Towards a Modular Architecture for eXtended Reality Systems*

Megha Quamara[1][0000−0001−9380−6916], Luca Capra[2], Valeria Villani[3][0000−0001−7619−0101], Cristiano Carlevaro[2], Oya Celiktutan[4][0000−0002−7213−6359], Alessandro Peretti[2][0000−0001−8027−5917], Marco Piazzola[2], Andrea Ruo[3][0009−0007−2183−2884], Lorenzo Sabattini[3][0000−0002−2734−5549], Viktor Schmuck[4][0000−0002−4308−1352], and Luca Viganò[1][0000−0001−9916−271X]

[1] Department of Informatics, King's College London, London, UK
{megha.quamara, luca.vigano}@kcl.ac.uk
[2] Spindox Labs srl, Trento, Italy {luca.capra, cristiano.carlevaro, alessandro.peretti, marco.piazzola}@spindox.it
[3] Department of Sciences and Methods for Engineering (DISMI), University of Modena and Reggio Emilia, Italy
{valeria.villani, andrea.ruo, lorenzo.sabattini}@unimore.it
[4] Department of Engineering, King's College London, UK
{oya.celiktutan, viktor.schmuck}@kcl.ac.uk

Abstract. For full-fledged social acceptance of eXtended Reality (XR) systems, emphasis should be on design prototypes to allow frictionless, context-aware, and secure interaction with non-specialized users. This necessitates a modular architecture to ensure that the system is versatile and applicable across applications, and is open to the integration of interaction modalities. We discuss our proposal for (and prototypical implementation of) a modular architecture for XR systems that relies on cloud infrastructure resources and edge computing frameworks with shared communication protocols for scalability. The modules are abstracted from both functional and non-functional requirements, including security.

Keywords: eXtended reality system · Modular architecture · Design · Implementation · Social acceptance.

1 Introduction

Context and Motivation. eXtended Reality (XR) is an umbrella term encompassing three primary constituents, *Virtual Reality (VR)*, *Augmented Reality (AR)*, and *Mixed Reality (MR)*, each extending the boundaries of human perception by offering a different interactive and immersive experience for human users. XR

* This work was supported by Horizon Europe program under the Grant Agreement 101070351 ("SERMAS: Socially-acceptable eXtended Reality Models and Systems") and by Innovate UK.

systems employ a spectrum of technologies like computer vision, motion tracking, and advanced displays that blend the physical (or real) world and the virtual world (or digital content) [6]. The scope of XR is extensive, driven by ongoing technological advancements (like improved hardware and software, wireless connectivity, Artificial Intelligence (AI) and machine learning integration) and cross-platform compatibility that continuously push its boundaries and possibilities. As XR continues to evolve, its implications are evident in diverse settings, like healthcare [3], Industry 4.0 [5], Internet of Things (IoT) [2], and industrial training [7].

To realize complex XR systems, a few architectures have been proposed in the literature, ranging from those addressing inter-device communication in industrial AR applications [9] to those using distributed data transmission models [13] and mobile device integration [16] in MR-based applications and services. However, these proposals are limited in addressing the challenges of expanding or upgrading the existing XR systems to accommodate more users, devices, or interaction modalities based on the systems' deployment contexts. Modular approaches, being adaptable, can be applied across various domains to improve the systems' scalability and offer the flexibility of incorporating new hardware and software components. However, existing modular architecture proposals like [25] do not provide an execution environment where the XR systems based on these architectures can support the integration of routines for diverse languages, human perception, and human-system interaction behaviors. Implementing such systems is even more technically challenging due to conflicts, inconsistencies, etc., across different deployment contexts and applications.

Contribution. In this paper, we discuss our proposal for a modular architecture that allows for the applicability of XR systems across applications. The architecture comprises hardware, software, and algorithmic modules derived from abstracting the system requirements. We implemented this architecture relying upon cloud infrastructure resources, supported with edge computing frameworks and shared communication protocols, to allow for scalability towards new application scenarios. XR systems built on this architecture can address the additional requirements introduced by these scenarios by instantiating the corresponding module. The architecture allows for fully distributed scenarios, irrespective of where the involved runtime is located. Another relevant trait of our architecture is that it facilitates building socially acceptable XR systems that can be used in diverse social contexts. Social acceptance influences the user experience of XR systems for their wide-scale adoption. Thus, we aim to ensure that the systems will satisfy the requirements and the needs of end users in compliance with the social context in which they are deployed, emphasizing features like security and privacy.

