaps found in the scientific literature, focusing on the use of:

- lightweight, easily extendable virtualization technology, using an architectural logic centered on the use of containers [49,50];
- automated processes for the distribution, configuration, and potentially migration of services [51].

Defining an infrastructure based on containerization offers significant advantages, especially in view of future developments; such a choice allows for continuous extension of the infrastructure through the addition of further functional modules, such as Artificial Intelligence,

Machine Learning, and User Profiling modules. Moreover, there are various techniques for migrating containers in edge and fog computing [52,53], which is a fundamental characteristic for a comprehensive implementation of the VU concept. Furthermore, it offers versatility in design, as it guarantees total management of the structure and its architectural properties, which translates into the ability to provide the developer with almost complete control over the Cloud architecture and the technologies it uses. The necessity to fully exploit the potential of containerization-based virtualization motivates the choice to propose a microservices architecture, which, unlike a monolithic architecture, breaks down the total application into a collection of smaller independent units [54]. Each of the units within the architecture processes its own services separately, meaning that each service has its own functional logic and direct access to the database, where necessary. Such a choice [55] responds to the needs developed from the state of the art and brings a series of benefits that, in this implementation, are fundamental:

- the deployment and updating of the individual service can be performed individually;
- an error within a single block or module will not affect any other construct within the overall application. Moreover, integrating new projects into a microservices architecture is not complicated;
- simpler understanding: by reducing the design to smaller, easily understandable blocks, the management of the entire structure becomes simpler;
- scalability: the microservices approach allows for independent scaling of each block or module, unlike a monolithic system where the entire application must be scaled even where it is not necessary;
- flexibility: the choice is not limited to specific development technologies or programming languages. The developer is free to make different choices depending on the needs of each service;
- agility: fault tolerance is generally guaranteed in microservices, as errors concern individual components and do not affect the entire process.

Since, one of the main purposes of the introduction of the VU is to build efficient user centric SIoT application, grouping VUs and SVOs by owner into a "User Cluster" of microservices offers several advantages in terms of implementation and management in SIoT environments. Here are some benefits of this approach:

- Isolation and Security: By grouping VUs and SVOs belonging to the same owner into a User Cluster, it allows for better isolation of data and services. This isolation enhances security by reducing the attack surface and limiting the potential impact of security breaches or vulnerabilities.
- Efficient Resource Allocation: User Clusters enable more efficient resource allocation by consolidating the microservices that serve a particular user. This consolidation reduces overhead and improves resource utilization, leading to better performance and scalability of the SIoT infrastructure.
- Simplified Management: Managing a User Cluster as a single entity simplifies administration tasks such as deployment, monitoring, and maintenance. System administrators can apply policies, updates, and optimizations uniformly across all microservices within the cluster, streamlining operations and reducing management overhead.
- Improved Interoperability: Grouping VUs and SVOs by owner promotes interoperability and seamless communication between related microservices. This coherence enables smoother interactions and data exchange within the SIoT ecosystem, enhancing user experience and system functionality.
- Scalability and Flexibility: User Clusters facilitate scalable and flexible deployment architectures, allowing for dynamic scaling of resources based on user demand and workload fluctuations. This elasticity ensures that the SIoT infrastructure can adapt to changing user requirements and accommodate growth without sacrificing performance or reliability.
- Customization and Personalization: User Clusters provide a platform for implementing customizations and personalizations tailored to individual users' preferences and needs. By grouping VUs and SVOs together, developers can design and deploy specialized functionalities and services that cater to specific user requirements, enhancing the overall user experience. Customization and Personalization: User Clusters provide a platform for implementing customizations and personalizations tailored to individual users' preferences and needs. By grouping VUs and SVOs together, developers can design and deploy specialized functionalities and services that cater to specific user requirements, enhancing the overall user experience.
- Data Aggregation and Analysis: Aggregating data within User Clusters enables comprehensive analysis and insights generation for user-centric SIoT applications. By consolidating data from VUs and SVOs associated with the same owner, it facilitates holistic understanding and utilization of user context information, driving intelligent decision-making and action within the system.

Overall, grouping VUs and SVOs by owner into User Clusters of microservices offers numerous advantages in terms of security, efficiency, manageability, interoperability, scalability, customization, and data utilization. This approach enables the development and deployment of user-centric SIoT solutions that are robust, scalable, and responsive to individual user needs and preferences. In the following sections, these choices are translated into the actual structures and functional modules designed. Furthermore, some fundamentally important processes for the overall operation of the Cloud/Edge infrastructure will be characterized, namely: the deployment of services, the configuration of the SVOs, and the migration mechanism that can be applied to the latter.

4. Proposed edge/cloud infrastructure

The term "edge/cloud infrastructure" refers to a virtual infrastructure accessible via a network or the Internet. It comprises components like servers, storage systems, edge devices, networks, virtualization software, services, and management tools, supporting the computing requirements of a edge/cloud computing model.

This research proposes a Edge/Cloud architecture for the efficient integration of the VU into the SIoT, divided into the following structures:

- **Platform:** The VU is proposed as one or more microservices interacting with Lysis platform [14]. Lysis, as a PaaS service, offers functionalities for managing IoT resources using the concept of SVO. However, Lysis has limitations that can be addressed by integrating an intelligent distributed agent, the VU, into the SIoT architecture. Lysis platform exploits an inner module in order to deploy the SVOs into a cloud space from a repository. This module will need the ability of deploying VUs as well.
- **Virtualization:** Virtualization technologies are crucial in the Social IoT paradigm and the proposed Edge/Cloud infrastructure. The group of virtual entities like SVOs and the VU, the User Cluster, is a key element in the virtualization structure.
- **Host:** The hosting environment distributes the virtualization infrastructure. Different environments can host one or more groups, with each host management module responsible for controlling and orchestrating container management, migration, and execution within the group of virtual entities.

To extend Lysis for the VU and the User Cluster, the platform must integrate new functional elements. This includes defining an interface for information exchange between the platform and the VU, and extending the platform with a enhanced Deployer module. The Deployer

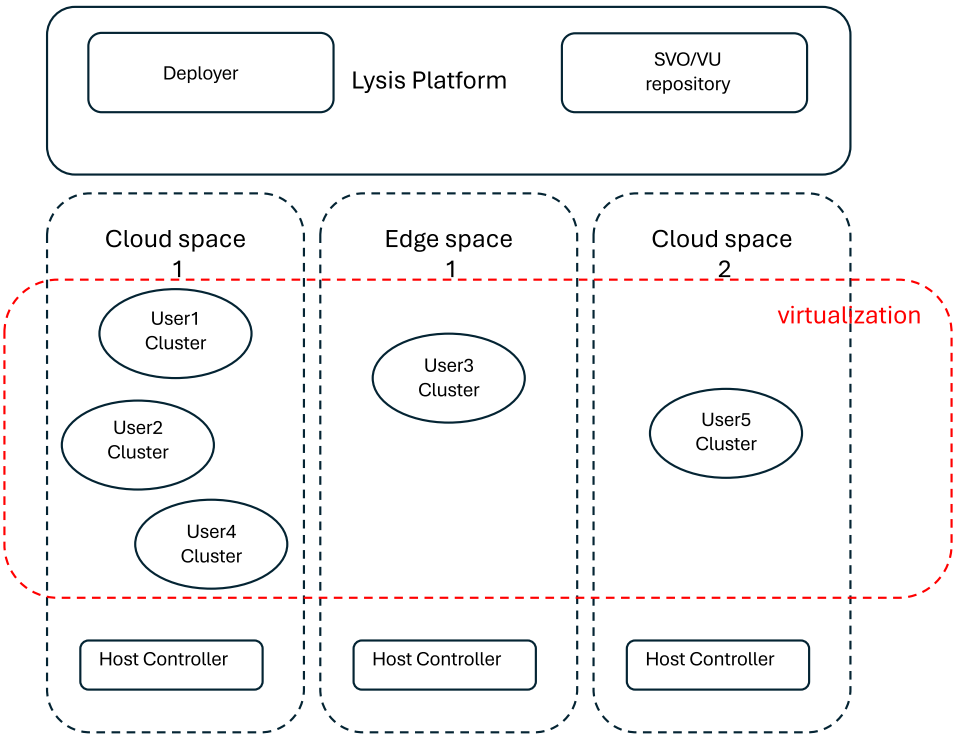


Fig. 1. Cloud/Edge Infrastructure.

manages service deployment requests from the users (through Lysis's web UI or VUs) and forwards them to the HC, which translates these requests into actual edge/cloud microservices.

The proposed Edge/Cloud infrastructure utilizes containers, identified as a lightweight and suitable solution for a distributed environment that allows for easy scalability. The Host Controller integrates logic for service deployment and migration. The platform can provide users with dedicated Cloud space, but users may also choose different machines, instances or edge devices to integrate into their personal execution and data storage space.

Thus, users have total control over the infrastructure, aligning with the identified requirements for the VU and its architecture. The detailed functional elements of the proposed infrastructure are schematized in Fig. 1.

The proposed solution adheres to the Lysis model and serves as an extension of the platform. A key aspect of this integration is establishing a communication channel between the VU and the management platform (i.e. Lysis platform).

4.1. Deployer

A critical step in making the infrastructure implementation effective is defining the elements necessary for an intelligent and automated deployment of services. This involves developing the steps, processes, and activities required to make the software system available to users. A key module in this process is the Deployer (Fig. 2(a)), an application integrated into Lysis that orchestrates deploy requests for new VUs, SVOs, and other services.

New VUs are deployed in the Cloud following a request from the Lysis platform, triggered by the registration of a new user. Once active, these VUs can request the deployment of SVOs and apps through the dedicated web interface. The Deployer manages these communications asynchronously. When a service deployment request is made, the Deployer translates it into a unique identifying string (a query) and adds it to the queue in the Query Management module. This string is sent back as an immediate response to the requesting module.

Since deployment can take a significant amount of time, the system is designed to initially send only the identifier of the request. The complete response, containing details like the URL of the new instance, is sent after the process is completed. This approach prevents the VU or Lysis platform from being unnecessarily occupied and avoids potential communication protocol exceptions due to response latency.

Each requesting module is responsible for tracking the information of the requested service until the deployment process concludes. The Deployer maintains information about each instantiated User Cluster, including its real-time status (active or inactive). This is achieved through specific data processing, which defines the attributes necessary to start the cluster in the hosting environment, including database access credentials and controls for future communications with the distribution module.

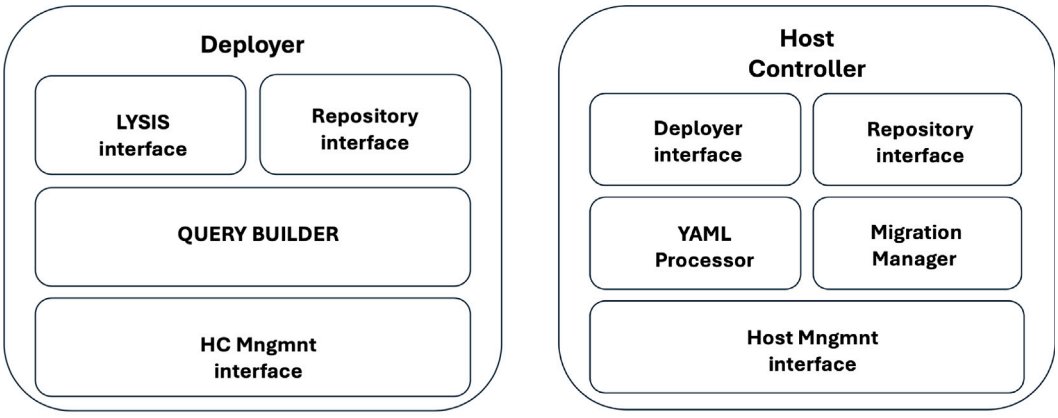
During communications with both upper and lower levels, the Deployer does not exchange any sensitive information, thereby maintaining user privacy and ensuring security in terms of interface.

4.2. Host Controller

The Host Controller (HC) is a service on the physical/virtual machine responsible for managing the deployment process in the hosting environment. It configures containers and interacts with the host system. The functional components of the HC are described below, as shown in Fig. 2(b).

The YAML Operator is a key component that handles reading, writing, and updating configuration files in YAML format. These files define services, networks, and volumes for multi-container applications. The YAML operator creates a directory for each user on the virtual machine, where it generates a configuration file based on a reference model. This model includes containers for user interface web applications and databases. Notably, the storage system is always associated with a volume to ensure data persistence.

The Host Management Interface manages operations processed directly on the machine, such as executing Shell commands, adding background processes, and searching for available ports. It monitors



(a) Deployer (b) Host Controller

Fig. 2. Main components for the deployment of SVOs and VUs in the edge/cloud architecture.

the execution of configurations contained in YAML files and identifies free ports for new services. This is crucial for deploying services like the User Interface, which need to expose a port for external communications.

The Deployer Communication interface plays a vital role in handling deployment requests from the Deployer. If recognized by the interface, the HC processes the request and adds the related background process. This communication is asynchronous, using a query queue system. The HC informs the Deployer when it starts the operation and then releases it from communication until the process is completed.

