An Event and Service Mesh Architecture Supporting Service Integration in Society 5.0 enabled Smart Cities

Alessandro Calvio
alessandro.calvio@unibo.it
University of Bologna
Bologna, Italy

Andrea Sabbioni
andrea.sabbioni5@unibo.it
University of Bologna
Bologna, Italy

Armir Bujari
armir.bujari@unibo.it
University of Bologna
Bologna, Italy

Luca Foschini
luca.foschini@unibo.it
University of Bologna
Bologna, Italy

ABSTRACT

Society 5.0 envisions a more resilient, sustainable, and human-centered society fostered by ever-evolving cooperation and knowledge sharing among the many digital systems already shaping our daily lives. However, the current state of smart cities often consists of siloed systems, with different actors and stakeholders managing their services and assets independently. This phenomenon is evident in both technological and operational domains, posing challenges to seamless collaboration. In this context, new cloud computing models and technologies like event and service mesh promise to reduce the burden associated with the development and integration of solutions. In the attempt to pave the way for more integrated IT environments, we propose a practical architecture that combines service and event mesh technologies, enabling the seamless exploitation of service invocation and composition based on event distribution and direct service calls. Our proposal allows applications to remain transparent of the underlying technology, facilitating various optimizations on the network and management plane, necessary to meet the diverse operational requirements of complex and heterogeneous applications. We validate our proposal in a real-use case scenario implementation, discussing the tradeoffs that emerge.

CCS CONCEPTS

• **Information systems** → **Information systems applications**; • **Computer systems organization** → *Dependable and fault-tolerant systems and networks*; • **Networks** → *Network management*.

KEYWORDS

Smart City, Service Integration, Event Mesh, Service Mesh, Middleware

GoodIT '23, September 06–08, 2023, Lisbon, Portugal

ACM ISBN 979-8-4007-0116-0/23/09…$15.00
https://doi.org/10.1145/3582515.3609568

ACM Reference Format:
Alessandro Calvio, Andrea Sabbioni, Armir Bujari, and Luca Foschini. 2023. An Event and Service Mesh Architecture Supporting Service Integration in Society 5.0 enabled Smart Cities. In *ACM International Conference on Information Technology for Social Good (GoodIT '23), September 06–08, 2023, Lisbon, Portugal.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3582515.3609568

1 INTRODUCTION

Recently, the concept of Society 5.0 is gaining increased attention and interest. Originating in Japan to promote the next phase of human history, it aims to create a new kind of *smart* and human-centered society, characterized by the pervasive use and seamless integration of technological solutions and digital systems, addressing the principal challenges of our time, improving the quality of life of the collective [20]. Although Society 5.0 efforts have a broader scope, it is clear that people tend to move to cities, and it is foreseen that by 2050 about two-thirds of the world's population will reside in urban areas. As more and more people move to cities, they bring their economic, cultural, social, and political interests, making cities the main center of human activity. Therefore, improving the quality of life in cities can have a significant impact on people's overall well-being.

The smart city paradigm seeks to improve urban infrastructure (e.g., public transportation), clean air and water, and ease access to quality healthcare, education, and cultural activities, making the environment more liveable for their residents. In turn, this can lead to increased productivity, economic growth, and social cohesion, as well as a more sustainable and resilient urban environment from both an individual and collective point of view. Indeed, the smart city concept plays a key role in improving living conditions by exploiting the synergistic interaction between Information and Communication Technologies (ICT) and the Internet of Things (IoT) [12]. Central to this vision is the idea of connection: by creating a network of interconnected devices, services, and people, it is possible to create a highly interoperable and efficient urban environment [11].

Moreover, smart cities become essential components promoting deep integrations among different domains, such as manufacturing, logistics, energy management, and agriculture etc. While this vision embodies many benefits, the reality is that these verticals operate as isolated, separated from the other sectors that make up

the landscape of society [16]. This is mainly because these domains are often managed by different departments or stakeholders within a city (e.g., transportation systems may be managed by a transportation department, while energy systems are managed by an energy company, etc.). In addition, different actors may have different levels of access and ownership over the data generated by their systems or may be concerned about data privacy and security when sharing data between different organizations. All these factors prevent us from acquiring a holistic view of the city's systems, thus, missing opportunities for integrated solutions to achieve common goals. As an example, municipalities could leverage the data from agriculture and logistics to improve the supply chain or from energy to reduce pollution levels.

The integration of these heterogeneous systems requires IT methodologies and services able to cope with the increasing complexity of the solutions we seek (Figure 1). In particular, new models of Cloud Computing offerings allow the offloading of management tasks to cloud providers, leaving customers with the duty of developing the business logic. However, the need of ensuring timely responsiveness and reducing latency for real-time decisions means that modern city infrastructures cannot rely on cloud solutions alone. The same abstraction level needs to be extended on distributed architectures that are spread over a continuum of IoT-Edge-Cloud resources spanning potentially different administrative domains. To support this paradigm, technologies like service and event mesh have emerged as promising solutions, abstracting the logic governing service-to-service communication from individual services, and introducing a dedicated control layer for service management.

In an effort to foster a federated collaborative environment supporting Society 5.0 and smart city sustainable development along different dimensions, our proposal aims to create an innovative distributed infrastructure that integrates service and event mesh approaches to meet the diverse needs and requirements of modern architectures. In our solution, the presence of a quality-aware middleware enables the adoption of a set of abstractions that ensure the transparency of the underlying technology with respect to the applications above. This abstraction layer empowers developers to define their desired qualities of service while offloading the responsibility of the actual implementation to the infrastructure.