Organization. We proceed as follows. Section 2 presents the state-of-the-art. Sections 3 and 4 detail the proposed XR system architecture and its implementation. Section 5 concludes and discusses future work.

2 State-of-the-Art

We discuss the state-of-the-art to put our work into content and motivate the need for a modular architecture for XR systems. Table 1 provides a comparative summary of some architectures that have been proposed recently for such systems.

Table 1: Comparative summary of related works and our proposed architecture.

Ref. (Year)	System type	Concern of interest	Applicability	Modularity	Security
[18] (2014)	MR	Mobile device support	Mobile applications	Not supported	No
[9] (2018)	AR	Device communication	Industry 4.0 shipyard	Not supported	No
[13] (2018)	MR	Sensor data transmission	Real-time collaborative systems	Not supported	No
[16] (2019)	MR	Device integration to IoT	IoT services	Not supported	Not detailed
[25] (2020)	XR	Human-system interaction	Industrial manufacturing	Supported	No
[20] (2021)	AR	Application deployment	Web browsers	Not supported	Yes
Our work	XR	Human-system interaction	Application-agnostic	Supported	Yes

In [9], Fernandez et al. proposed an architecture to streamline real-time device communication in industrial AR applications. The architecture relies on fog computing to reduce latency and offload cloud systems, as well as cloudlets to address fog gateways' limited computing power for computationally intensive services. For MR applications, some specific architecture variations exist for supporting mobile MR devices [18], device integration to IoT services [16], and distributed sensor data transmission based on client-server and peer-to-peer models [13].

Serras et al. [25] proposed an Interactive XR (IXR) architecture for enhancing non-savvy operators' capabilities in the Industry 4.0 paradigm. IXR references the Spoken Dialogue System's architecture to implement different AR mechanisms by integrating modules for voice-enabled human-system interaction. Despite IXR [25] focusing on the usability goals, it is abstract, lacks articulation of modules' technicalities, and is limited to verbal modes of interaction.

From the application deployment viewpoint, Pereira et al. [20] presented an AR-based architecture, ARENA, for building and hosting multi-user interactive XR applications on WebXR-capable browsers based on the user's location. This is achieved by capturing system resources' states in scenarios to reduce latency and for runtime management.

XR systems often involve intricate hardware and software components and need to scale to meet application-specific requirements. For social acceptance of these systems, it is required that they adapt to how non-specialized users perceive and engage with them in a social context. An architecture is thus called for that, at the same time, (i) can provide a structured framework to capture hardware and software components for engineering such complex systems that are applicable in different contexts, (ii) supports modularity to allow scaling up and extensions of the systems, (iii) enables social acceptance of the systems. Unfortunately, none of the existing architectures provides all these features in

unison. More specifically, existing architectures do not provide a holistic system addressing human-system interaction, and they are limited in combining cloud and edge computing to address functional and non-functional requirements like security and privacy, given the social context of the system, regardless of the application domain.

To realize an XR system that combines hardware, software, and algorithmic modules to provide an XR experience that enables socially acceptable interactions with non-specialized users and adapts to the deployment context, we need to adopt a modular architecture that supports different interaction modalities (e.g., verbal and non-verbal, like user's facial expressions) across applications, ensures general applicability by being application-agnostic (i.e., not applicable only to a specific kind of setting like IoT), and is open to the integration of external inputs. In the remainder of this paper, we discuss our proposal for such an architecture. The designed modules incorporate requirements and end users' needs, specifically to support a large set of languages and enable new interaction forms to avoid bias whilst ensuring accessibility and security, which, in the present context, we consider representative features of the system's social acceptability.

3 Proposed Architecture

The central concept of our proposed XR system architecture is *modularity*. The system is composed of separate but highly interconnected modules, where each module performs specialized tasks and can interact with other modules through two well-known architectural styles: *Representational State Transfer Application Programming Interface* (RESTful API) [10] and *publish/subscribe* (PUB/SUB) events [8]. The collective set of modules facilitates proper system functioning.

The overall architecture, shown in Fig. 1, enables fully distributed scenarios, regardless of the location of the involved runtime. XR runtimes can leverage remote resources transparently to enable complex service compositions and a mesh of different runtimes, facilitating multi-user and multi-domain interaction models. This can significantly enhance social acceptance of XR systems by providing versatile and inclusive experiences that cater to diverse user needs and preferences in various contexts.