Migration Management is responsible for managing the migration process of the user's SVOs in edge computing environments. This module enables the VU to bring resources closer to the user, enhancing performance in terms of latency. HCs running on different machines or instances can communicate to facilitate the movement of services upon a migration request. The key activity here is transferring the persistent data collected by the SVO from the current environment to the new one, primarily involving the database transfer.

4.3. The User Cluster

The User Cluster is a collection of containers connected to the same network, providing a secure communication environment by avoiding the use of public IP for internal communications. This cluster is a key innovation in the proposed Cloud infrastructure, as it consolidates all the IoT services of a Lysis user in the virtual world, mirroring their management of devices and applications in the real world.

The services encompassed within the User Cluster include:

- *The VU:* The trusted virtual counterpart of the user, integrating them into the digital world by encompassing the HDT of the user.
- *SVOs:* Virtualizations of objects endowed with relational capabilities.
- *The User Interface:* Connects the virtualization level with the human user.
- *Applications:* IoT services that the user manages on Lysis, along with their shared back-end.
- *Storage Systems:* Databases for storing persistent data collected by the VU and SVOs.

This cluster configuration not only enhances the security of communications between different components but also facilitates efficient management and integration of various IoT services within the Lysis platform.

4.3.1. Virtual user

The VU concept has been realized as an application within a microservices architecture, compliant with the Lysis platform. It aims to extend the platform's functionalities and enhance the automation level of its overall operation. The VU replaces the user in burdensome operations, maintains an updated user context, and is based on past choices. Novel models are useful as they use analysis of context histories to improve applications and services, to provide proactive management of the interactions with the physical environment or to prevent risks [56,57]. Users can access IoT services and applications in real-time and historical data are exploited at different contextual levels [58].

For initial prototyping and modeling, the focus was on developing the VU's communication functionalities and defining structures for an automated deployment process in the Cloud. A prototype has been developed with the potential for easy extension, allowing the integration of additional modules necessary for the VU's operation. The choice of a container-based distributed microservices architecture ensures scalability, making it suitable for artificial intelligence applications and offering a customized environment with good portability and low overhead [59].

As depicted in Fig. 3, the VU develops vertically in the architecture, communicating with applications and SVOs within its domain. A shared back-end at the application level intelligently makes available the resources collected by the VU, reducing redundant service requests. The user interface, located in the VU domain, connects the virtual world managed by the VU with the real world of the human user.

4.3.2. User interface

The VU needs to interface directly with the human user to ensure the set of operations that characterize its functioning. During the learning process, the VU can directly ask the user for static data or formulate specific questions to study their habits and preferences. Statistical and machine learning methodologies have proved their predictive power in real-world context being able to extract important relationships and information [60]. Modeling context as additional feature similar to users and services is an effective approach to predict and define interactions [61]. Moreover, AI is a promising technology for the profiling of users and devices exploiting historical data [62]. As part of our previous work, a chatbot-based virtual assistant to further enhance user interaction with the IoT platform was developed [16]. This chatbot leverages a serverless architecture and AI tools, and it is integrated with Telegram to enable natural language communication. The chatbot assists users in various tasks, including setting device parameters, requesting data, and managing services, significantly simplifying the interaction process. We developed a simplified web interface designed

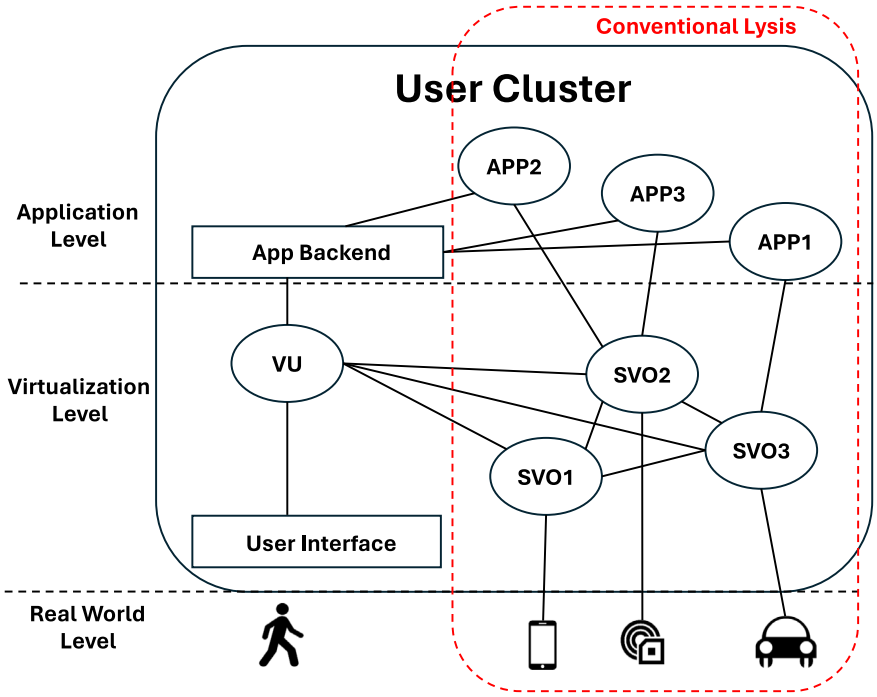


Fig. 3. Microservices involved in the User Cluster.

to perform the basic tasks required for our tests in order to keep a low complexity of the system and to focus on the architecture-related actions. This interface allows users to manage and control SVOs effectively within the Lysis platform. Users can perform essential operations such as deploying SVOs, updating configurations, and monitoring the status of their devices. The web interface ensures that even users with minimal technical expertise can interact with the system efficiently.

During the initialization phase, i.e., when the user wants to add SVOs and/or apps, it needs a dedicated system that can forward requests directly to the VU. Once registered on Lysis, the user is directly redirected to the dedicated web interface, offered and controlled by the VU. From here, each user can monitor their services and profile (see Fig. 4). The major functionality, in the current design, is offered on the page dedicated to adding a new SVO, graphically consistent with the Lysis theme. The QR code, associated with the device to be virtualized, can be easily scanned at this stage, through the camera of smartphones and PCs, or alternatively, by uploading an image containing the two-dimensional barcode. It will no longer be necessary to fill out many other fields except the basic ones, such as the name the user wants to give to the service and possibly its geographic location; everything needed is contained in the QR code. When the user proceeds with the creation, the web page forwards the relevant request to the VU, which starts the initialization of a new SVO in the Cloud. It should be specified that the user interface is a simple and lightweight web application, whose main use is to provide an easily viewable page for the user to monitor their IoT resources, along with the addition of new services. Consequently, it does not use any storage system, but requests the VU, whenever necessary, information related to the SVOs, apps, and the profile to be displayed, or forwards to the same VU the data related to the service for which initialization is requested. To keep the user updated on the outcome of the operations they perform on the platform, WebSocket technology [63] was used, which allows opening a bidirectional interactive communication session between the user's browser and a server. With this API, the web interface can receive messages from the VU's server and display them on the page as a simple notification for the user .

4.3.3. Social virtual object

The SVO plays a crucial role in the proposed architecture by responding to the application layer's requests. The VU communicates directly with active SVOs, maintaining an updated user context, indirectly communicating with physical objects, and leveraging the social network created by the objects for dedicated service definitions. The SVO, as defined in [64], extends the functionalities of the VU to objects.

The structure of the SVO includes:

- *Configuration Handlers:* These enable initialization, registration, and deletion of devices. Initially, these operations are performed by the physical object and the human user. However, with the VU's introduction, it takes over most configuration tasks.
- *Northbound Handlers:* These facilitate resource exchange with the higher level, managing events related to sensor scheduling, trigger addition, sensor management, and actuator control.
- *Social Enabler (SE) (Fig. 5):* This module implements the social functionalities of the object, managing owner and parental relationships, and enabling geographical and encounter-based relationships. The SE publishes its URL via MQTT for visibility and interacts with the Cache memory and Database for social sharing between virtualized objects.

To integrate this prototype into the new architecture, functionalities for SVO management operated by the VU have been added. This includes initial logic for setting controls and sending commands by the VU. Context-aware computing is a new approach to allow understanding context, environment, or status using data from devices to react accordingly in autonomous ways, requiring both sensing and increasingly learning, as IoT systems get more data and better learning from this big data [65]. Context-aware computing connects a variety of information found in the real world turning real situations into information, and providing human-oriented decision making through the application of a variety of machine learning techniques, such as feature extraction, learning, and inference [66]. The VU performs intelligent context analysis through data collection, extending pre-existing functionalities with a VU-SVO interface.

In the configuration phase, the VU interacts with the social object through APIs previously used between SVO and RWO. The initialization

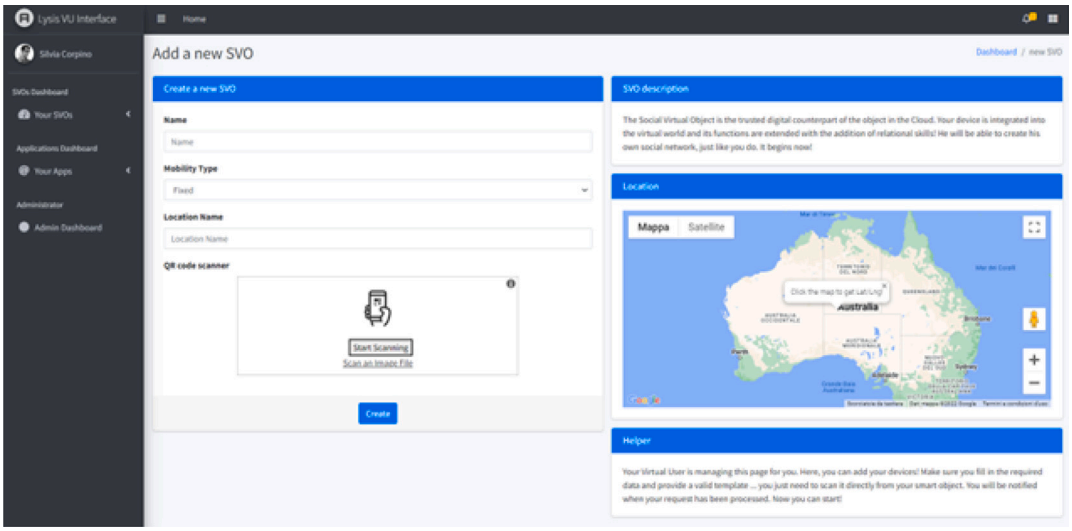


Fig. 4. Registration of a new SVO on the VU's web user interface.

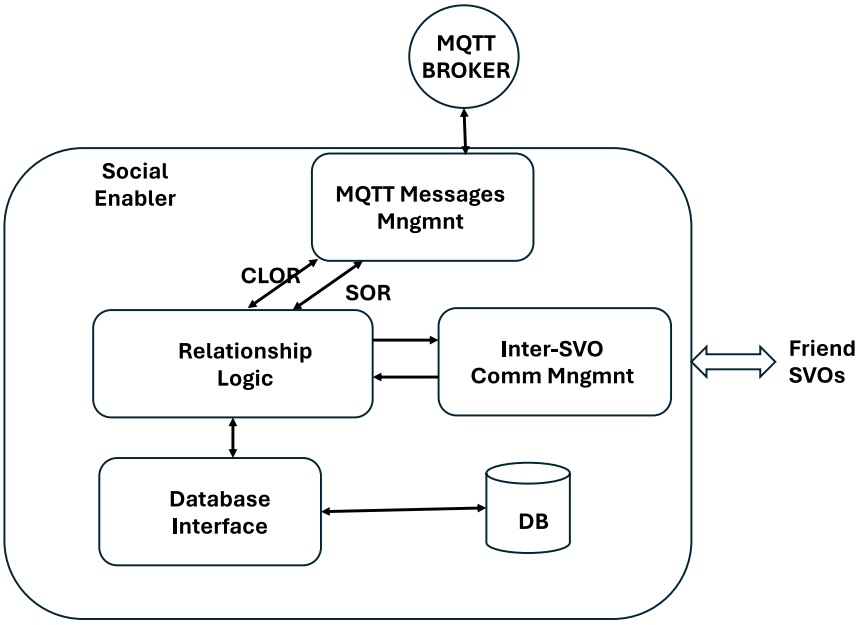


Fig. 5. Social Enabler.

and deletion of a new SVO are now operated by the VU, while registration, though automated, remains the task of the physical object. A listener has been implemented to receive responses from SVOs based on controls set by the VU or related commands.

These functionalities, combined with the Social Enabler's ability to build a social graph, are crucial for continuously studying and updating the user context. A first logic for recognizing devices worn by the user has been implemented through the VU-SVO interface. This includes a control that the VU can set on sensors to recognize their location or state. Any change in state is promptly communicated to the VU, exemplifying the intelligent functioning of the VU in context analysis. These capabilities are essential for efficiently responding at the application level, with future development of these functionalities being of significant interest.