The work is structured as follows: Section 2 discusses some technological solutions under scrutiny. Section 3 provides an overview of related work similar in effort to ours. Section 4 introduces the layered reference architecture and its functional components, discussing some technological enablers. In Section 5 we conduct an experimental evaluation of the approach, and finally, Section 6 concludes the work.

2 BACKGROUND

For many years, monolithic software has ruled the development practice. However, as applications start to grow in complexity, this methodology embodies many limitations. Beyond the rigid organisational aspects of the design and implementation process, this approach is not cost-effective and not capable of fully leveraging the benefits at the core of the cloud paradigm, such as resource multiplexing and task pipelining. By breaking down big applications into small independent and loosely coupled services, a microservice architecture provides a more flexible, scalable, and resilient approach to deal with the complexity of enterprise systems from both a development and organizational point of view [13].

Having a set of smaller computational entities allows for being more resilient concerning elevated and sudden changes in demand or usage patterns; each service can be independently scaled, leading to better overall resource utilization [14]. To this end, management and orchestration frameworks platforms such as Kubernetes can be exploited to handle horizontal and vertical scaling of resources depending on the quality specification of applications. Although the microservice-oriented architecture has become an established software engineering practice, connecting, configuring, and managing heterogeneous assets and services belonging to potentially different administrative domains is a challenging and open research question [24].

This problem is further exacerbated as the number of services grows, undermining the reliability of the entire application. Moreover, unlike a monolithic application where software interactions are self-contained, different microservices are not always colocated and may be hosted on distinct nodes, requiring the use of the network to perform successful communication. This leads to the introduction of additional latency and overhead at the system level and, at the same time, of complexity at the structural level since each service must also handle the network communication logic.

Yet, another challenge is the monitoring of the individual services and the service chain as a whole. With many actors working together, it can be difficult to monitor the performance of each service to identify performance bottlenecks or errors. Effective monitoring, metrics collection, and distributed tracking mechanisms constitute essential building blocks for healthy and effective service operations. Finally, security issues are also more critical in architectures of this type. The presence of different services exposes multiple points of attack, and vulnerabilities in one service can compromise the security of the entire system [26].

The advantages brought by the microservices approach introduce several new issues related to configuring, connecting, and maintaining the entities involved, diverting programmers' attention away from developing business logic. To handle the complexity of integration, service and event mesh represent two emerging and promising technologies. The idea behind the use of these technologies is to abstract, through an infrastructure layer, the application logic part from the management and communication part. This is done through the use of software components called proxies that are deployed alongside each microservice and that take care of the data plane part of the infrastructure [6]. In addition to this, a set of external components take care of the control and configuration logic of the proxy components. The main difference between the two technologies lies in the communication paradigm adopted, with a service mesh approach implementing a synchronous interaction between microservices, while the latter uses an asynchronous approach facilitated by the use of brokers.

In this work, we propose an innovative infrastructure support, integrating both technologies into a single platform capable of fostering the diversity of service requirements present in modern smart cities.

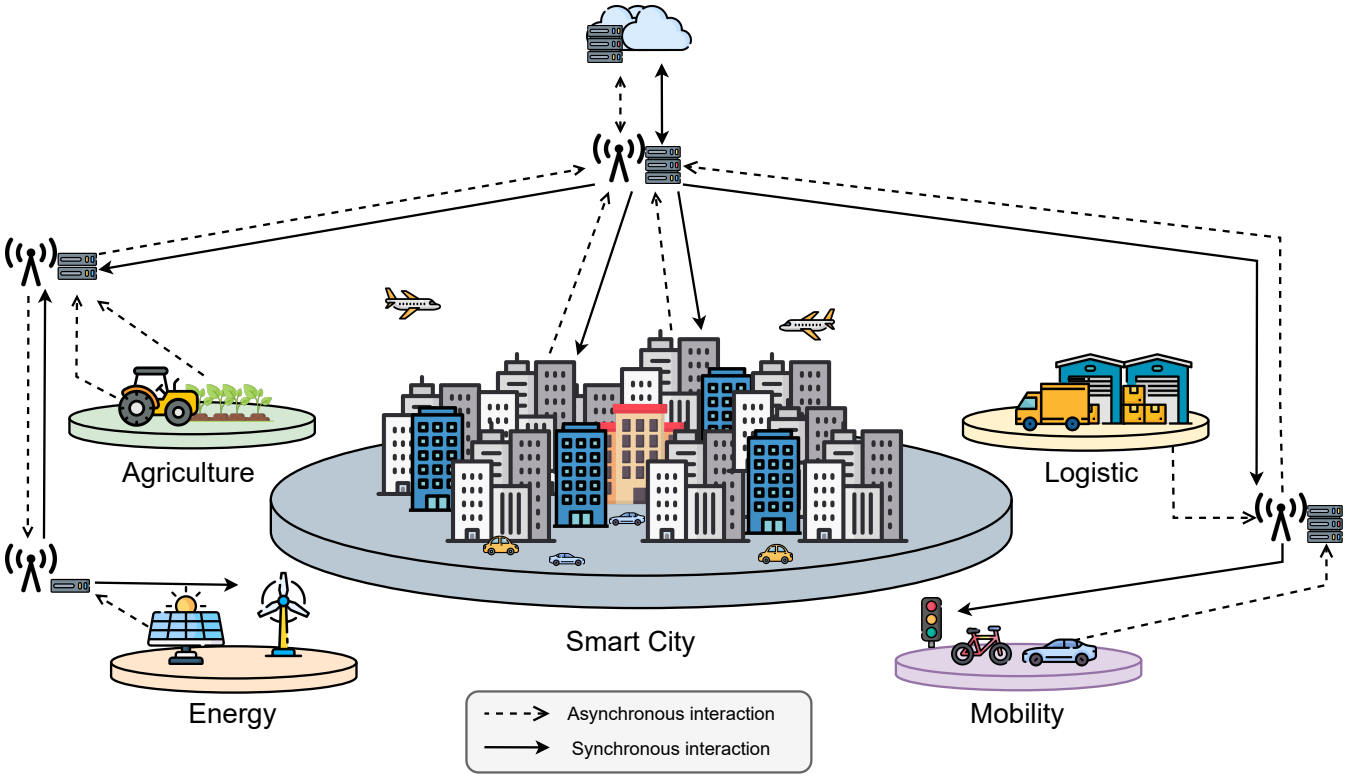


Figure 1: Society 5.0 promotes the integration among the different actors managing different dimensions of Smart Cities' functioning in different ways. This calls for new forms of infrastructures able to leverage both synchronous and asynchronous interactions.