Organization into Modules. Modules are integrated within the platform via integration points, which provide connections across various interfaces. The modules are defined to distribute responsibilities for implementing the required functionalities and simplifying the implementation of new requirements. To achieve this, we used a requirement-driven approach to identify the necessary modules for the system, starting from the requirements (e.g., functional, non-functional, security, privacy) and, where possible, consolidating and abstracting them to identify the corresponding and required functionality. Consequently, a module represents a piece of software and/or hardware of the system that implements a set of requirements or a single requirement in some cases.

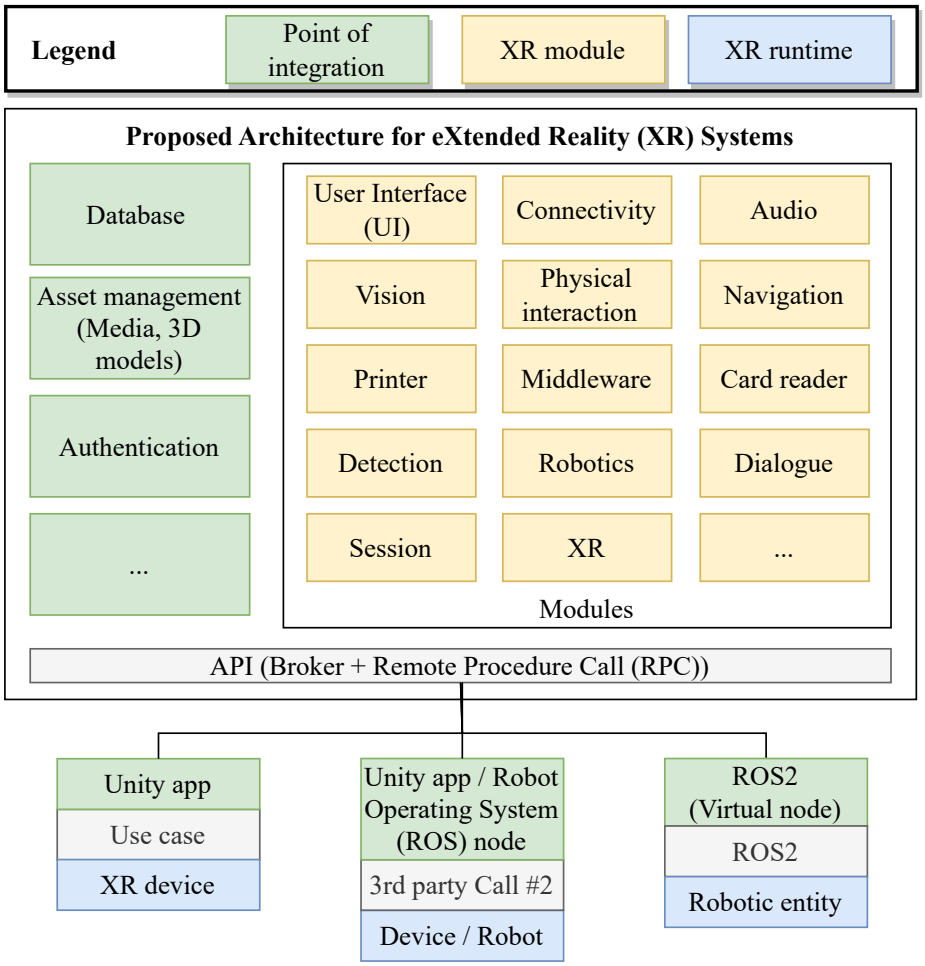


Fig. 1: Proposed architecture for eXtended Reality (XR) systems.

We considered this approach for the following advantages. Firstly, it enables a clear identification of the responsibilities of each module, reducing the risk of leaving some requirements unaddressed. Secondly, it allows scalability towards new application scenarios and use cases. The system can address any additional requirements introduced by a new use case by implementing a corresponding module that does not conflict or overlap with existing ones. This can be facilitated by an architecture that supports straightforward integration of new modules and implements a shared communication protocol among the modules. Hence, adding, changing, or deleting a (third-party) component without affecting the rest of the system requires less effort, unlike with a non-modular approach.

Specifically, each module can be defined according to the following specifications: (i) *functionalities*; (ii) *hardware specifications*; (iii) *software specifications*;

(iv) *input/output from/to other modules*; (v) *addressed requirements*. The list of modules required to make the system socially acceptable will be discussed in Section 4.