4.3.4. Applications

In the Lysis platform, personal IoT applications of the user can directly access SVOs through Northbound handlers. These handlers enable applications to configure sensor data scheduling or set triggers for specific conditions on the transmitted data, whether numerical or related to the physical object's state. SVOs can also communicate via a dedicated Publish/Subscribe channel for transmitting information based on the set triggers.

The integration of the VU into Lysis introduces an alternative approach for the application layer. As depicted in Fig. 3, the VU provides a shared back-end to the user's IoT applications. This shared environment makes available resources that have been previously requested or recent information collected through continuous analysis of the user context. For instance, the updated list of devices currently worn by the user can be "published" through the shared back-end interface.

This approach allows an application to acquire necessary information for its service without further consulting the VU or the SVOs. Concurrently, the VU can keep applications updated without needing to establish a private connection each time. To manage the back-end securely, a logic based on authorization or the use of access keys is required.

However, this aspect of implementation has not been fully explored in the current research work and is left for future development.

4.4. Implementation technologies overview

The cloud/edge infrastructure employs HTTP and MQTT protocols for robust connection handling and lightweight communication [67–69]. Key technologies include:

- **FastAPI**: A high-performance web framework for developing various components, chosen for its asynchronous support and developer-friendly nature, based on Starlette and Pydantic for asynchronous services and data validation [70–72].
- **Flask**: A Python micro-web framework utilized for developing the SVO, offering functionalities like header parsing and cookie handling [73,74].
- **MongoDB**: A NoSQL database selected for its scalability and flexibility in handling large data volumes, used for managing information of the VU [75,76].
- **SQLite**: Employed for SVOs, providing an ideal solution for individual object data management due to its object-like nature [77].
- **Docker**: Used for deploying the infrastructure, offering efficient server utilization and enhanced security, supported by Docker Hub for image distribution and Docker Compose for multi-container applications [78,79].

4.5. Configuration of a social virtual object

The components and interfaces described in Section 4 facilitate the automated management of configuration processes. This is particularly evident when a user, already registered on the platform, adds a new SVO from their personal console.

The process begins with the user scanning the QR code in the web interface, providing a name for the new SVO, and, if applicable, location information for fixed objects. The QR code content is processed by the User Interface application's back-end and sent, along with other data, to the VU via an HTTP POST request.

The VU then searches its memory for a matching template. Upon finding a match, it tracks the object's information obtained through the interface and sends a deployment request for the service to the Deployer. These temporary data are stored in the VU's memory pending the completion of the deployment process.

The VU initializes the SVO using a configuration script (*Initialize Handler*), providing necessary values such as access keys. The subsequent registration step is executed by the physical device itself through the *Register Handler* script.

In previous Lysis implementations, data required for registration of the device had to be manually entered by the user. In contrast, the proposed application contains predefined information specific to the device type in the corresponding template. Thus, the VU sends most of the registration information directly to the physical device, which registers based on the received package. This communication utilizes the MQTT protocol, with the device listening on a topic contained in the QR code. The topic's encrypted string, decrypted using a key associated with the Lysis template, ensures that the device does not receive unauthorized messages, enhancing the user's security.

4.6. Deployment

Deployment involves the activities of installation, configuration, and enabling that make a software system available for use. Once deployed in the hosting environment, an application becomes an accessible resource at a specific URL. Typically, the host is a virtual machine capable of supporting the deployment process and offering the necessary performance for the proper operation of the deployed service. In this case, particular attention has been given to the deployment procedures of VUs and SVOs, with the process for IoT applications being analogous.

As previously mentioned, the architecture's services, in terms of deployment operations, rely on a specific module: the Deployer. This innovative web application acts as an intermediary between the virtualization layer and the virtual machines where the architecture is distributed. The Deployer exposes an interface for receiving deployment requests from both VUs and the Lysis platform, of which it is a part. It also implements communication with the lower-level deployment environment.

To fully automate the process, another crucial service, the Host Controller (HC), was introduced. While the Deployer handles communications, orchestrating the reception of new requests and the sending and manipulation of results, the HC executes the actual configuration and execution of Docker containers on the machine. For the HC to process requests and recognize the Deployer as a reliable source, it authenticates using a Bearer Token, a type of Access Token used for authorization in compliance with the OAuth2 standard [80].

The overall operation involves the Deployer collecting requests from the upper level through its interfaces (Fig. 2(a)). It generates a query associated with each request, processes the data for the service to be distributed, and initiates communication with the host level. Once the host level provides the outcome of the distribution process, the Deployer processes the data again, manipulating the database in preparation for forwarding the host level response to Lysis or the VU.

4.6.1. Virtual User deployment

Upon a new user's registration on Lysis, the platform initiates communication with the Deployer's dedicated interface, as shown in Fig. 6. Lysis assigns the user a cloud space for the Docker infrastructure, which is essential for hosting the VU. Each assigned machine must run the Host Controller (HC) application. The Deployer receives details about the Docker images for the VU and its web interface, including version tags, and static user data like personal information and email address.

The Deployer constructs a set of attributes necessary for the HC, such as the request identifier (*query*), user ID, and database access credentials. After authentication and data validation using Pydantic models, the HC starts the deployment process and sends an acknowledgment signal (*"ACK"*) to the Deployer. The user ID serves as a directory name for the Compose file and to identify the network associated with the VU's cluster, ensuring uniqueness.

The HC finds available ports for the services, specified in the YAML file. Although all services in the same cloud space share the same IP, they are available on different ports. The final URL is sent upon deployment completion. If the deployment process is unsuccessful, indicated by a non-zero system response code, the Deployer either maintains or removes the associated data from the database. For successful deployments, the Deployer generates a key for securing communications between the new VU and Lysis, sent to both parties. Additionally, the Deployer creates another key for future communications with the initialized VU.

This deployment process ensures that each VU is uniquely associated with a user and securely integrated into the cloud infrastructure, with the Deployer and HC playing critical roles in orchestrating and executing the deployment.

4.6.2. Social Virtual Object deployment

The configuration process for a new SVO involves several architectural elements, with the Deployer playing a crucial role in ensuring that the SVO is distributed and accessible in the cloud.

To add an SVO to the architecture, the human user scans the associated QR code and creates it through the web interface. This action triggers a request to the VU, which then constructs the necessary information for service deployment, including the Docker image and tag. The data is associated with the template identifier provided during scanning, which the VU has previously stored and kept updated through the Lysis platform interface. If there is no match between the provided

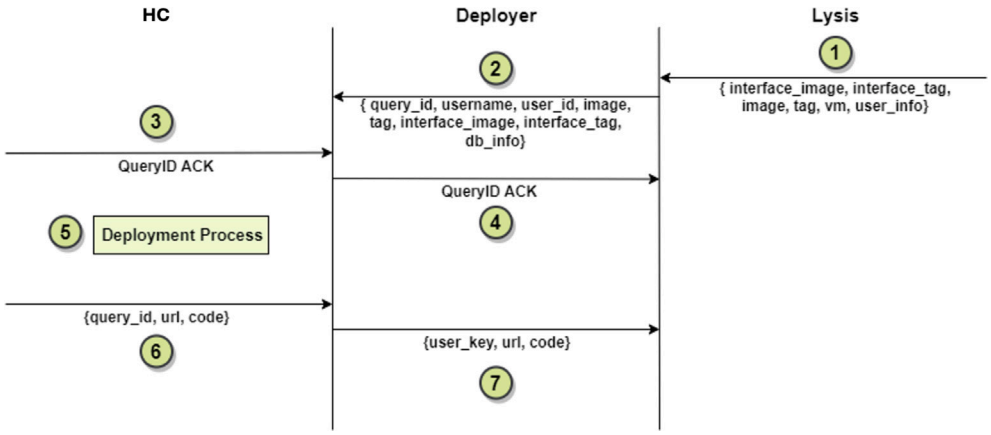


Fig. 6. Deployment process for the Virtual User.

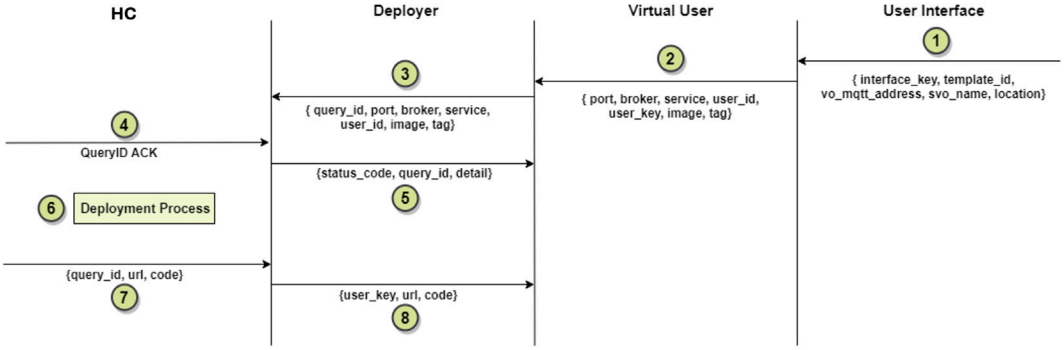


Fig. 7. Deployment process for the Social Virtual Object.

information and the VU's semantic models, the operation is rejected, and the user is alerted.

Fig. 7 illustrates the deployment process for a mobile object, but it is noted that the process is similar for fixed objects, with additional location information provided during creation. The VU supplies the MQTT broker address and the MQTT protocol port number for the SVO to interface with the physical object.

The deployment process then evolves similarly to that of the VU. A key difference is that the directory containing the cluster's basic elements, such as the compose file and volumes, already exists. The YAML operator only needs to add the new content. The correct folder is identified using the user ID, as multiple VUs can be instantiated on the same machine.

After modifying the configuration file, Docker-Compose is invoked via Shell. Existing containers remain unchanged, while new ones are instantiated in the same network. The SVO application and SQLite3 database containers are executed using their respective images. When the HC sends the outcome to the Deployer, it forwards it to the VU via an HTTP POST request.

The VU then handles the configuration operations, communicating directly with both the SVO and the real object.

4.7. Migration of a Social Virtual Object

The capability of the VU to migrate services, especially those of the SVOs, is a crucial feature for its integration into Lysis. This functionality ensures improved performance in terms of latency and connection speed.

As discussed in Section 4.2, a specific interface has been developed to handle requests for migrating an SVO from a cloud environment to an edge environment. As shown in Fig. 8, when the VU identifies the need to migrate a service, it first verifies the availability of spaces in the new area or the addition of new virtual machines by the user. The migration request, including the service URL and the IP address of the new Host Controller (HC), is then forwarded to the HC currently hosting the SVO. Notably, only the IP address is required, as the HC applications are available at the same port on each machine.

This process employs asynchronous communication based on queries, generated by the VU. Upon receiving authorization from the current HC, confirmed through an access key, the HC sends an acknowledgment signal (ACK) and initiates the migration's first phase. This phase involves locating the database file of the requested service within the VU's directory and transmitting the SQLite file content to the recipient.

The new HC, upon receiving the SVO data, starts the second phase of the migration in the background and sends an ACK signal associated with the query to the sender. This phase includes instantiating a new container for the SQLite file and another container for the specific SVO template application, i.e., its Docker image.

If the migration is successful, the initiating HC receives confirmation, deletes the corresponding query and the migrated service data file, and removes the associated containers. The response, containing the new address of the SVO, is transmitted to the requesting VU on the route associated with the query identifying the entire migration process.

4.8. Security and privacy in the proposed architecture

Although security is not the primary focus of this paper, it is essential to address the main security challenges and best practices in microservices-based IoT architectures, especially when integrating social features that handle personal user data [81,82].

Our architecture employs HTTPS and TLS for encrypting communication between various components, including the deployer, Host Controller (HC), Virtual Users (VUs), and Social Virtual Objects (SVOs).

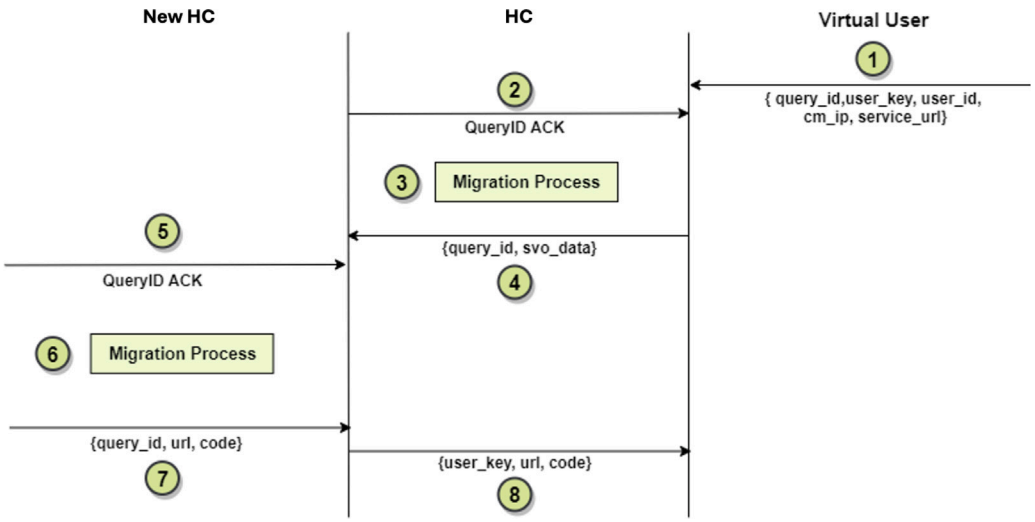


Fig. 8. Migration process of an SVO.