3 RELATED WORKS

Methodologies and mechanisms addressing the effective integration of enterprise systems have long attracted the attention of both academia and industry. In [18], the authors discuss potential challenges associated with integrating heterogeneous IoT devices and networks. To overcome this challenge, the authors emphasize the crucial role of middleware solutions, meant as a set of abstractions that allow developers to focus on the core business logic. The paper also provides an overview of different types of middleware, including event-based and service-oriented approaches.

Additionally, numerous projects are dedicated to addressing the issue of service integration within smart city environments. In [25], the authors present a summary of notable smart city initiatives that specifically tackle service integration. They characterize these initiatives based on three major facilitators: semantic functional description of city objects, a distributed service directory, and planning tools for composing services.

In [19], the authors propose a communication model that combines the Request-Response and Publish-Subscribe paradigms in order to address various functional and non-functional requirements in ubiquitous systems. At the functional level, the model enables synchronous message exchange and asynchronous notifications, making it suitable for developing software applications and services such as chats, task monitoring, multimedia information exchange, and real-time collaboration. The advantage is that developers only need to understand the communication semantics rather than specific mechanisms, which simplifies development and promotes independence from distinct technologies.

To the best of our knowledge, the integration of service mesh and event mesh represents a novel and emerging research area within the field of service integration. In [1], Li *et al.* conducted an investigation into the possibilities of using service mesh as a next-generation technology for achieving seamless integration between microservices. The authors were able to capture the main characteristic of this technology and how it could be exploited in different scenarios ranging from traditional enterprise applications to the telecommunication field. While service mesh offers benefits in managing communication among microservices, reliance on a proxy component embodies some limitations due to the presence of a single point of failure. In their work, Rusti *et al.* [21] applied the concept of service mesh in MATILDA, a framework tackling the overall lifecycle of design, development, and orchestration of 5G-ready applications and 5G network services over programmable infrastructure. Here, the service mesh is deployed in the form of a network slice, closely aware of the available network resources to realize the abstraction plan of the business logic part from the network-specific functionalities.

Similarly, in [23] the authors advocate for the use of a service mesh architecture, specifically emphasizing the benefits of Istio[6], a popular service mesh implementation. The work highlights the standardized and declarative approach offered by the service mesh paradigm, enabling seamless management of interactions and runtime capabilities without requiring modifications to the application code. This research highlighted the advantages of service mesh, providing insights into its usage and benefits for microservices-based applications.

While there is a lack of relevant works specifically addressing the concept of an event mesh in the research area, it is worth noting that the event-driven approaches have been widely acknowledged as an enabler in smart city scenarios.

In [15], the authors developed an event-based architecture that enables the management and collaboration of heterogeneous sensors for monitoring public spaces. In [17], the authors present SEMi, an event-based architecture with components that meet important requirements for smart city platforms, including scalability, flexibility, and heterogeneity in event processing. The architecture is evaluated, demonstrating the effectiveness of event processing in real-time scenarios.

Drawing upon previous research, this paper introduces an innovative architecture designed for the integration of services in smart city environments. Our main contribution regards the investigation of the efficient utilization of both service mesh and event mesh within a unified architecture, catering to the diverse needs of city applications. Additionally, our effort also aims to promote the wider interest in these technologies within the research community.

In the following section, we provide a comprehensive overview of our architecture, discussing its essential layers and functional components.

4 SERVICE AND EVENT MESH PLATFORM TO SUPPORT MULTI-SITE APPLICATIONS

Considering the heterogeneity and constraints of applications and services hosted on potentially different administrative domains, while supporting the development of Society 5.0, a one size fits all architectural style is hardly the solution. Our proposal integrates both event and service mesh paradigms, aiming to overcome their inherent limitations and exploit their complementarities. In particular, considering complex scenarios integrating multiple services and data sources coming from different Society 5.0 actors, the approach enables the joint exploitation of event mesh technology, for those service compositions seeking more decoupled and asynchronous interactions, and service mesh approaches for those requiring more strict, synchronous, and controlled interactions.

To this end, we propose a layered architecture comprised of data (overlay) and a control plane, shown in Figure 2. Through this section, we provide a comprehensive overview of each layer, including their functional responsibilities and how they interact and contribute to the overall system.

4.1 Application Layer

The Application Layer represents the first component within the proposed conceptual architecture, situated in the upper part of the framework. This layer encompasses all the services and assets developed by different actors operating within the Society 5.0 environment and provides tools for them to exploit the infrastructure below. The heterogeneity here mainly materializes on several dimensions, including but not limited to operational requirements and development specification that depends on the purposes and functions of each service.

As an example, let us consider the multitude of sensors in a smart city deployment (such as sensors, vehicles, cameras, etc.) continuously generating valuable (raw) data. These data need to be collected from multiple sources and processed in real-time by high-performance components situated at the edge or cloud levels. The primary operational requirements in this scenario are high throughput and low latency. The focus is on efficiently processing large volumes of data in real-time, with no immediate need for data reliability. Therefore, the emphasis is on handling the massive influx of data swiftly to support timely decision-making.