Communication Model. To facilitate communication among modules, a unified communication model at the module or application level is proposed, as illustrated in Fig. 2. This model encompasses the following exchange patterns:

- ***Request/Response Model over HTTP and JSON:*** This communication model provides request/response patterns, wherein a client "calls" a server by sending an HTTP request, typically in JSON format, and the server responds with an HTTP response, also often in JSON format. A straightforward use case is retrieving information by initiating an instantaneous synchronous task (e.g., turning on a light or opening a gate).
- ***Event-driven Messaging Model via MQTT:*** This communication approach offers a PUB/SUB model and is suitable for message-passing events and near real-time notifications from the system. The process is generally asynchronous and is best suited for use cases where there is no specific timing between data exchanges or where the amount of data sent does not require explicit feedback from the receiver(s) (e.g., when a motion sensor detects an activity, it sends messages to the subscribed devices, allowing lights to activate and cameras to record real-time events).

Having a dual mode of communication is advantageous in terms of flexibility, scalability, and integration capabilities. For instance, by incorporating PUB/SUB capabilities, the system can better handle large volumes of events or messages, which can be challenging to achieve with only RESTful APIs. Also, PUB/SUB systems excel at real-time communication. When combined with RESTful APIs, these systems can address both real-time and request-response communication. Further, each module shares *Swagger* [26] for RESTful API and PUB/SUB events. Swagger is a set of open-source tools and specifications that provide a standardized way to describe, document, and consume RESTful APIs and PUB/SUB events. It does not specify the method for exchanging information with the API module but helps define and document the API structure.

Interoperability among Modules. Interoperability among modules and applications is achieved by using two open standards, viz., *OpenAPI* [19] and *AsyncAPI* [4], which provides a formalized way to define data models (including message payloads) and endpoint interfaces. OpenAPI offers tools and standardized definitions for APIs based on the request-response communication model. AsyncAPI is a recent initiative supported by the Linux Foundation, which aims to provide means to describe the PUB/SUB communication model. From the specifications of the considered use case or application scenario, other tools can create wrappers with client or server code for use during development, formal validation of sent and received messages, generation of user interfaces used as documentation and, in some cases, interaction with server-side endpoints. For

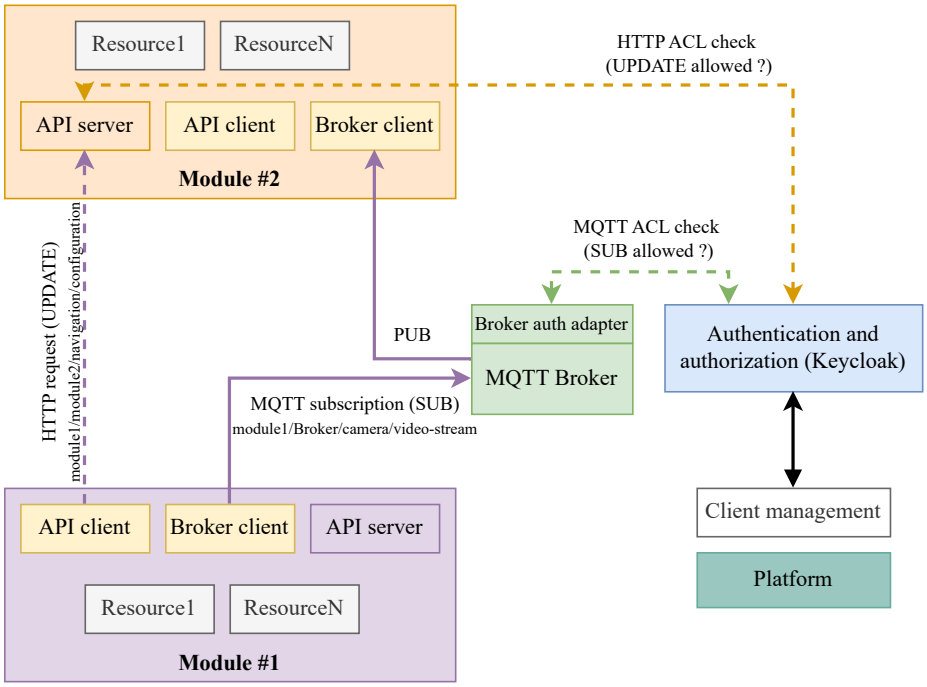


Fig. 2: Model for unified communication among modules.

modules running on the same XR platform (e.g., a game engine), communication among modules is managed within the platform itself, without the need for wrapping.