This ensures data confidentiality and integrity during transmission. For authentication, the VU interface uses user passwords, though OAuth 2.0 could also be implemented for enhanced security.

Access to SVOs is controlled through multiple layers of API keys, which are generated by VUs and regularly updated to maintain security. The HC must be added to a whitelist by the deployer to operate within Lysis, and authorization is managed through rotating keys, providing a robust authentication mechanism.

Network insulation is implemented for communications between VUs and their owned SVOs, using a proxy for connections to other clusters or IoT applications. This setup ensures secure and isolated communication channels. Furthermore, MQTT communications are secured with TLS, adding an additional layer of security for data transmitted over this protocol.

Security in IoT environments also involves evaluating the trustworthiness of objects, as misbehaving nodes can exploit social relationships to launch attacks or monopolize services. Nodes with strong social ties can collaborate to dominate specific service classes, impacting the fairness and efficiency of resource distribution. Given that trust evaluation is deeply integrated with IoT service discovery, the concept of Trust Management (TM) is crucial. Research by Nitti et al. [83] has demonstrated the effectiveness of a social approach in evaluating trust within the SIoT paradigm. They proposed a subjective algorithm that SVOs can use to generate a trust-ordered list of resources for upper layers. However, these algorithms often require significant communication between nodes to maintain updated trust values, which can be resource-intensive. In Lysis, the use of a distributed SVO approach and edge/cloud technologies helps to mitigate these challenges. Data ownership is a critical aspect of our architecture. Users can maintain ownership of their data by choosing to store it on their edge devices or in their own cloud space, providing flexibility and control over personal data.

Compliance with relevant standards is a cornerstone of our security strategy [84]. We adhere to the General Data Protection Regulation (GDPR), ensuring that personal data is handled with the utmost care and in accordance with legal requirements. Additionally, we follow the guidelines of ISO/IEC 27001 for information security management, and implement recommended security controls and practices from the National Institute of Standards and Technology (NIST).

By integrating these security measures and best practices, our proposed architecture provides robust protection against potential security threats. This enhances the overall reliability and trustworthiness of the system, making it well-suited for real-world applications where security and privacy are paramount.

5. Experimental evaluation

In the SIoT, the proliferation of connected devices often necessitates continuous updates to user profiles to maintain consistency and personalization of services. This process can be both burdensome and inefficient. Therefore, it is crucial to develop solutions that minimize or eliminate the need for manual profile updates while enhancing the user experience in user-centric applications. The first objective is to analyze the current challenges related to user profile management in SIoT. This includes examining the issues arising from the need for frequent updates due to the vast array of devices and their dynamic interactions. Understanding these challenges is essential for devising effective solutions. Next, the goal is to propose a solution that integrates VU with SIoT. This approach involves defining how an HDT-based solution can automate user profile management. The HDT method creates a digital representation of the user that can dynamically interact with devices and systems, facilitating more autonomous and adaptive management of information. This integration aims to streamline the process of keeping user profiles updated without manual intervention. Finally, it is important to evaluate the specific benefits that the implementation of a VU + SIoT solution would bring to user-centric applications. These benefits include reducing the need for manual updates, increasing automatic personalization, and improving interoperability among devices. By addressing these areas, the combined HDT and SIoT approach can significantly enhance the efficiency and user-friendliness of managing user profiles in a highly connected environment. As outlined in Section 4, the proposed architectural solution integrates the concept of the VU into the Lysis platform. This integration focuses on defining infrastructural properties, developing interfaces, and functional processes such as service deployment, configuration, and migration of SVOs/VUs.

To assess the effectiveness of the overall Cloud infrastructure, it is crucial to create a prototype that simulates the actual functionality intended for real-world application. This prototype should emulate the use of the VU, SVOs, Deployer, and Host Controller within Lysis, as described in previous sections.

To provide a comprehensive analysis, we conducted scenario-based testing that reflects different real-world applications. These scenarios include varying the number of connected devices, the frequency of interactions, and the complexity of tasks performed by the SVOs. The testing scenarios are as follows:

- High Device Density: This scenario simulates an environment with a high number of IoT devices, assessing the system's ability to manage numerous simultaneous connections and data exchanges.

Table 2
Performance-Optimized of Deployer and Hosting Environment Configuration.

	Deployer	Host controller
Instance type	t2.micro	t2.large
Virtual CPU	1 vCPU	2 vCPUs
Memory	1 GB	8 GB
Storage	8 GB EBS root volume	8 GB EBS root volume

- Variable Load Patterns: We tested the system under fluctuating operational loads, mimicking real-world conditions where device interactions and data transmission rates vary throughout the day.
- Complex Interaction Patterns: This scenario evaluates the system's performance with complex interaction patterns, such as those found in smart cities or industrial IoT applications, where devices must frequently communicate and coordinate actions.

The experimental evaluation of the architecture involved a set of metrics to study specific fundamental architectural properties. Quantitative results are obtained through a series of executions SVOs/VUs. These executions stressed the system by simulating various deployment processes with different parallel workloads. Additionally, a functional simulation of the system was conducted, progressively populating it with an increasing number of instances of SVOs.

The experiments primarily focused on the response of CPU, Memory, data traffic, and connectivity, following the overhead produced by running various processes or receiving considerable data traffic input for both the VU and SVO services. While no deployment or simulation was conducted for the applications, the procedure for them is analogous to that for an SVO. Therefore, it is expected that the results for applications would be similar, on average, to those obtained for virtual objects.

5.1. Testbed

The instance simulating the operation of the Deployer and Host Controller were configured with the specifications summarized in Table 2. For the experimental evaluation, two EC2 (Elastic Compute Cloud) instances provided by Amazon Web Service [85] were utilized. AWS offers a range of processors, storage systems, networks, and operating systems to suit the specific needs of the infrastructure being implemented.

Each vCPU corresponds to a thread of either an Intel Xeon or AMD EPYC core. An Amazon EBS volume is a durable block-level storage device attachable to instances. Ubuntu was installed as the operating system on both virtual instances.

For the hosting environment where the Host Controller application was executed, a higher-performance instance was selected. This choice was made considering a configuration that demands greater performance capabilities. The table details various aspects such as CPU processing power, memory capacity, storage capabilities, and instance type, all of which are essential for ensuring optimal functioning and efficiency of the Host Controller application. By opting for a higher-performance instance, the hosting environment can better accommodate the application's requirements, thereby enhancing its overall performance and responsiveness.

The following section details the tools used for measurement during the experimental evaluation.

5.2. Tests on the deployment of the virtual user and social virtual object

The first series of experiments focused on executing deployment processes for the VU application and the corresponding User Cluster (UC). Initially, the UC does not contain active containers for SVOs and applications. Each process is initiated by the Host Controller (HC), following a request from the Deployer, which in turn receives it from the Lysis platform. In the experimental setup, JMeter simulates the behavior of Lysis following the registration of a new user.

Two types of time intervals were evaluated:

- **Time A:** the interval between the HC receiving the request and executing the containers for the requested application.
- **Time B:** the interval between the HC receiving the request and sending the response to the Deployer (and consequently, to the requesting module).

Fig. 9 shows the results obtained during various service deployment processes, characterized by progressively increasing workloads for the hosting instance.

The first deployment process for a single VU occurs in an environment with no other running containers or Docker images in the instance's cache memory. The deployment time (Time A) is relatively high due to the need to download Docker images from Docker Hub. The HC constructs the response containing the new service URL and waits seven seconds (an empirically established interval) before transmitting it to ensure all containers are operational. These results represent the consistent difference between Time A and Time B.

Subsequent experiments involved deploying multiple VUs and their clusters:

- three VUs over three seconds (one request per second).
- five VUs over three seconds (one request every six-tenths of a second).
- ten VUs over three seconds (one request every three-tenths of a second).

The average values of the time intervals for these deployments were calculated. The system showed increasing stress with more parallel requests, impacting the deployment times. Graphically, it can be observed that processing periods for three and five VUs are handled quite swiftly and successfully, despite an increase between experiments. Both Time A and Time B values further escalate when deploying ten VUs, corresponding to ten User Clusters, which are progressively requested at intervals of one unit every three seconds. The instance struggles more with deploying a significant number of VUs within a relatively narrow timeframe. It is important to consider that the infrastructure must already support other clusters, meaning resources are partially utilized, resulting in greater challenges in managing parallel processes and corresponding HTTP communication for transmitting responses to the higher level. This translates to longer deployment intervals. However, the average recorded value remains below the one-minute threshold, which is deemed acceptable and rather performing when compared to other deployment strategies.

The experimental evaluation conducted for VUs was similarly applied to SVOs. In these experiments, the JMeter tool simulates the functionality of the user interface module. The results are presented through graphs, accompanied by a brief analysis.

Also, Fig. 9 illustrates the variation in Deployment times (Time A and B) during the deployment processes of an increasing number of SVOs, with requests transmitted within a three-second interval. Similar to the VU deployment, the first SVO deployment on the instance shows higher deployment time due to the requirement for downloading Docker images. Subsequent deployments are faster as fewer containers are instantiated per request.

The CPU and RAM usage during these deployments are shown in Fig. 10 and Table 3, respectively.

The experimental results presented in Fig. 10 provide a comprehensive overview of the performance of our proposed architecture in terms of deployment time and CPU usage for different numbers of SVOs and VUs. The figure illustrates both the average deployment time and CPU usage, with error bars representing the standard deviation. The deployment time data indicates a clear trend where the time required for deployment increases with the number of SVOs and VUs.

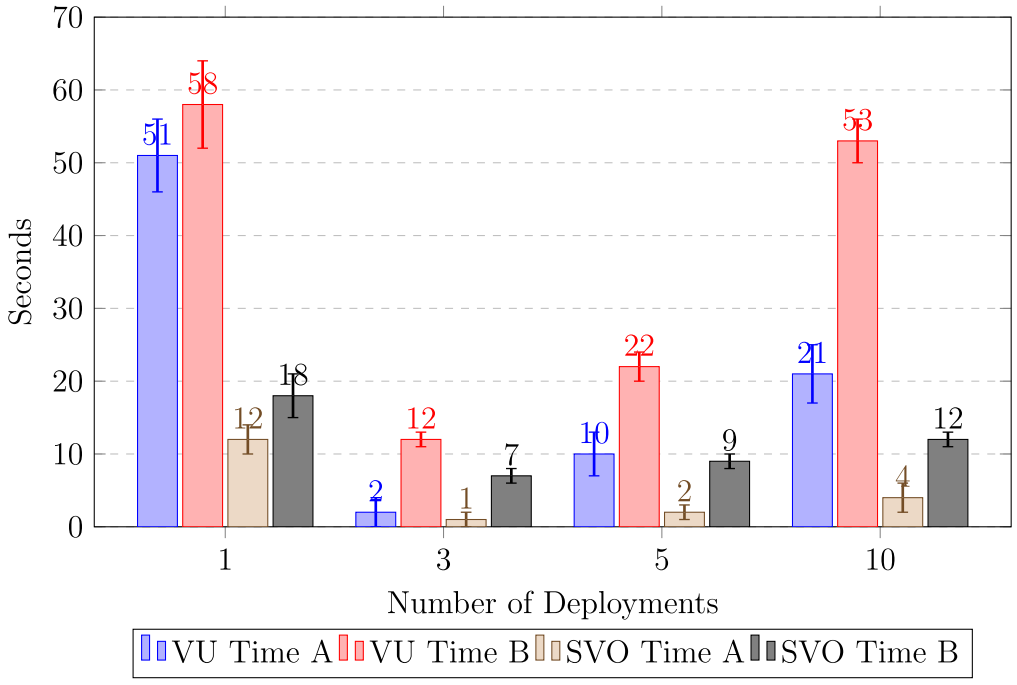


Fig. 9. Comparison of Deployment Times for VU and SVO under Two Different Setups. Error bars represent standard deviation.

Table 3
Average Percentage of RAM in the deployment processes for SVO expressed in seconds.

	RAM % usage
1 SVO	12
3 SVO	14
5 SVO	18
10 SVO	23

For instance, deploying 1 SVO takes an average of 10 s, whereas deploying 10 SVOs takes significantly longer, averaging 40 s. Similarly, deploying 1 VU takes an average of 15 s, while deploying 10 VUs takes 50 s. This increase in deployment time is expected due to the additional resources and coordination required to deploy multiple SVOs and VUs simultaneously. However, the relatively moderate increase in deployment time suggests that the proposed architecture manages scaling efficiently. The CPU usage data shows a similar trend, with CPU load increasing as the number of SVOs and VUs grows. The CPU usage peaks at 90% for the deployment of 10 SVOs and 95% for the deployment of 10 VUs, compared to 20% for 1 SVO and 25% for 1 VU. This substantial increase highlights the higher computational demand required to handle more SVOs and VUs. The data also indicates that the CPU usage stabilizes relatively quickly after the peak, demonstrating the system's ability to handle peak loads and return to normal operating conditions efficiently. The relatively low standard deviation values indicate consistent performance across multiple trials, reinforcing the robustness of the proposed architecture.