Another example pertains to the control aspect that a smart city may exercise over its territory. In this case, numerous services may practically interact with the physical assets spread over the city, such as traffic lights or energy grids, to tweak so to optimize their duty cycling. The operational requirements for this type of interaction involve high-reliability constraints so to maintain a consistent state of the system. Point-to-point interactions are necessary, ensuring that commands or changes made to physical assets are reliably executed and consistently reflected across the entire system. Additionally, there is a need for rollback and atomicity of operations, allowing the system to revert to a previous stable state in the event of errors or failures, ensuring the integrity and consistency of the controlled assets.

Applications rely on a Quality-Aware Middleware component, providing a high-level abstraction over the resources, and facilitating the seamless integration between application/service components. An important responsibility of the latter is to provide an abstraction over the underlying communication model employed by applications, thereby enabling developers to prioritize communication semantics and the desired qualities of service. Depending on the application, developers can define their preferred communication semantics, such as reliable or ordered delivery, while also allowing them to specify latency requirements and the necessity for receiving responses from recipients. By abstracting these aspects, the middleware shields developers from the complexities associated with underlying communication technologies and protocols, enabling them to focus on the core aspects of their applications.

Among the abstraction, it offers the capability to establish connections with other services, customized to include desired communication properties, such as specifying the preferred delivery semantics, such as at least once, at most once, or exactly once. Furthermore, the middleware provides flexibility by allowing applications to choose the type of interaction they prefer. A synchronous communication enables blocking requests, while an asynchronous connection facilitates non-blocking interaction, allowing the application to continue processing without waiting for a response. In the case of asynchronous communication, the middleware requires the specification of appropriate handlers to process incoming messages. It is important to note that the abstraction chosen by the service does not necessarily reflect the underlying communication used in the infrastructure. The control plane performs the resolution of the

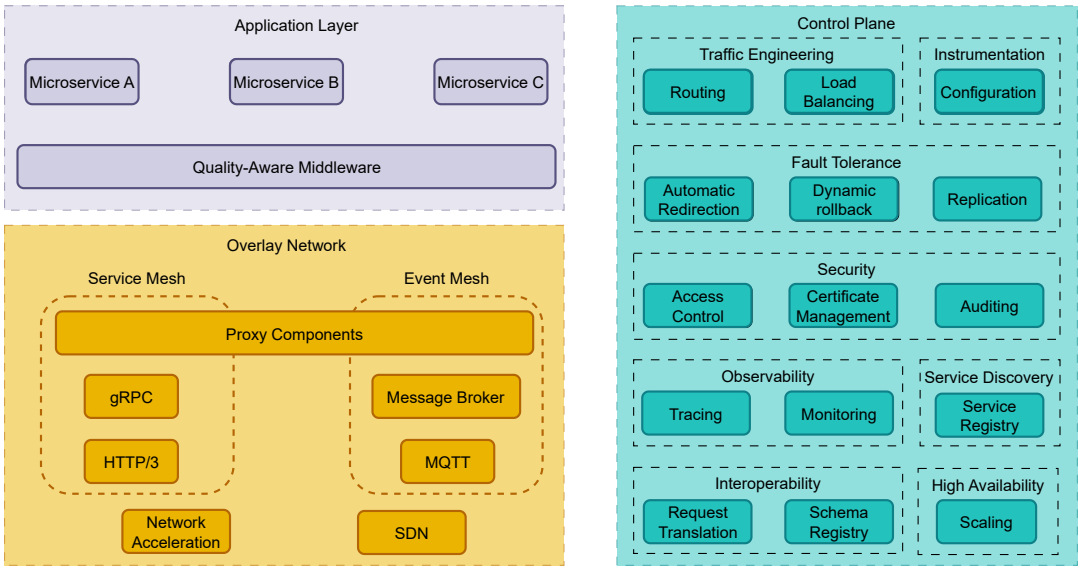


Figure 2: A layered reference architecture to promote service integration by leveraging a network of services and events. The application layer deals with application development, dealing with abstractions, and interfacing applications with the underlying resources. The overlay network deals with the practical issues involved in implementing the correct communication model, while the control plane includes a set of tools to properly manage and coordinate the entire system.

requirements based on criteria such as the foreseen length of the service interaction.

The middleware has the responsibility of selecting the most suitable concrete interaction model and technology to fulfil the prescribed constraints and qualities of service, ensuring that the chosen communication model aligns with the specifications and desired semantics articulated by developers.

On a practical note, the middleware can be implemented as an SDK, exposing a range of abstraction elements, empowering developers to proficiently utilize the platform, and harnessing the advantages proffered by the architecture. By furnishing an SDK, developers gain access to pre-built functions and libraries that can expedite communication with the underlying infrastructure. This activity is currently a work in progress.

4.2 Overlay network

The next macro layer, the Overlay Network, emerges as the first infrastructural layer within the proposed architecture, taking on the responsibility of practically implementing the desired communication semantics according to specifications dictated by the applications. It comprises several components, ranging from the application level (OSI/ISO), where the proper tool technologies and protocols are selected, to the network level, to realize the actual data plane with custom qualities of service.

As highlighted in Figure 2, the first component to come into play is the proxy element. In line with the state-of-the-art in service mesh and event mesh solutions, its primary role revolves around intercepting requests from the Application Layer and forwarding them to their intended destinations. This layer takes charge of implementing the necessary mechanisms for different interaction models, such as synchronous and asynchronous patterns. In practice, this can be done by having separate proxies designated for each interaction model, ensuring that the respective elements appropriately handle the specific details and requirements.