Security. We also incorporate solutions to address security, as it is one of the representative features of social acceptance in the present context. Establishing security requires treating the XR system as a socio-technical system, in which technology and human users interact by exchanging messages and data, and in which both technology and humans contribute to the security or can cause vulnerabilities. In this case, we used a message-signing approach based on the JSON Web Token (JWT) [14] to ensure the integrity of messages exchanged between modules. The signed JWT tokens allow to check whether the information contained within the token has been tampered with during communication. Likewise, inspired by Zero Trust Architecture (ZTA) [24], a centralized component in charge of authentication and authorization can identify and decide which user can retrieve information or perform action on protected resources. This will help in mitigating concerns related to the safeguarding of user information, unauthorized use of interaction data, and manipulation of XR experiences, which can contribute to social acceptance by ensuring the integrity of the technology.

4 Implementation

We now describe our toolkit as an extension of a prototypical implementation of the proposed XR system architecture. It comprises a set of methods and tools to simplify the design, development, deployment, and management of socially acceptable XR systems.

4.1 Description of the Modules

The toolkit comprises the following core modules, among others, that aim at realizing a representative set of socio-technical requirements:

- ***Detection module:*** This module supports the perception of the world through different sensors, prioritizing on-edge processing to limit the transmission and potential diffusion of user data. As hardware specification, it requires a camera and a microphone. It carries out tasks such as user recognition (the user is standing in front of the camera), determining user intention (the user's intention to interact with the system), user characterization in terms of mood (sentiment analysis), user age estimations, and user sentiment extraction from speech.
  We applied different technologies to achieve these functionalities. For example, for user detection and characterization, we used the *Faceapi Javascript* [15] library. This library allows the detection and recognition of user faces for web applications by extracting emotions from face analysis and age estimations. It is implemented on top of Tensorflow.js, a Google AI library for Javascript language.
  Specifically, we used three pre-trained AI models to extract the information about the user:
  - *Tiny face detector* based on TinyYolov2, to detect a face in front of the camera;
  - *Face expression net*, to detect user expression (i.e., happy, fearful, sad, disgust, angry, surprise, and neutral);
  - *Age gender net*, to estimate the age of the user in front of the camera.

  Furthermore, this module can perform object detection through the camera. Regarding the sentiment extraction from the user speech, we used the *SpeechBrain* [27] toolkit. It is an open-source, all-in-one speech toolkit based on PyTorch. It can perform different tasks, such as speech recognition, speaker recognition and diarization, speech enhancement and separation, multi-microphone signal processing, end-to-end spoken language understanding, and language modeling.
- ***Robotics module:*** This module enables seamless interaction between the physical and XR-based virtual worlds. It is an interface between the architecture and components based on ROS2 [23], an open-source robotic operating system. It allows communication with several robot applications, such as robotic arms.
  Having a module that is a gateway towards ROS2 applications opens the XR architecture to have extensive interoperability with a range of robotic

systems, as ROS2 is widely used in robotics. This component enables communications within the distributed system and allows direct exchange of events from robots to the other modules. The type of robot depends on the specific application scenario. For instance, if there is a need for the robot to move within an indoor environment, a mobile base would be required. Conversely, if there is a need to grasp or hand over objects, a robotic arm is needed.

- ***Dialogue module:*** This module enables natural language interaction with the XR systems and performs speech-related tasks. It can get user speech from a microphone to perform tasks, such as converting speech to text, sending text to a Large Language Model (LLM), and getting the response. Currently, two different services, viz., *whisper* [21] and *speech-to-text* by Google [12] are integrated for converting speech to text and can be used alternatively. The module gets the text from the user's speech and injects it into an LLM (LLM prompting). So far, the LLMs integrated inside the architecture are *chatGPT* [17] and *google-flan-t5* [11]. The LLM is configured to get the responses in chunks to speed up its real-time response.
- ***User Interface (UI) module:*** This module offers points of interaction to enable integrated multi-modal UI features. It allows manipulating the system's graphical interface, such as creating buttons, forms, and cards. Depending on the application contention, it is required to choose the way to represent the UI and the type of screen to use.
- ***Session module:*** This module provides functionality for handling different types of sessions inside the system. One session type keeps track of the agent's status. An *agent* is a high-level functionality which uses one or more modules to perform specific tasks. Through this session, it is possible to recognize system errors for debugging. Another session type is a *user* session, which can track the user's interaction with the system from start to end. For example, when the user approaches the camera, the session starts to couple user interaction with the system until the interaction ends. By using sessions, the system can manage multiple interactions or agents simultaneously.
- ***XR module:*** This module handles XR environment settings. Currently, it is possible to add, delete, or update markers. A *marker* is an object, picture, or QR code to track static or dynamic objects in the user environment. Based on the marker type, the user's position in the environment can be recognized—for instance.
  Moreover, this module can provide information about the occlusion of a 3D object. For example, given the user's position and a 3D asset model, the module indicates whether a physical obstacle blocks the 3D object.