The system handles the distribution of ten SVOs with less stress compared to the VU, indicating that fewer resources are necessary for SVO deployment. However, in real-world applications with multiple clusters, higher computational capabilities might be required.

Fig. 11 presents the HTTP performance metrics, specifically focusing on latency and connection time, for up to 19 concurrent HTTP POST requests generated during the test. The connection time remains practically constant throughout the test. The minimum connection time observed is 120 ms, while the maximum is 134 ms. This stability indicates a consistent and reliable network connection, irrespective of the increasing number of requests or the type of communication, whether VU or SVO.

Latency tends to increase with the number of transmitted requests. For VU communications, the latency peaks significantly, with the last two requests recording high values of approximately 36,069 ms and 35,938 ms. Similarly, for SVO communications, latency also increases, displaying notable peaks under high load conditions. These high latency values were obtained by disabling HTTP timeouts in the FastAPI applications involved. Without this adjustment, connections would have dropped, and while the processes would have completed, the results would not have been correctly transmitted. Therefore, these high latency values are not desirable in practical applications.

The tests were conducted under stress conditions, with HTTP timeouts disabled to capture the maximum latency values, which would otherwise result in dropped connections. These high latency values are not typical and represent extreme stress scenarios, indicating the system's performance limits. The testbed used had limited performance capabilities, with the Deployer's virtual RAM size being only 1 GB. Additionally, the geographical distance between the local machine in Italy and the EC2 instances in Northern Virginia contributed to the latency, as the physical separation added to the delay.

For SVO communications, the observed behavior is similar to that of VU, albeit more pronounced. This is due to the request originating from the user interface, passing through the HC, VU, and Deployer, and then reversing the path. As the number of requests transmitted increases within the same time frame (three seconds), the communication latency for transmitting a request to the HC and receiving a QueryACK becomes progressively higher. In the SVO experiments, JMeter simulates the connections received and established by the user interface. Users request the addition of a new SVO service through the interface, which is translated into a deployment request by the Virtual User. This process generates nested HTTP POST requests from the interface to the host, introducing delays in the connection path. Additionally, the Deployer, given the technical specifications of the chosen instance, is not suitable for handling overload conditions caused by receiving numerous parallel requests. The HC also takes longer each time it needs to add a background process to those already running.

To mitigate high latency and avoid HTTP connection drops, several approaches can be considered. Optimizing resource allocation

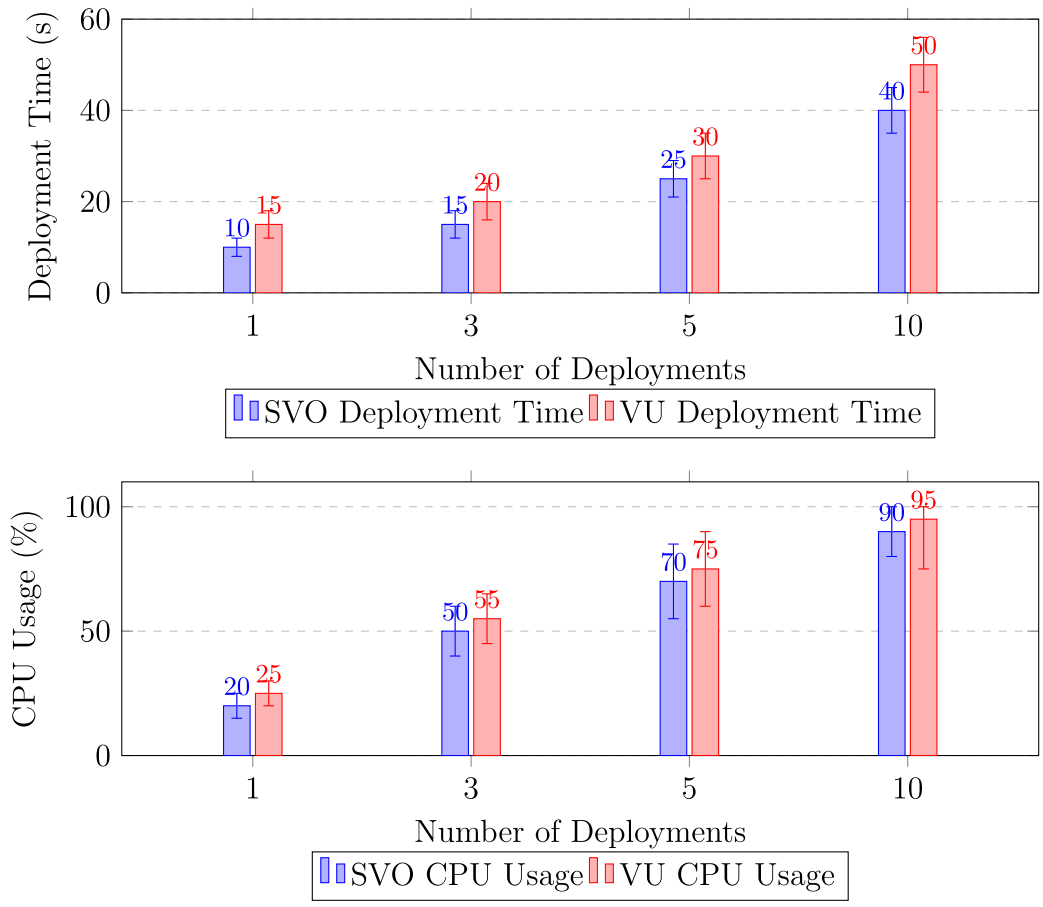


Fig. 10. Average Deployment Time and CPU Usage for Different Numbers of SVOs and VUs. Error bars represent standard deviation.

by defining an appropriate logic for migration processes can ensure resources are as close as possible to the client or platform, reducing cloud distances. Using instances or virtual machines with better performance capabilities can handle higher workloads and manage numerous parallel requests more efficiently.

Overall, the results indicate that while the system maintains stable network connectivity, there are areas for improvement in optimizing server resources to reduce latency spikes. Implementing load balancing techniques, dynamic scaling, and continuous monitoring can help address these issues, ensuring better performance under stress conditions.

5.3. Cloud infrastructure in operational conditions

The development of the Cloud infrastructure involved defining functional modules and interfaces, which, during operation, facilitate a dense exchange of information and execute critical processes such as deployment. This section focuses on analyzing the Cloud system's infrastructural characteristics under operational conditions, particularly CPU usage and data traffic.

A Python demo application was developed to simulate the functioning of real objects associated with SVOs in User Clusters. This application mimics the registration process and continuous data transmission by the objects, including numerical data and mac-addresses, akin to a Raspberry device. The SVOs schedule scans of wireless interfaces to identify their location or find other SVOs for establishing relationships. This data is forwarded to the owner VU for context analysis.

The following scenarios were simulated to produce significant incoming traffic on the virtual instance:

- One VU and one SVO in a single User Cluster.
- One VU and three SVOs in a single User Cluster.
- One VU and five SVOs in a single User Cluster.
- Two VUs and two SVOs in two User Clusters (one SVO per user).
- Two VUs and six SVOs in two User Clusters (three SVOs per user).
- Two VUs and ten SVOs in two User Clusters (five SVOs per user).

5.3.1. Data traffic analysis

Fig. 12 presents the incoming network traffic generated during the functional simulations.

The simulations reveal that even a single User Cluster can generate considerable traffic. In real-world applications, this traffic is expected to increase significantly as the VU processes information and interacts with other modules, leading to an increase in HTTP requests and overall traffic. Considering the operation of a single VU managing a single SVO, a throughput value of approximately 5 kbps is already achieved. This value tends to increase almost linearly when two additional services of the same type are added to the same User Cluster simultaneously. It is evident that even the services related to a single User Cluster alone can generate significant traffic even when simulating simple data transmission. In an actual real-world application, not only does the VU receive information of interest, but it also processes decisions based on them and transmits commands and specifications to the functional modules within its domain, which in turn transmit the corresponding responses through additional HTTP requests that contribute to the overall traffic. Communication is anticipated to experience substantial growth. This prediction finds support in the examination of data traffic in scenarios marked by the creation and subsequent functioning of two User Clusters, each linked to a growing array of services. In typical operational settings, the average incoming data bit rate easily

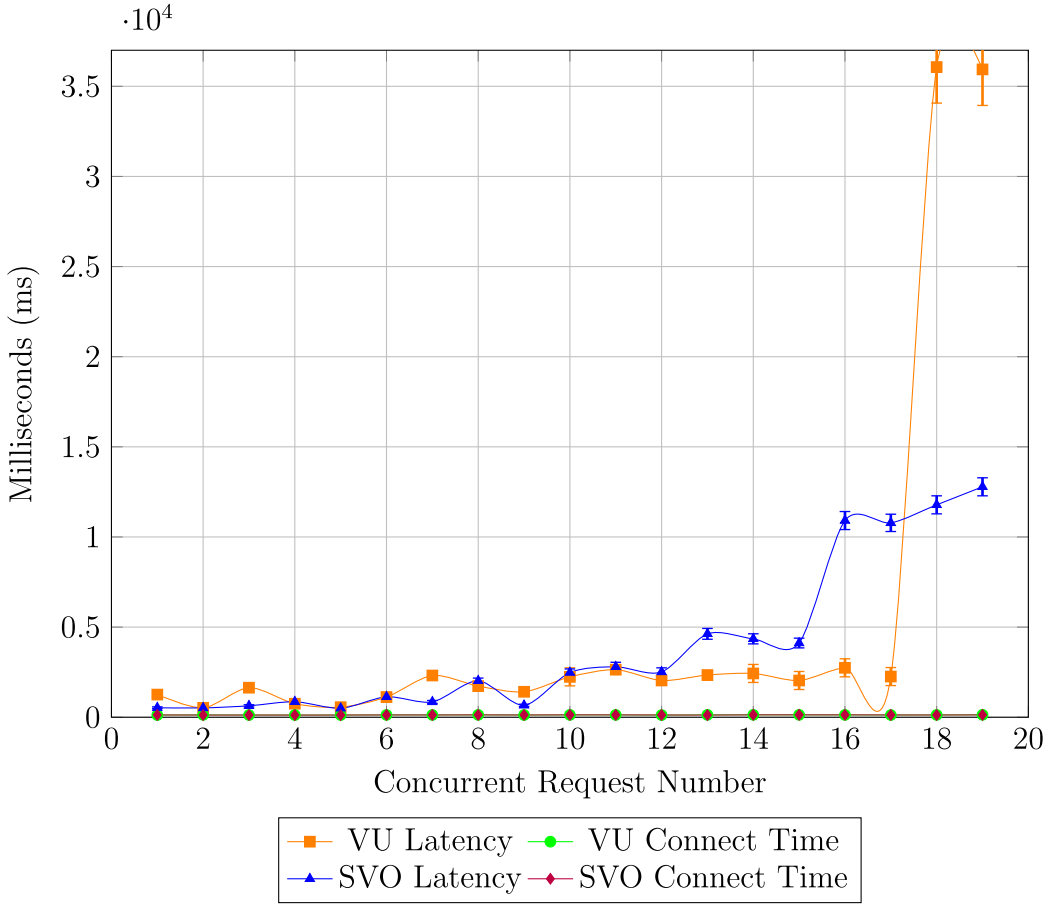


Fig. 11. HTTP Performance Metrics: Latency and Connect Time for Each HTTP POST Request for VU and SVO. Error bars represent standard deviation for latency. Testing was performed under stress conditions with a limited performance testbed.