When considering service mesh architectures, there are already established, cloud-native solutions available, such as Envoy[2] and HAProxy [4], that can be readily adopted as implementation choices. These tools have robust support for state-of-the-art synchronous protocols like HTTP/3 and gRPC[3], ensuring efficient and reliable communications. They are also extensible in the sense that custom modules can be implemented to enhance their capabilities. It is important to note that these components are better suited for synchronous interactions.

When it comes to the event mesh, relying solely on a proxy is not sufficient. To support decoupled and transparent interactions effectively, another component is required. This is where message-oriented middleware technology, such as a broker, plays a crucial role. Brokers typically implement queue systems that store messages and facilitate the decoupling of applications, ensuring the correct and reliable transmission of messages, and enabling asynchronous communication within the infrastructure. State-of-the-art message middleware solutions like HiveMQ[5] that support the MQTT protocol or the NATS[7] broker with its homonymous protocol, can be readily utilized. Each proxy can employ any technology that supports message brokering, allowing for the deployment of a federated and heterogeneous network of brokers, enhancing the overall flexibility and adaptability of the architecture.

Enabling effective communication between the proxy component and the upper layers of the architecture is a noteworthy design consideration. In particular, leveraging interprocess communication mechanisms becomes an appealing option. Technologies such as

shared memory can be utilized to establish efficient and low-latency communication channels between the proxy and the upper layers.

On the other hand, in situations where the proxy is remotely deployed, alternative techniques come into play. An option is represented by ReST interactions that are commonly employed in such cases, while another technique that can be employed is RDMA [22], which enables direct data transfer between memory spaces of different machines.

To achieve the desired QoS, a set of techniques that operate at the OSI/ISO L2/L3 levels are required. Network acceleration approaches can play a crucial role in optimizing network performance and enhancing data transfer when stringent latency/bandwidth requirements are requested. In our case, modifications in the proxy component are required to enable them to use such techniques by means of technologies like DPDK (Data Plane Development Kit) or XDP (Express Data Path) which bypass the network stack and move the packet processing to the userspace.

4.3 Control Plane

The Control Plane represents another important macro layer in our architecture, managing and configuring overlay network components, providing mechanisms for the coordination and the actual transmission of messages between hosts. Hence, the role of the control plane is to ensure that the isolated set of proxies work as a coherent and distributed system. This is achieved through a set of components that handle various aspects of the system, including service discovery, security measures, fault tolerance, and more. These components are offered to the user with different levels of granularity, depending on the complexity of the control plane implementation. In some cases, the control plane allows human operators to manually configure each aspect of the system, providing complete control over the configuration. Alternatively, there may be varying degrees of autonomy based on high-level policies set by the administrator. This enables a balance between manual control and automated management, depending on the specific requirements and preferences of the deployment.

Given that our architecture aims to support multi-site service integration, it becomes necessary to administer several pools of services located in different places and managed in heterogeneous ways using various orchestration platforms such as Kubernetes or Nomad[8]. In this context, multiple deployment options are available for the control plane components. One approach involves having a global instance that governs each service, providing centralized control and coordination. Alternatively, a hierarchical structure can be implemented, where control plane elements are deployed in proximity to the services or at aggregated points that offer a holistic view of the configuration and networking landscape.

Going more into detail, the control plane's instrumentation capability has the crucial function of setting up and managing proxy components at the application layer. Its role is to ensure the proxies are created and configured correctly, aligning with the operational requirements specified by the applications. This configuration process ensures proper connectivity between the proxies and their corresponding services. Additionally, for the event mesh part, it is also necessary to provide the proxies with all the proper information (i.e. location, type of broker, protocol, and topic) to interact with the broker.

The instrumentation component of the control plane relies heavily on service discovery to maintain a comprehensive understanding of the system's current configuration. This is essential because the landscape of services can undergo changes due to scaling events or failures. By collaborating closely with the service discovery mechanism, the instrumentation component ensures it has the most up-to-date information about the network locations of service instances to update proxy configurations. Service discovery achieves this by leveraging a distributed database, such as etcd, to serve as the service registry. The service registry acts as a centralized repository that stores and updates the network locations of service instances. It provides a reliable and scalable solution for maintaining an accurate record of the services available within the architecture.

When a proxy starts up, it subscribes to the nearest service registry by providing all the necessary details to describe its specific instance and establishing itself as an active participant in the architecture. Additionally, the proxy has to provide some kind of heartbeat metrics in order to confirm its operational status and detect node failures. The service registry also stores additional metadata that proves useful for other control components within the architecture. This metadata provides valuable insights into the operational constraints and capabilities of specific services.

The metadata associated with each service instance can be used to configure the tracing and observability components within the control plane. Tools like Prometheus[10] or OpenTelemetry[9] allow the extraction of various metrics and indicators, such as response times, error rates, and resource utilization, to assess the health and performance of the services.

Another control aspect within the architecture concerns the fault tolerance capability, which can vary depending on the type of interaction being employed. In case of node failures, synchronous interactions tend to have slower and more expensive recovery processes. When a node fails in synchronous communication, the sender must wait for a timeout to occur before redirecting the requests to an alternative host. This delay can impact the responsiveness of the system. In contrast, asynchronous communication provides a more flexible approach. Interactions in asynchronous communication are not coupled to a single host. Instead, by utilizing mechanisms like grouped subscriptions, messages can be delivered only to active subscribers. On the other hand, service mesh technology enhances fault tolerance by facilitating the implementation of rollback mechanisms to create atomic operations. In cases where a rollback is necessary, the service mesh provides the necessary support to revert to a previous stable state, ensuring the integrity and consistency of operations. However, implementing rollback mechanisms in event mesh architectures can be more challenging. This is due to the inherent difficulty of reconstructing the service chain that served a specific interaction, making the process more complex and potentially limiting the ability to perform efficient rollbacks.