By leveraging OpenAPI and AsyncAPI specifications (cf. Section 3), tools can 1) generate code to ease development and integration, 2) facilitate formal validation and generation of user interfaces, and 3) ease documentation. By using these standards, the level of abstraction is high, making the modules implementation-agnostic in terms of programming languages. Each module has its codebase and can be developed, deployed, and maintained independently. In such a case, if a module fails, it may not affect the entire application.

4.2 Access Control

We implemented our XR system architecture with the ZTA model (cf. Section 3). In ZTA, there is no default trust for internal or external parties of the system [1]. If a module exchanges data with another module via REST call or PUB/SUB events, it must be authenticated and authorized through a JWT token. Communication is allowed if the module has permission to exchange (send or receive) data with another module. To manage authentication and authorization, we used *Keycloak* [22]. It is open-source software that allows single sign-on with identity and access management, aimed at modern applications and services. It offers an OAuth 2.0 and OpenID-compatible implementation and integrated user management.

The toolkit follows an Access Control List (ACL) pattern, focusing on resource-based or scope-based access control. The ACL model is based on resource/scope definitions mapped to permissions that evaluate if clients with specific permission assigned are allowed to access protected resources in a module. Access and ACL are performed on the obtained JWT tokens, providing, for example, a client ID and its password. Notably, other flows (e.g., implicit or access code) are also possible, following the oAuth2/OpenID Connect (OIDC) specification.

At the platform level, two types of users are identified by roles: *standard users* and *admin*. Users can create and manage applications they own. Admins have higher privileges that allow them to manage all the resources (e.g., applications or modules) in the toolkit. An application can be managed with a dedicated client having full access to all the application-specific resources (those enabled by modules at the platform level). Applications can declare their modules that will generate clients with ACL-level mapping to specific resources and scopes constrained in the application context. These clients should have minimum permissions to deliver their objective and are installed in the hardware or software implementations of the application modules in a given scenario.

4.3 Addition/Installation of a Module

Fig. 3 depicts a high-level schema of how a platform admin or a user with permission to manage platform-wide modules can add a new module.

The user registers module information to the platform module that loads the external module's API specifications reference and maps with the provided resources/scopes. The toolkit exposes an additional set of request-response APIs, protected by the provided resources/scopes and forwards the request to the external module's API. Similarly, the external module can use the generated client credentials to connect to the broker and publish or receive the topics mapping on the configured resources/scopes. The application developers can then add those resources/scopes to their client configurations to enable interaction with the external modules. The modules can be re-implemented and swapped without further intervention by reusing the same API and events signature.

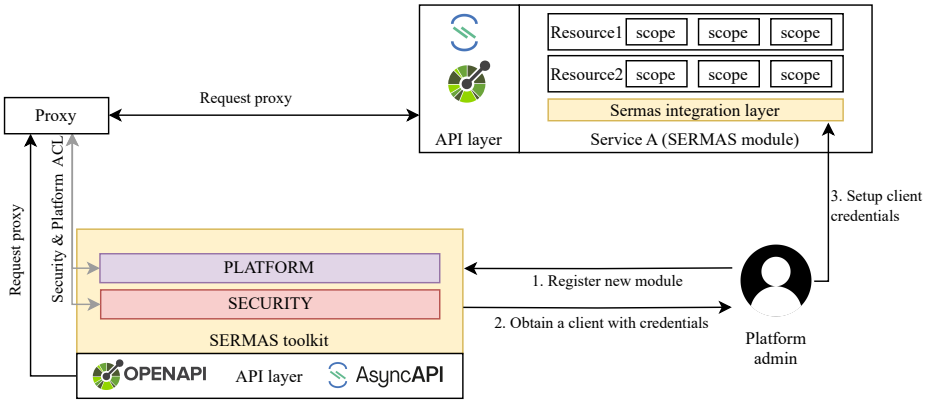


Fig. 3: Modular approach of the toolkit to add new modules.

A new module can be installed using the toolkit and the Command Line Interface (CLI)[5]. The CLI implements most of the toolkit APIs and facilitates interaction with the available services and modules.

Fig. 4 shows the command to add a module with the reference URL of the module.

```
$ ./dev-cli.sh -l debug platform module add http://172.17.0.1:3002
```

Fig. 4: SERMAS admin CLI command to add a module.