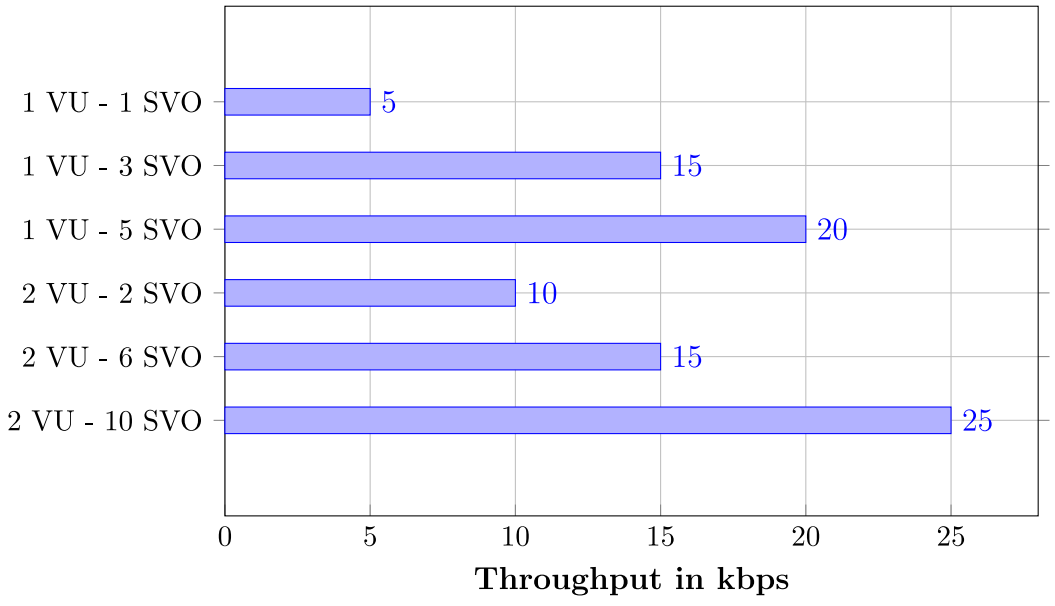


Fig. 12. Throughput for different configurations of Virtual Users (VU) and Social Virtual Objects (SVO) measured in kbps.

surpasses 25 kbps. In this regard, it is interesting to assess how the instance responds to the computational complexity required as a result of managing the modules during operation and due to the generation of data traffic on the network associated with the operation of these services.

5.3.2. CPU usage analysis

Fig. 13 shows the CPU usage during the operation of the Cloud infrastructure under different simulated scenarios.

The data is illustrated as vertical bars representing the average CPU usage percentage, with error bars indicating standard deviation

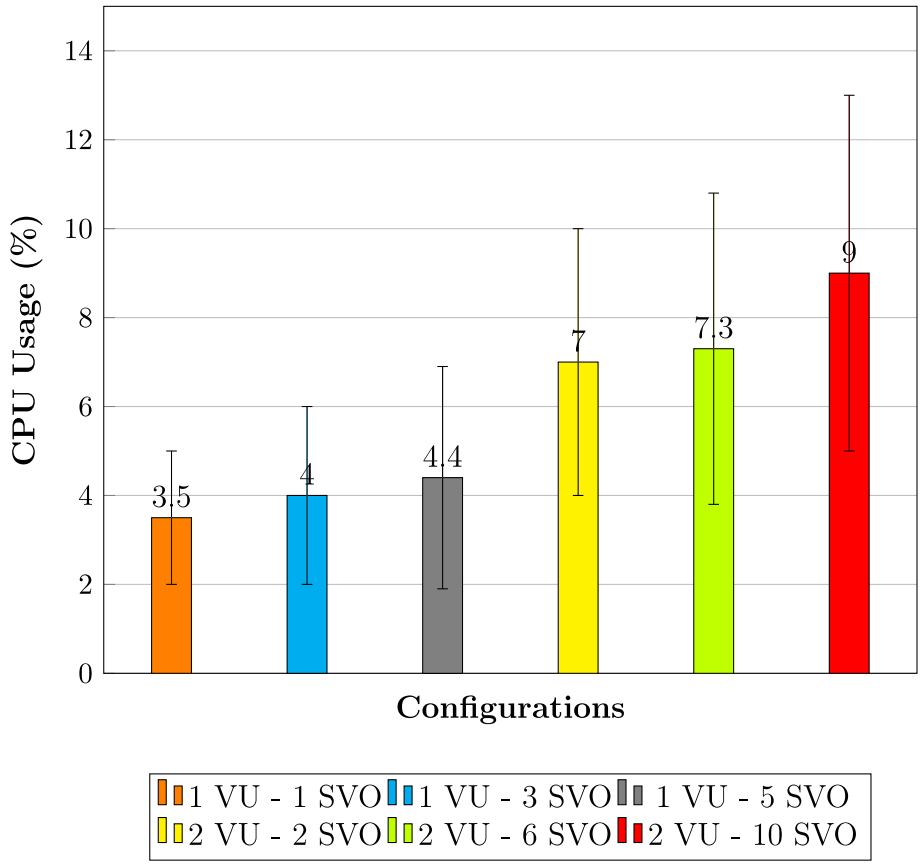


Fig. 13. CPU usage for different configurations of Virtual Users (VU) and Social Virtual Objects (SVO). Error bars represent typical standard deviation in IoT scenarios.

in typical IoT scenarios. The configurations range from minimal interaction (1 VU - 1 SVO) to more complex setups (2 VU - 10 SVO), providing a comprehensive view of the system's performance under varying loads. The simplest configuration (1 VU - 1 SVO) has the lowest CPU usage, as expected, due to the minimal processing load. As the number of SVOs increases for a single VU, there is a moderate rise in CPU usage, demonstrating the incremental processing requirements. When the number of VUs is increased, even with a relatively smaller number of SVOs, the CPU usage jumps significantly. This suggests that the system's overhead grows considerably with additional user management tasks. The highest CPU usage is observed in scenarios with multiple VUs and a larger number of SVOs, highlighting the system's need to balance multiple concurrent interactions.

These observations provide valuable insights into the scalability and efficiency of the proposed architecture. The system maintains reasonable CPU usage across various configurations, indicating robust performance and efficient resource utilization. Error bars reflecting the standard deviation in typical IoT scenarios illustrate that while there is variability, the system consistently manages CPU resources effectively.

5.4. Comparison with Google App Engine

The Cloud infrastructure developed in this research, as experimentally evaluated, presents significant differences from previous Lysis platform integrations, particularly those using Google App Engine (GAE) [86]. While GAE was a convenient solution in the initial phase, it proved limiting for large-scale distributed environments. In contrast, the containerization-based virtualization adopted in this research aligns better with the strategic requirements and allows for easy scalability, crucial for ongoing VU research. The original version of Lysis GAE to run the SVOs. The deployment process involves the deployer connecting to GAE through SSH and executing a series of commands: creating a new project, uploading the source code, and running the service. This traditional approach necessitates uploading the entire source code each time an SVO is created or updated.

Table 4
Comparison of average deployment time (in seconds).

	GAE	Docker (Best case)	Docker (Worst case)
Virtual User	179 s	2 s	58 s
Social Virtual Object	168 s	1 s	18 s

In contrast, the proposed model optimizes this process significantly. The HC downloads the source code only once. For the creation of similar SVOs, re-uploading the entire source code is unnecessary. The layered structure of containers further enhances this efficiency by requiring only the different layers of images to be downloaded.

For the reference setup of GAE, we used the GAE standard environment, configured with automatic scaling, warmup calls, shared memcache, and Firestore.

A critical aspect of performance comparison is the average deployment time of applications. Table 4 presents a comparison of average deployment times for VU and SVO applications on GAE and the proposed Docker-based infrastructure.

The data in Table 4 were obtained by distributing code packages for VU and SVO applications on GAE and Docker. The Docker-based infrastructure, even in its worst-case scenario, significantly outperforms GAE, with deployment times well under one minute for both VU and SVO services.

One of the key improvements in the Docker-based infrastructure is the automation of service deployment, a feature lacking in the GAE integration. GAE does not support parallel deployment processes, requiring manual file uploads for each service deployment. This limitation always places GAE deployments in a "worst-case" scenario.

Table 5
Summary of Scalability Experimental Results.

Scenario	Metric	Result
High Device Density	Response Time	Stable (250 ms)
	System Overhead	Low
Variable Load Patterns	Performance Stability	High
	Resource Utilization	Optimized
Complex Interaction Patterns	Coordination Efficiency	Seamless
	Scalability	High [8,14]

In contrast, the Docker-based infrastructure, with the integration of the Host Controller and the new Deployer prototype, automates service distribution in response to user requests. This automation, combined with the use of Docker containers, significantly enhances the operational efficiency of the Lysis platform.

5.5. Scalability analysis

Scalability is a critical aspect of the SIoT, ensuring that the system can handle a large number of devices and maintain performance under varying operational loads and interaction patterns. The SIoT guarantees scalability through its underlying social network structure, where each SVO operates independently and can be deployed in various locations across the cloud or edge environments. This decentralization results in a peer-to-peer (P2P) social network that inherently supports scalable operations.

The proposed model supports vertical scalability by enhancing the capabilities of host machines and horizontal scalability by increasing the number of replicas across different hosts. Vertical scalability is achieved by upgrading the resources (CPU, memory, etc.) of the host machines, allowing them to handle more significant loads. Horizontal scalability involves adding more instances of hosts, distributing the load, and ensuring that no single point becomes a bottleneck.

The experimental results demonstrate the system's scalability. Under high device density, the system maintained consistent performance, with negligible increases in response time and system overhead. In scenarios with variable load patterns, the system dynamically adjusted resources, ensuring stable performance without significant delays. Complex interaction patterns were handled efficiently, with the decentralized nature of the SIoT allowing for seamless coordination between SVOs (see Table 5).

The results support our assertion that the SIoT architecture provides robust scalability, capable of handling diverse and demanding IoT environments. By leveraging the social network structure and the flexible deployment options of cloud and edge computing, the system can scale both vertically and horizontally, accommodating increasing numbers of devices and varying interaction complexities.

6. Conclusion and future works

As the Internet of Things (IoT) grows, including millions of devices into a globally linked network, maintaining this huge and varied ecosystem becomes increasingly difficult. The notion of the Social Internet of Things (SIoT) has arisen as a viable solution to this problem, incorporating social dynamics into IoT to improve interoperability and user participation. Within this environment, our study aimed to address the crucial need for a scalable, user-centric architecture capable of meeting the SIoT's growing needs.

Our suggested architecture is a huge step forward in this sector, combining Virtual Users (VUs) and Social Virtual Objects (SVOs) with a powerful Cloud infrastructure that is designed for scalability, security, and ease of use. We created a system that allows for quick, secure communication across a wide range of IoT installations, including those at the network's edge, using containerization and a modular architecture. The results of our experimental studies demonstrate the usefulness of our strategy, with significant gains in deployment times and operating efficiency when compared to standard platforms such as Google App Engine.

The results of our findings are significant, providing a new structure for SIoT deployments that prioritize user engagement and system scalability. The automation of IoT service administration via VUs, in particular, stands out as a crucial breakthrough that reduces complexity for end users while also democratizing access to IoT technology. This user-centric approach improves the accessibility of IoT systems while also paving the way for more customized, adaptive services that can learn and change in response to user behaviors and preferences.

Experiments were conducted considering different real-world application scenarios which included variable number of connected devices, frequency of interactions, and complexity of tasks performed by the SVOs. The results indicated as deployment time and CPU usage increase with the numbers of SVOs and VUs. This increase is expected due to the additional resources and coordination required to deploy multiple SVOs and VUs simultaneously. However, the relatively moderate increase in deployment time suggests that the proposed architecture manages scaling efficiently. The CPU usage data highlights the higher computational demand required to handle more SVOs and VUs. The relatively low standard deviation values indicate consistent performance across multiple trials, reinforcing the robustness of the proposed architecture.

Latency and connection time for up to 19 concurrent HTTP POST requests generated during the test showed how the connection time remains practically constant throughout the entire duration indicating a consistent and reliable network connection, irrespective of the increasing number of requests or the type of communication, whether VU or SVO. Instead, latency increases with the number of transmitted requests, suggesting that while the system maintains stable network connectivity, there are areas for improvement in optimizing server resources to reduce latency spikes. Implementing load balancing techniques, dynamic scaling, and continuous monitoring can help address these issues, ensuring better performance under stress conditions.

Finally, experimental results demonstrate as under high device density, the system maintained consistent performance, with negligible increases in response time and system overhead. In scenarios with variable load patterns, the system dynamically adjusted resources, ensuring stable performance without significant delays.

Notwithstanding these encouraging findings, there are some limits to our research. Our controlled experimental setting might not adequately capture the complexities and unpredictabilities inherent in real-world deployments. Therefore, to confirm the architecture's efficacy in a variety of real-world contexts, further study needs go beyond laboratory settings. Novel context-aware computing methods will be considered to improve learning collected big amount of data together with the use of machine learning techniques, with the main objective of reacting in the most effective way to the context and environment in which users and devices are immersed. Developing distributed user learning models could facilitate automated control of SVO rules, enhancing the efficiency and responsiveness of SIoT systems by allowing dynamic adaptation to user preferences. Additionally, creating heuristics for controlling access to sensitive information within the SIoT framework can lead to more secure systems that protect user privacy while enabling seamless interactions between devices.

Investigating the learning of complex user profile data structures offers the potential to improve user integration into the complex world of SIoT and IoT in general, resulting in more personalized and context-aware IoT applications. Furthermore, developing migration heuristics for managing the continuum between edge and cloud computing can optimize resource allocation, enhancing the performance and scalability of SIoT systems. Exploring advanced user interfaces with the integration of Large Language Models (LLMs) can provide more intuitive and efficient ways for users to interact with their SIoT devices, improving overall user experience.