Finally, interoperability of communication becomes a challenge in the architecture due to the heterogeneity of interactions. Ensuring effective collaboration between services with different supported formats or interaction requirements necessitates the transformation and adaptation of information to align with the specific formats they use at the application level. The control plane takes on the responsibility of storing all the relevant information regarding the format supported by each service, along with the schema of the actual data. This information is stored in a schema registry, which acts as a distributed repository for schemas and format specifications.

5 PERFORMANCE EVALUATION

In the following, we present a preliminary experimental assessment of a proof of concept implementation of the proposed layered architecture, assessing service and event mesh approaches.

5.1 Experimental Settings

With reference to the scenario in Figure 1, the investigation of service chains is relevant because of the multiplicity of application domains that could be integrated. For example, a service chain could be concerned with the use of traffic data to calculate optimal routes for delivery and order fulfillment purposes. Here several services could be included: one service could deal with the collection of traffic data from the various city sources, a second could use it to calculate the level of traffic and congestion, and a third would use this information to calculate the optimal route to meet delivery times.

To assess our proposal, we implemented two use case scenarios consisting of interconnected services, each operating in conjunction with a pair of proxies that enable synchronous and asynchronous interactions, respectively.

The first use-case, adhering to the service mesh approach, has been developed as a ReST service interacting through synchronous interactions with their sidecar proxies. In this case, we rely on the Envoy proxy, a widely recognized and extensively used technology in service mesh solutions such as Istio. To elaborate on our setup, each proxy is configured with two Envoy listeners that facilitate smooth bidirectional ReST communications among the various services in the network. The first listener acts as an entry point, forwarding incoming requests from the associated benefit to the network, while the second listener enables the reception and processing of messages from the network back to the client. To ensure appropriate message routing, each Envoy listener is equipped with an HTTP Communication Manager filter, which directs each message to the correct destination. This configuration assumes that each proxy has complete knowledge of the services present in the system, thereby facilitating efficient and accurate message routing.

In the second use-case, the asynchronous interaction of the systems is facilitated through the utilization of NATS, a high-performance message middleware, deployed on a separate node. To enable the asynchronous communication of the services, we built a dedicated proxy as a REST service that accepts requests from its respective service and forwards them to the NATS message broker. During the bootstrap phase, each proxy subscribes to a specific topic unique for each service (e.g. the proxy of service A is mapped to topicA, etc.). When a service requires asynchronous communication, it sends a request to its corresponding proxy. It encapsulates the request into a message and publishes it to the destination topic on the broker. This approach allows the service to proceed without waiting for the complete propagation of messages throughout the system. Similarly, when the proxy consumes messages from the mapped topic, it extracts the request data and forwards it to the appropriate service for processing.

In our experimental study, we aim to investigate and analyze the performance of the system with varying numbers of chained services. In specific, we consider five different configurations starting with 2 chained services up to a maximum of 5 together with their proxies (Figure 3), all deployed as containerized functions on separate VMs connected through a 1Gbit network in a star topology. Each node consists of a virtual machine with Ubuntu 22.04 LTS operating system, equipped with 8GB of RAM and 4 virtual CPUs. The system was stressed using the JMeter tool colocated with the first service in the chain.

5.2 Results

Figure 4a shows the end-to-end latency in milliseconds, exploiting both synchronous and asynchronous interaction. On the x-axis, we considered the number of concatenated services and, by extension, the number of steps each request must go through sequentially before it can provide a response. Each configuration was tested with a constant request ratio of 20 request/s over a 5-minute period. In the synchronous case, we observed a significant upward trend in latency times. Initially, the latency values were below 10 milliseconds, but they dramatically increased to approximately 9 seconds as the number of services in the chain increased. In contrast, the asynchronous case demonstrated a more consistent performance, with latency ranging between 25 and 40 milliseconds, even when utilizing a chain of 5 services.

The observed behavior stems primarily from the correlation between the request load and the number of concurrent connections that each service needs to handle during message processing. In periods of high request load or sudden spikes in traffic, numerous clients may simultaneously establish new connections to the services. This influx of connection requests can overpower the server's capacity to handle and process them, leading to queuing as excess connection requests await available resources, including CPUs, RAM, and bandwidth. This is particularly evident in the transition from 3 to 4 microservices, where the phenomenon of queuing starts to become dominant in the performance of the entire system. Additionally, the time for resources to become available again depends on the processing time of each service.

The queuing phenomena are further amplified by the sequential execution nature of services. This means that the first services in a service chain encounter lengthier waiting times as they must wait for all subsequent services to finish their internal logic. The waiting time is compounded by the fact that resources occupied by these initial services cannot be freed until the entire chain completes processing.

The asynchronous interaction model, facilitated through the utilization of a message broker, addresses these issues. It enables the system to decouple the request-handling process from immediate