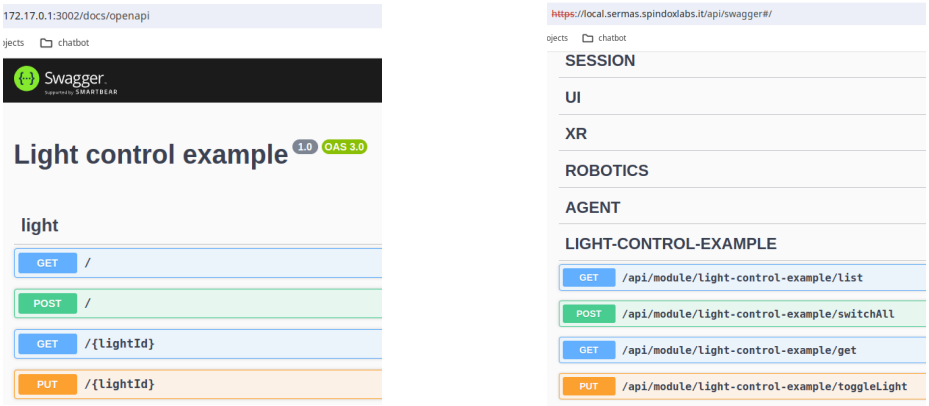
This operation will look up the '*.well-known/sermas.json*' path to obtain details about the module, such as the paths to OpenAPI and AsyncAPI specifications and additional configurations for the exposed module. After the successful execution of the command, the new module will be made available by the toolkit API (cf. Fig. 5).

Similarly, the AsyncAPI, with the supported events, is exposed by the toolkit (cf. Fig. 6). Since the broker in use is the toolkit one, the external modules are required to connect and handle the request with their client implementations.

5 Outlook

We presented our proposal of a modular architecture for XR systems to allow for their general applicability across applications. The modularity of the architecture is ensured by providing a cloud infrastructure supported by edge computing frameworks, giving the possibility to extend the system's functionalities. The

[5] https://github.com/sermas-organization/sermas-cli



(a) (b)

Fig. 5: (a) Original module specification and (b) Modules exposed by the toolkit.

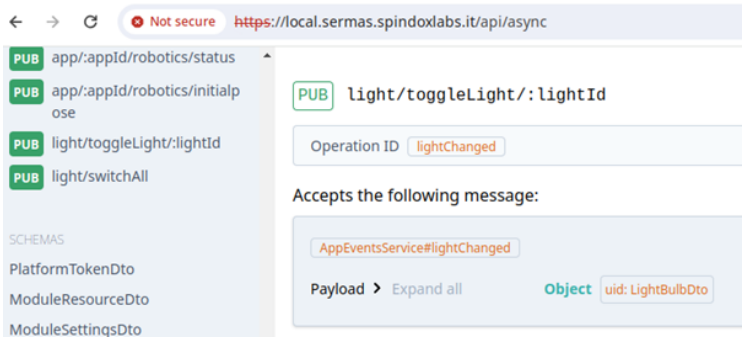


Fig. 6: Toolkit AsyncAPI with new module extension.

proposed architecture is multi-modal, incorporating verbal and non-verbal interaction, and supports both request-response and event-driven messaging models for communication among modules. We also discussed features (such as security implemented through access control) to enable the social acceptance of the resulting XR systems.

In the future, we aim to validate the proposed architecture's applicability using several case studies in different scenarios. Noteworthy among these scenarios are:

– *Remote XR-based journalism training with advanced virtual avatar:* This scenario involves deploying an XR and web-based training solution customized for journalists. This will be achieved by merging a versatile, platform-agnostic avatar (an XR component) with an LLM-driven chatbot system (an AI component), which can be used to enable tailored and practical training for diverse types of journalists in situations where they might encounter

sensitive, awkward, or dangerous scenarios. Participants will engage with a socially acceptable virtual avatar in a digital-remote location, eliminating the need for specialized end-user equipment.
- *Customer Reception Kiosk:* This application implements an XR system at conference centers, aimed at optimizing customer reception processes and elevating service standards. The system will offer various functionalities, e.g., providing general information, facilitating physical access to company premises, and offering details about conferences and meeting spaces. Emphasis will be on fostering seamless, face-to-face interactions between customers and the XR system through intuitive touch and voice-based communication.
- *Virtual assistant for Info Point and service offering:* This scenario focuses on developing an XR system tailored for conversational customer interaction. Designed to meet specific service requirements, the system will engage with customers to deliver information and services. The interaction between the XR system and the user will be customized to create user-friendly, human-like communication. The XR system will be able to move towards the customers and execute physical interaction with them.