CRediT authorship contribution statement

Roberto Girau: Writing – review & editing, Writing – original draft, Supervision, Project administration. **Matteo Anedda:** Writing – review & editing, Writing – original draft, Validation. **Roberta Presta:** Writing – review & editing, Methodology. **Silvia Corpino:** Software, Data curation, Conceptualization. **Pietro Ruiu:** Writing – review & editing, Validation. **Mauro Fadda:** Writing – review & editing, Funding acquisition. **Chan-Tong Lam:** Writing – review & editing, Funding acquisition. **Daniele Giusto:** Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] L. Atzori, A. Iera, G. Morabito, The Internet of Things: A survey, Comput. Netw. 54 (15) (2010) 2787–2805, http://dx.doi.org/10.1016/j.comnet.2010.05.010, URL https://www.sciencedirect.com/science/article/pii/S1389128610001568.

[2] J. Ramakrishnan, M.S. Shabbir, N.M. Kassim, P.T. Nguyen, D. Mavaluru, A comprehensive and systematic review of the network virtualization techniques in the IoT, Int. J. Commun. Syst. 33 (7) (2020) e4331, http://dx.doi.org/10.1002/dac.4331, e4331 IJCS-17-0853.R1, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/dac.4331 URL https://onlinelibrary.wiley.com/doi/abs/10.1002/dac.4331.

[3] V. Beltran, A.M. Ortiz, D. Hussein, N. Crespi, A semantic service creation platform for social IoT, in: 2014 IEEE World Forum on Internet of Things, WF-IoT, 2014, pp. 283–286, http://dx.doi.org/10.1109/WF-IoT.2014.6803173.

[4] R. Girau, R. Cossu, M. Farina, V. Pilloni, L. Atzori, Virtual user in the IoT: Definition, technologies and experiments, Sensors 19 (20) (2019) http://dx.doi.org/10.3390/s19204489, URL https://www.mdpi.com/1424-8220/19/20/4489.

[5] M. Nitti, V. Pilloni, G. Colistra, L. Atzori, The virtual object as a major element of the Internet of Things: A survey, IEEE Commun. Surv. Tutor. 18 (2015) 1, http://dx.doi.org/10.1109/COMST.2015.2498304.

[6] L. Atzori, D. Carboni, A. Iera, Smart things in the social loop: Paradigms, technologies, and potentials, Ad Hoc Netw. 18 (2014) 121–132.

[7] D. Guinard, V. Trifa, T. Pham, O. Liechti, Towards physical mashups in the web of things, in: Proceedings of INSS'09, 2009.

[8] L. Atzori, A. Iera, G. Morabito, M. Nitti, The social Internet of Things (SIoT) – When social networks meet the Internet of Things: Concept, architecture and network characterization, Comput. Netw. 56 (2012) http://dx.doi.org/10.1016/j.comnet.2012.07.010.

[9] L. Atzori, A. Iera, G. Morabito, SIoT: Giving a social structure to the Internet of Things, IEEE Commun. Lett. 15 (11) (2011) 1193–1195, http://dx.doi.org/10.1109/LCOMM.2011.090911.111340.

[10] S. Ali, M.G. Kibria, M.A. Jarwar, H.K. Lee, I. Chong, A model of socially connected web objects for IoT applications, Wirel. Commun. Mob. Comput. 2018 (2018) 6309509, http://dx.doi.org/10.1155/2018/6309509.

[11] J.E. Kim, X. Fan, D. Mosse, Empowering end users for social Internet of Things, in: IoTDI '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 71–82, http://dx.doi.org/10.1145/3054977.3054987.

[12] K.M. Alam, M. Saini, A. El Saddik, tNote: A social network of vehicles under Internet of Things, in: R.C.-H. Hsu, S. Wang (Eds.), Internet of Vehicles – Technologies and Services, Springer International Publishing, Cham, 2014, pp. 227–236.

[13] G. Ruggeri, O. Briante, A framework for IoT and E-health systems integration based on the social Internet of Things paradigm, in: 2017 International Symposium on Wireless Communication Systems, ISWCS, 2017, pp. 426–431, http://dx.doi.org/10.1109/ISWCS.2017.8108152.

[14] R. Girau, S. Martis, L. Atzori, Lysis: A platform for IoT distributed applications over socially connected objects, IEEE Internet Things J. PP (2016) 1, http://dx.doi.org/10.1109/JIOT.2016.2616022.

[15] I. Farris, R. Girau, L. Militano, M. Nitti, L. Atzori, A. Iera, G. Morabito, Social virtual objects in the edge cloud, IEEE Cloud Comput. 2 (6) (2015) 20–28, http://dx.doi.org/10.1109/MCC.2015.116.

[16] R. Cossu, R. Girau, L. Atzori, Lysis chatbot: A virtual assistant for IoT platforms, ITU J. Future Evol. Technol. 2 (2021) 81–91, http://dx.doi.org/10.52953/MCYX4245.

[17] M. Grieves, Digital twin: Manufacturing excellence through virtual factory replication, 2015.

[18] M. Grieves, J. Vickers, Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems, 2017, pp. 85–113, http://dx.doi.org/10.1007/978-3-319-38756-7_4.

[19] R. Minerva, G.M. Lee, N. Crespi, Digital twin in the IoT context: A survey on technical features, scenarios, and architectural models, Proc. IEEE 108 (10) (2020) 1785–1824, http://dx.doi.org/10.1109/JPROC.2020.2998530.

[20] S. Haag, R. Anderl, Digital twin – Proof of concept, Manufact. Lett. 15 (2018) 64–66, http://dx.doi.org/10.1016/j.mfglet.2018.02.006, Industry 4.0 and Smart Manufacturing, URL https://www.sciencedirect.com/science/article/pii/S2213846318300208.

[21] M. Kunath, H. Winkler, Integrating the Digital Twin of the manufacturing system into a decision support system for improving the order management process, Procedia CIRP 72 (2018) http://dx.doi.org/10.1016/j.procir.2018.03.192.

[22] Q. Qi, D. Zhao, T.W. Liao, F. Tao, Modeling of cyber-physical systems and digital twin based on edge computing, fog computing and cloud computing towards smart manufacturing, in: International Manufacturing Science and Engineering Conference, in: Additive Manufacturing; Bio and Sustainable Manufacturing, vol. 1, 2018, http://dx.doi.org/10.1115/MSEC2018-6435, arXiv:https://asmedigitalcollection.asme.org/MSEC/proceedings-pdf/MSEC2018/51357/V001T05A018/2520134/v001t05a018-msec2018-6435.pdf V001T05A018.

[23] R. Casadei, M. Viroli, Collective abstractions and platforms for large-scale self-adaptive IoT, in: 2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems, FAS*W, 2018, pp. 106–111, http://dx.doi.org/10.1109/FAS-W.2018.00033.

[24] I. Bedhief, L. Foschini, P. Bellavista, M. Kassar, T. Aguili, Toward self-adaptive software defined fog networking architecture for IIoT and industry 4.0, in: 2019 IEEE 24th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks, CAMAD, 2019, pp. 1–5, http://dx.doi.org/10.1109/CAMAD.2019.8858499.

[25] E. Negri, L. Fumagalli, M. Macchi, A review of the roles of digital twin in CPS-based production systems, Procedia Manuf. 11 (2017) 939–948, http://dx.doi.org/10.1016/j.promfg.2017.07.198, 27th International Conference on Flexible Automation and Intelligent Manufacturing, FAIM2017, 27-30 June 2017, Modena, Italy, URL https://www.sciencedirect.com/science/article/pii/S2351978917304067.

[26] G.A. Carella, M. Pauls, T. Magedanz, M. Cilloni, P. Bellavista, L. Foschini, Prototyping nfv-based multi-access edge computing in 5G ready networks with open baton, in: 2017 IEEE Conference on Network Softwarization, NetSoft, 2017, pp. 1–4, http://dx.doi.org/10.1109/NETSOFT.2017.8004237.

[27] S.R. Jeremiah, L.T. Yang, J.H. Park, Digital twin-assisted resource allocation framework based on edge collaboration for vehicular edge computing, Future Gener. Comput. Syst. 150 (2024) 243–254.

[28] Y. Dai, Y. Zhang, Adaptive digital twin for vehicular edge computing and networks, J. Commun. Inf. Netw. 7 (1) (2022) 48–59.

[29] O. El Marai, T. Taleb, J. Song, Roads infrastructure digital twin: A step toward smarter cities realization, IEEE Netw. 35 (2) (2020) 136–143.

[30] K. Zhang, J. Cao, Y. Zhang, Adaptive digital twin and multiagent deep reinforcement learning for vehicular edge computing and networks, IEEE Trans. Ind. Inform. 18 (2) (2021) 1405–1413.

[31] W. Sun, H. Zhang, R. Wang, Y. Zhang, Reducing offloading latency for digital twin edge networks in 6G, IEEE Trans. Veh. Technol. 69 (10) (2020) 12240–12251.

[32] Y. Wang, J. Fang, Y. Cheng, H. She, Y. Guo, G. Zheng, Cooperative end-edge-cloud computing and resource allocation for digital twin enabled 6G industrial IoT, IEEE J. Sel. Top. Sign. Proces. (2023).

[33] I. Graessler, A. Pöhler, Integration of a digital twin as human representation in a scheduling procedure of a cyber-physical production system, in: 2017 IEEE International Conference on Industrial Engineering and Engineering Management, IEEM, IEEE, 2017, pp. 289–293.

[34] S.D. Okegbile, J. Cai, C. Yi, D. Niyato, Human digital twin for personalized healthcare: Vision, architecture and future directions, IEEE Netw. (2022) 1–7, http://dx.doi.org/10.1109/MNET.118.2200071.

[35] Z. Cheng, Human digital twin with applications, in: Proceedings of the 7th International Digital Human Modeling Symposium (DHM 2022) and Iowa Virtual Human Summit 2022 -, 2022, URL https://api.semanticscholar.org/CorpusID:251870156.

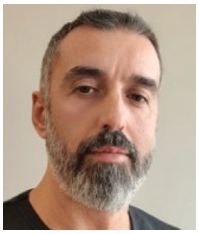
[36] I. Graessler, A. Poehler, Integration of a digital twin as human representation in a scheduling procedure of a cyber-physical production system, in: 2017 IEEE International Conference on Industrial Engineering and Engineering Management, IEEM, 2017, pp. 289–293, http://dx.doi.org/10.1109/IEEM.2017.8289898.

[37] C. Wang, Z. Cai, Y. Li, Human activity recognition in mobile edge computing: A low-cost and high-fidelity digital twin approach with deep reinforcement learning, IEEE Trans. Consum. Electron. (2024) 1, http://dx.doi.org/10.1109/TCE.2024.3375859.

[38] M. Younis, Internet of everything and everybody: Architecture and service virtualization, Comput. Commun. 131 (2018) 66–72.

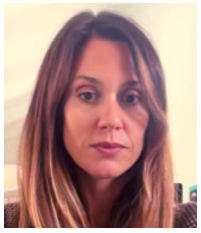
[39] I. Ullah, M. Sohail Khan, D. Kim, et al., IoT services and virtual objects management in hyperconnected things network, Mob. Inf. Syst. 2018 (2018).