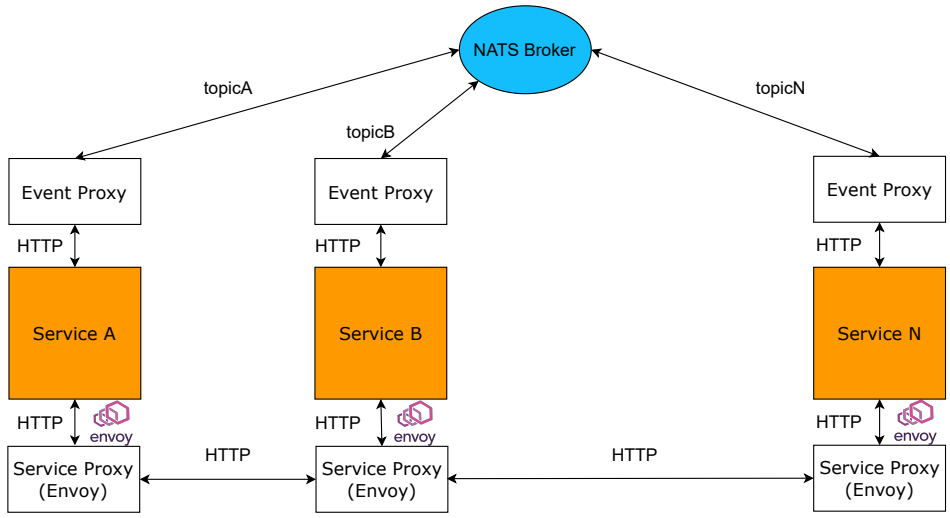
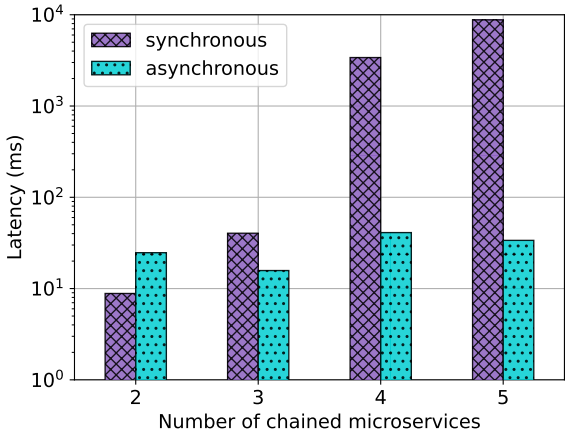
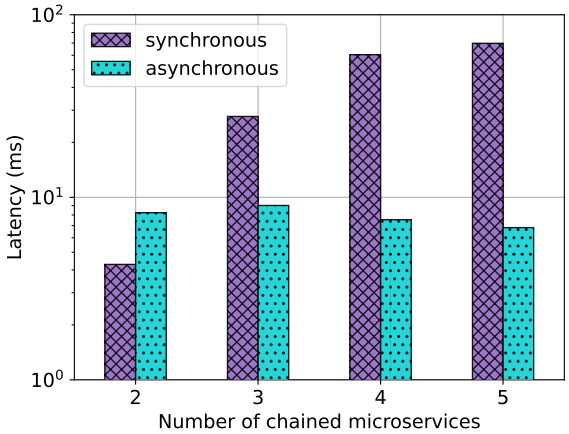


Figure 3: Deployment architecture used to assess the reference scenarios



(4a) End-to-End latency varying the number of chained services with a traffic load of 20 request/s over a 1-minute period



(4b) Response latency varying the number of chained services with a traffic load of 20 request/s over a 1-minute period

responses, allowing for parallel processing of multiple requests. Instead of waiting for each request to finish before proceeding to the next one, requests can be dispatched to the appropriate services and processed independently.

This approach enhances the utilization of system resources. Rather than dedicating resources exclusively to a specific request until it completes processing, resources can be swiftly released and made available for handling subsequent requests. Consequently, the system can effectively manage a higher request load without overwhelming resources or experiencing significant queuing phenomena.

Figure 4b provides a closer examination of the previous graph, focusing on the final response time, in milliseconds, of the service chain, which represents the time for the response to travel from the last service in the chain back to the first.

The graph exhibits a similar trend to the previous analysis, where the latency times in the synchronous case increase as the number of concatenated services grows, ranging from 3 milliseconds to approximately 70 milliseconds. This can be attributed to the fact that each response must traverse the entire chain in reverse, resulting in increased latency. Conversely, in the asynchronous case, the response times remain stable between 7 and 9 milliseconds. This is mainly because of the direct delivery of the response to the respective service without the need to pass through intermediary services. As a result, the response always requires a single step, in contrast to the previous case where the number of hops is proportional to the length of the service chain.

Based on these findings, it becomes evident that the synchronous interaction offers distinct advantages, particularly for smaller service chains where the overhead introduced by including a third

actor like the broker does outweigh the cost of maintaining connections. However, as the complexity of interactions increases, the asynchronous approach proves to be more advantageous in terms of ensuring improved response times and overall quality of service.

6 CONCLUSION

The continuous evolution of technologies is rapidly shaping our society, craving for a more interconnected and cooperative environment. In this context, the development and affirmation of new models of cloud computing bridge the gap with the rising complexity of interconnecting systems and dealing with highly distributed infrastructure. To support more effective integration of Society5.0 services we proposed an innovative architecture combining service and event mesh approaches enabling an efficient and transparent exploitation of both asynchronous decoupled and synchronous coupled communication paradigms. Results showed that both approaches have distinct advantages in terms of communication capabilities and preservation of the quality of service, motivating their joint use. By necessitating an additional layer such as the federated network of brokers, asynchronous communication does not fit scenarios with small interactions due to the overhead introduced. In this case, synchronous approaches show their potential in maintaining the overall latency low. On the other hand, as the complexity starts growing the limitation of blocking communication manifests their limitation calling for more decoupled techniques. Different aspects can be addressed to expand the proposed architecture. One direction is to evaluate the applicability of the architecture in the context of the cloud continuum. This involves exploring how the architecture can effectively leverage heterogeneous resources and adapt to new network technologies, such as Wi-Fi 6 and 5G, to ensure optimal performance and scalability. Another area regards the development of optimization techniques for proxy distribution. This includes investigating algorithms and strategies to efficiently allocate and manage proxies across the infrastructure.