[40] L. Atzori, J.L. Bellido, R. Bolla, G. Genovese, A. Iera, A. Jara, C. Lombardo, G. Morabito, Sdn&nfv contribution to IoT objects virtualization, Comput. Netw. 149 (2019) 200–212.
[41] Z.U. Shamszaman, M.I. Ali, Toward a smart society through semantic virtual-object enabled real-time management framework in the social Internet of Things, IEEE Internet Things J. 5 (4) (2017) 2572–2579.
[42] W. Shi, S. Dustdar, The promise of edge computing, Computer 49 (5) (2016) 78–81, http://dx.doi.org/10.1109/MC.2016.145.
[43] M. Satyanarayanan, The emergence of edge computing, Computer 50 (1) (2017) 30–39, http://dx.doi.org/10.1109/MC.2017.9.
[44] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge computing: Vision and challenges, IEEE Internet Things J. 3 (5) (2016) 637–646, http://dx.doi.org/10.1109/JIOT.2016.2579198.
[45] A. Barbalace, M. Karaoui, W. Wang, T. Xing, P. Olivier, B. Ravindran, Edge computing: The case for heterogeneous-ISA container migration, in: Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, ACM Association for Computing Machinery, 2020, pp. 73–87, http://dx.doi.org/10.1145/3381052.3381321, URL https://conf.researchr.org/home/vee-2020 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE 2020 ; Conference date: 17-03-2020 Through 17-03-2020.
[46] S. Wang, J. Xu, N. Zhang, Y. Liu, A survey on service migration in mobile edge computing, IEEE Access 6 (2018) 23511–23528, http://dx.doi.org/10.1109/ACCESS.2018.2828102.
[47] O. Oleghe, Container placement and migration in edge computing: Concept and scheduling models, IEEE Access 9 (2021) 68028–68043, http://dx.doi.org/10.1109/ACCESS.2021.3077550.
[48] https://cloud.google.com/appengine/docs/java/search/.
[49] R. Morabito, Virtualization on Internet of Things edge devices with container technologies: A performance evaluation, IEEE Access 5 (2017) 8835–8850.
[50] P. Chhikara, R. Tekchandani, N. Kumar, M.S. Obaidat, An efficient container management scheme for resource-constrained intelligent IoT devices, IEEE Internet Things J. 8 (16) (2020) 12597–12609.
[51] L. Liu, Y. Ding, X. Li, H. Wu, L. Xing, A container-driven service architecture to minimize the upgrading requirements of user-side smart meters in distribution grids, IEEE Trans. Ind. Inform. 18 (1) (2021) 719–728.
[52] C. Dupont, R. Giaffreda, L. Capra, Edge computing in IoT context: Horizontal and vertical linux container migration, in: 2017 Global Internet of Things Summit, GIoTS, 2017, pp. 1–4, http://dx.doi.org/10.1109/GIOTS.2017.8016218.
[53] Z. Tang, X. Zhou, F. Zhang, W. Jia, W. Zhao, Migration modeling and learning algorithms for containers in fog computing, IEEE Trans. Serv. Comput. 12 (2019) 712–725.
[54] R. Pérez de Prado, S. García-Galán, J.E. Muñoz-Expósito, A. Marchewka, N. Ruiz-Reyes, Smart containers schedulers for microservices provision in cloud-fog-IoT networks. Challenges and opportunities, Sensors 20 (6) (2020) 1714.
[55] L.M. O., I.A. A., L.H. O., L.B. A., Microservices architecture: a better option over monolithic architecture, in: International Conference of Sciences, Engineering & Environmental Technology, ICONSEET, 2020, pp. 148–156.
[56] R.S. de Souza, J.L. Lopes, C.F.R. Geyer, A. Cardozo, A.C. Yamin, J.L. Barbosa, An architecture for IoT management targeted to context awareness of ubiquitous applications., J. Univers. Comput. Sci. 24 (10) (2018) 1452–1471.
[57] A.S. Filippetto, R. Lima, J.L.V. Barbosa, A risk prediction model for software project management based on similarity analysis of context histories, Inf. Softw. Technol. 131 (2021) 106497.
[58] A. Taherkordi, F. Eliassen, M. Mcdonald, G. Horn, Context-driven and real-time provisioning of data-centric IoT services in the cloud, ACM Trans. Internet Technol. (TOIT) 19 (1) (2018) 1–24.
[59] M. Sun, Y. Guo, hu Zhang, W. Cao, M. Yuan, Performance comparison of multiple containers running artificial intelligence applications, J. Phys. Conf. Ser. 1948 (1) (2021) 012005, http://dx.doi.org/10.1088/1742-6596/1948/1/012005.
[60] S. Polymeni, E. Athanasakis, G. Spanos, K. Votis, D. Tzovaras, IoT-based prediction models in the environmental context: A systematic Literature Review, Internet Things 20 (2022) 100612.
[61] Y. Chen, P. Yu, Z. Zheng, J. Shen, M. Guo, Modeling feature interactions for context-aware QoS prediction of IoT services, Future Gener. Comput. Syst. 137 (2022) 173–185.
[62] N. Koursioumpas, S. Barmpounakis, I. Stavrakakis, N. Alonistioti, AI-driven, context-aware profiling for 5G and beyond networks, IEEE Trans. Netw. Serv. Manag. 19 (2) (2021) 1036–1048.
[63] P. Murley, Z. Ma, J. Mason, M. Bailey, A. Kharraz, WebSocket adoption and the landscape of the real-time web, in: Proceedings of the Web Conference 2021, WWW '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 1192–1203, http://dx.doi.org/10.1145/3442381.3450063.
[64] S. Corpino, S. Mirri, M. Sole, D. Giusto, G. Pau, R. Girau, On implementing socialization algorithms on virtual objects in the social IoT, in: 2022 IEEE 19th Annual Consumer Communications & Networking Conference, CCNC, IEEE Press, 2022, pp. 307–312, http://dx.doi.org/10.1109/CCNC49033.2022.9700724.
[65] O.B. Sezer, E. Dogdu, A.M. Ozbayoglu, Context-aware computing, learning, and big data in Internet of Things: a survey, IEEE Internet Things J. 5 (1) (2017) 1–27.
[66] H.J.T. Manaligod, M.J.S. Diño, S. Ghose, J. Han, Context computing for Internet of Things, J. Ambient Intell. Humaniz. Comput. 11 (2020) 1361–1363.
[67] S. Mohan, K. B. R, A survey on IoT platforms, Int. J. Scient. Res. Mordern Educ. 1 (2016) 468.
[68] J. Mineraud, O. Mazhelis, X. Su, S. Tarkoma, A gap analysis of Internet-of-Things platforms, Comput. Commun. 89–90 (2016) 5–16, http://dx.doi.org/10.1016/j.comcom.2016.03.015.
[69] B. Wukkadada, K. Wankhede, R. Nambiar, A.V. Nair, Comparison with HTTP and MQTT in Internet of Things (IoT), in: 2018 International Conference on Inventive Research in Computing Applications, ICIRCA, 2018, pp. 249–253.
[70] https://fastapi.tiangolo.com/.
[71] https://asgi.readthedocs.io/en/latest/.
[72] https://pydantic-docs.helpmanual.io/.
[73] https://werkzeug.palletsprojects.com/en/2.1.x/.
[74] https://jinja.palletsprojects.com/en/3.1.x/.
[75] https://www.mongodb.com/it-it.
[76] W. Ali, M. Majeed, A. Raza, M.U. Shafique, Comparison between SQL and NoSQL databases and their relationship with big data analytics, Asian J. Comput. Sci. Inf. Technol. 4 (2019) 1–10, http://dx.doi.org/10.9734/AJRCOS/2019/v4i230108.
[77] https://www.sqlite.org/sqlar.html.
[78] https://www.docker.com/.
[79] https://hub.docker.com/.
[80] https://oauth.net/2/.
[81] Z. Ma, J. Zhang, Efficient, traceable and privacy-aware data access control in distributed cloud-based IoD systems, IEEE Access 11 (2023) 45206–45221.
[82] L. Babun, K. Denney, Z.B. Celik, P. McDaniel, A.S. Uluagac, A survey on IoT platforms: Communication, security, and privacy perspectives, Comput. Netw. 192 (2021) 108040.
[83] M. Nitti, R. Girau, L. Atzori, Trustworthiness management in the social Internet of Things, IEEE Trans. Knowl. Data Eng. 26 (5) (2013) 1253–1266.
[84] M. Anedda, A. Floris, R. Girau, M. Fadda, P. Ruiu, M. Farina, A. Bonu, D.D. Giusto, Privacy and security best practices for IoT solutions, IEEE Access 11 (2023) 129156–129172.
[85] https://aws.amazon.com/it/ec2/].
[86] https://cloud.google.com/appengine.



Roberto Girau is assistant professor at University of Bologna, Department of Computer Science and Engineering since 2021. He received the M.S. degree in Telecommunication Engineering and his Ph.D. degree in electronic engineering and computer science from the University of Cagliari, Italy in 2012 and in 2017, respectively. From 2012 to 2020, he worked as researcher at the Department of Electrical and Electronic Engineering of the University of Cagliari, developing an experimental platform for the social Internet of Things. His main research areas of interest are IoT with particular emphasis on its integration with social networks, software engineering, smart cities and cloud computing.

Matteo Anedda (Senior Member, IEEE) is assistant at University of Cagliari, Department of Electrical and Electronic Engineering since 2017. He received a M.Sc. degree (summa cum laude) in Telecommunication Engineering and a Ph.D degree in Electronic and Computer Engineering from the University of Cagliari in 2012 and 2017, respectively. He has been a visiting Erasmus student for eight months with the University of Basque Country, Bilbao, Spain, in 2010, where he has carried out his M.Sc. thesis under supervision of Prof. P. Angueira. He was a visiting researcher with Dublin City University, Ireland, under the supervision of Prof. G. Muntean in 2015 and Universidad de Montevideo, Uruguay, under the supervision of Prof. R. Sotelo in 2016, for seven months each. In 2020 he was short visiting professor at the Department of Electronics and Computers of the Transilvania University of Brasov in Romania. Dr. Anedda's research interests are IoT and Smart Cities, 5G networks and network selection, real-time applications, adaptive multimedia streaming, and heterogeneous radio access environment. Since January 2023, he has been chair of the IEEE BTS Italy Chapter and secretary of the IEEE CTSoc SMC TC.



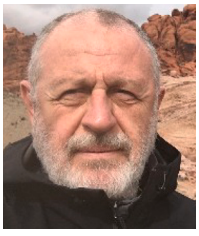
Roberta Presta is assistant professor at University Suor Orsola Benincasa of Naples (Italy), Department of Education, Psychology and Communication. She received a M.Sc. degree in Telecommunications Engineering and a Ph.D. in Information and Automation Engineering both from University Federico II of Naples in 2009 and 2013. She has been working since 2013 as a researcher at the Scienza Nuova interdepartmental research and project design center of University Suor Orsola Benincasa. She received a Ph.D. in Humanities and Technologies from the same university in 2021. Her research is framed in the field of Human Computer Interaction and deals with the design and assessment of interactive systems in different application domains.

Silvia Corpino obtained the bachelor's degree in Electrical, Electronic and Computer Engineering in 2019 and subsequently the master's degree in Electronic Engineering (Embedded Electronics) in 2022 with the thesis "Definition and implementation of the Cloud infrastructure for the integration of the Virtual User in the Social Internet of Things". She has in fact worked as a researcher on virtualization approaches in the Social Internet of Things. Dr. Corpino's interests are IoT , SIoT and Smart Cities. She currently works as a software developer.

Pietro Ruiu obtained the Ph.D. in Electrical, Electronics and Communications Engineering from the Polytechnic of Turin in 2018 and the Master Degree in Telecommunications Engineering from the same University in 2006. He is currently assistant professor (tenure track) at the Biomedical Science Department of the University of Sassari, with main interests on computer vision, artificial intelligence and computing infrastructures. Between 2013 and 2018 he was the head of the Infrastructures and Systems for Advanced Computing (IS4AC) Research Unit at the LINKS Foundation with main interests in heterogeneous infrastructures, cloud automation, data privacy, energy efficiency of computing and network resources. From 2007 to 2013 worked as Researcher at Istituto Superiore Mario Boella (ISMB), in the field of computing infrastructure, studying technologies such as Cloud Computing, Grid Computing, High performance computing (HPC) and virtualization. He was Organizer and TPC member in several international conferences and workshops.

Mauro Fadda is assistant professor in Telecommunications at the Department of Biomedical Sciences of the University of Sassari. Previously, he worked as assistant Professor at the Department of Electrical and Electronic Engineering (DIEE) of the University of Cagliari. He received a Ph.D. in Electronic and Computer Engineering in 2013 from the University of Cagliari. He spent a period of visiting Ph.D. student at the Department of Electronics and Telecommunications, Faculty of Engineering of Bilbao, University of the Basque Country (UPV/EHU) in Spain in 2012. In 2020 he was short visiting professor at the Department of Electronics and Computers of the Transylvanian University of Brasov in Romania. He is a member of the National Interuniversity Consortium for Telecommunications (CNIT) since 2011. From March 2020 he was elected Chair of the Italian chapter of the Broadcast Technology Society (BTS) of the Institute of Electrical and Electronic Engineering (IEEE). Dr. Fadda is co-author of an extensive list of articles published in journals and international conference proceedings. He has served as chair for various international conferences and workshops. He is Associate Editor for IEEE Access magazine and Topic Editor for Sensors magazine. His research currently focuses on telecommunications issues, on cognitive radio systems, on signal processing for radio communications, on the transmission and processing of multimedia data and on state-of-the-art digital systems.

Chan-Tong Lam (Senior Member, IEEE) received the B.Sc. (Eng.) and M.Sc. (Eng.) degrees from Queen's University, Kingston, ON, Canada, in 1998 and 2000, respectively, and the Ph.D. degree from Carleton University, Ottawa, ON, in 2007. He is currently an Associate Professor with the Faculty of Applied Sciences, Macao Polytechnic University, Macau, China. From 2004 to 2007, he participated in the European Wireless World Initiative New Radio (WINNER) Project. He has published more than 100 publications in refereed journals and conferences. His research interests include mobile wireless communications, digital signal processing, machine learning in communications, and computer vision in smart cities. He is a member of the IEEE Communications Society.



Daniele D. Giusto is full professor of telecommunications at the University of Cagliari, Italy, since 2002. He received his Laurea (M.S.) degree in electronic engineering and his Dottorato di Ricerca (Ph.D.) degree in telecommunications from the University of Genoa, Italy, in 1986 and 1990, respectively. Since 1994, he has been a permanent faculty member in the Department of Electrical and Electronic Engineering, University of Cagliari. Dr. Giusto is senior member of IEEE and was a member of IEEE Standard committee (2007–2010). He is the recipient of an IEEE Chester Sall paper award (1998) and of the AEI Ottavio Bonazzi best paper award (1993). He was the Italian Head of Delegation in the ISO-JPEG committee from 1999 to 2018, and is the Italian Head of Delegation in the ISO-Smart Cities committee since its foundation (2016). His research interests are in the area of smart cities, sensor networks and IoT, mobile and professional networks, digital media.