REFERENCES

[1] 2019. Service Mesh: Challenges, state of the art, and future research opportunities. *Proceedings - 13th IEEE International Conference on Service-Oriented System Engineering, SOSE 2019, 10th International Workshop on Joint Cloud Computing, JCC 2019 and 2019 IEEE International Workshop on Cloud Computing in Robotic Systems, CCRS 2019* (5 2019), 122–127. https://doi.org/10.1109/SOSE.2019.00026

[2] 2023. Envoy Proxy. https://www.envoyproxy.io/ Accessed: 2023-06-9.

[3] 2023. gRPC. https://grpc.io/ Accessed: 2023-07-16.

[4] 2023. HAProxy. https://www.haproxy.com/ Accessed: 2023-06-9.

[5] 2023. HiveMQ. https://www.hivemq.com/ Accessed: 2023-07-16.

[6] 2023. Istio Architecture. https://istio.io/latest/docs/ops/deployment/architecture/ Accessed: 2023-06-9.

[7] 2023. NATS.io. https://nats.io/ Accessed: 2023-07-16.

[8] 2023. Nomad. https://www.nomadproject.io/ Accessed: 2023-07-16.

[9] 2023. OpenTelemetry. https://opentelemetry.io/ Accessed: 2023-07-16.

[10] 2023. Prometheus. https://prometheus.io/ Accessed: 2023-07-16.

[11] Bengt Ahlgren, Markus Hidell, and Edith C.-H. Ngai. 2016. Internet of Things for Smart Cities: Interoperability and Open Data. *IEEE Internet Computing* 20, 6 (2016), 52–56.

[12] Paolo Bellavista, Carlo Giannelli, Stefano Lanzone, Giulio Riberto, Cesare Stefanelli, and Mauro Tortonesi. 2017. A Middleware Solution for Wireless IoT Applications in Sparse Smart Cities. *Sensors* 17, 11 (2017).

[13] Tomas Cerny, Michael J Donahoo, and Michal Trnka. 2018. Contextual understanding of microservice architecture: current and future directions. *ACM SIGAPP Applied Computing Review* 17, 4 (2018), 29–45.

[14] Nathan Cruz Coulson, Stelios Sotiriadis, and Nik Bessis. 2020. Adaptive Microservice Scaling for Elastic Applications. *IEEE Internet of Things Journal* 7, 5 (2020), 4195–4202.

[15] Luca Filipponi, Andrea Vitaletti, Giada Landi, Vincenzo Memeo, Giorgio Laura, and Paolo Pucci. 2010. Smart City: An Event Driven Architecture for Monitoring Public Spaces with Heterogeneous Sensors. *IEEE International Conference on Sensor Technologies and Applications*, 281–286.

[16] Riccardo Petrolo, Valeria Loscrí, and Nathalie Mitton. 2014. Towards a smart city based on cloud of things. *Proc. of ACM International Workshop on Wireless and Mobile Technologies for Smart Cities*, 61–66.

[17] Jurairat Phuttharak and Seng W. Loke. 2023. An Event-Driven Architectural Model for Integrating Heterogeneous Data and Developing Smart City Applications. *Journal of Sensor and Actuator Networks* 12 (2 2023), 12. Issue 1.

[18] Mohammad Abdur Razzaque, Marija Milojevic-Jevric, Andrei Palade, and Siobhan Clarke. 2016. Middleware for Internet of Things: A Survey. *IEEE Internet of Things Journal* 3 (2 2016), 70–95. Issue 1.

[19] Carlos Rodríguez-Domínguez, Kawtar Benghazi, Manuel Noguera, José Luis Garrido, María Luisa Rodríguez, and Tomás Ruiz-López. 2012. A Communication Model to Integrate the Request-Response and the Publish-Subscribe Paradigms into Ubiquitous Systems. *Sensors* 12 (6 2012), 7648–7668. Issue 6.

[20] Carolina Narvaez Rojas, Gustavo Adolfo Alomia Peñafiel, Diego Fernando Loaiza Buitrago, and Carlos Andrés Tavera Romero. 2021. Society 5.0: A Japanese Concept for a Superintelligent Society. *Sustainability* 13 (6 2021), 6567. Issue 12. https://doi.org/10.3390/su13126567

[21] Bogdan Rusti, Horia Stefanescu, Jean Ghenta, and Cristian Patachia. 2018. Smart City as a 5G Ready Application. *Proc. of IEEE International Conference on Communications (COMM)*, 207–212.

[22] Andrea Sabbioni, Lorenzo Rosa, Armir Bujari, Luca Foschini, and Antonio Corradi. 2022. DIFFUSE: A DIstributed and decentralized platForm enabling Function composition in Serverless Environments. *Computer Networks* 210 (6 2022), 108993.

[23] Ozair Sheikh, Serjik Dikaleh, Dharmesh Mistry, Darren Pape, and Chris Felix. 2018. Modernize Digital Applications with Microservices Management Using the Istio Service Mesh *(CASCON '18)*. IBM Corp., USA, 359–360.

[24] Bhagya Nathali Silva, Murad Khan, and Kijun Han. 2018. Towards sustainable smart cities: A review of trends, architectures, components, and open challenges in smart cities. *Sustainable Cities and Society* 38 (4 2018), 697–713.

[25] Fikret Sivrikaya, Nizar Ben-Sassi, Xuan-Thuy Dang, Orhan Can Gorur, and Christian Kuster. 2019. Internet of Smart City Objects: A Distributed Framework for Service Discovery and Composition. *IEEE Access* 7 (2019), 14434–14454.

[26] Muhammad Usman, Simone Ferlin, Anna Brunstrom, and Javid Taheri. 2022. A Survey on Observability of Distributed Edge & Container-Based Microservices. *IEEE Access* 10 (2022), 86904–86919.