The above scenarios will involve implementing the proposed architecture to realize XR systems suitable for such settings, checking for correspondence between the selected modules and the case-study-specific requirements, and evaluating the security and privacy of modules and their communication in compliance with the requirements. In this direction, an aspect worth exploring would be to verify that the composition of secure modules yields a secure XR system, thereby ensuring that the adopted modular approach preserves security.

References

1. Ahmed, I., Nahar, T., Urmi, S.S., Taher, K.A.: Protection of sensitive data in zero trust model. In: Proceedings of the international conference on computing advancements. pp. 1–5 (2020)
2. Andrade, T., Bastos, D.: Extended reality in iot scenarios: Concepts, applications and future trends. In: 2019 5th Experiment International Conference (exp. at'19). pp. 107–112. IEEE (2019)
3. Andrews, C., Southworth, M.K., Silva, J.N., Silva, J.R.: Extended reality in medical practice. Current treatment options in cardiovascular medicine **21**, 1–12 (2019)
4. Asyncapi: Asyncapi.com. https://www.asyncapi.com, accessed: September 2023
5. Cardenas-Robledo, L.A., Hernández-Uribe, Ó., Reta, C., Cantoral-Ceballos, J.A.: Extended reality applications in industry 4.0.-a systematic literature review. Telematics and Informatics p. 101863 (2022)
6. Çöltekin, A., Lochhead, I., Madden, M., Christophe, S., Devaux, A., Pettit, C., Lock, O., Shukla, S., Herman, L., Stachoň, Z., et al.: Extended reality in spatial sciences: A review of research challenges and future directions. ISPRS International Journal of Geo-Information **9**(7), 439 (2020)
7. Doolani, S., Wessels, C., Kanal, V., Sevastopoulos, C., Jaiswal, A., Nambiappan, H., Makedon, F.: A review of extended reality (xr) technologies for manufacturing training. Technologies **8**(4), 77 (2020)

8. Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M.: The many faces of publish/subscribe. ACM computing surveys (CSUR) **35**(2), 114–131 (2003)
9. Fernández-Caramés, T.M., Fraga-Lamas, P., Suárez-Albela, M., Vilar-Montesinos, M.: A fog computing and cloudlet based augmented reality system for the industry 4.0 shipyard. Sensors **18**(6), 1798 (2018)
10. Fielding, R.T.: Rest: architectural styles and the design of network-based software architectures. Doctoral dissertation, University of California (2000)
11. Google: Model card for flan-t5 xxl. https://huggingface.co/google/flan-t5-xxl (2023), accessed: October 2023
12. Google: Speech-to-text. https://cloud.google.com/speech-to-text (2023), accessed: October 2023
13. Hamza-Lup, F.G., Hughes, C., Rolland, J.P.: Sensors in distributed mixed reality environments. arXiv preprint arXiv:1811.11955 (2018)
14. Jones, M., Bradley, J., Sakimura, N.: Json web token (jwt). Tech. rep., Internet Engineering Task Force (IETF) (2015)
15. Justadudewhohacks: Faceapi. https://github.com/justadudewhohacks/face-api.js (2014), accessed: September 2023
16. Lee, S., Lee, G., Choi, G., Roh, B.h., Kang, J.: Integration of onem2m-based iot service platform and mixed reality device. In: 2019 IEEE International Conference on Consumer Electronics (ICCE). pp. 1–4. IEEE (2019)
17. Liu, Y., Han, T., Ma, S., Zhang, J., Yang, Y., Tian, J., He, H., Li, A., He, M., Liu, Z., Wu, Z., Zhao, L., Zhu, D., Li, X., Qiang, N., Shen, D., Liu, T., Ge, B.: Summary of chatgpt-related research and perspective towards the future of large language models. Meta-Radiology (2023)100017 (2023)
18. Ogino, T., Matsuda, Y., Kawabata, D., Yamazaki, K., Kimura, A., Shibata, F., et al.: A distributed framework for creating mobile mixed reality systems. In: 2014 IEEE 3rd Global Conference on Consumer Electronics (GCCE). pp. 327–331. IEEE (2014)
19. Openapis: Openapis.org. https://www.openapis.org, accessed: September 2023
20. Pereira, N., Rowe, A., Farb, M.W., Liang, I., Lu, E., Riebling, E.: Arena: The augmented reality edge networking architecture. In: 2021 IEEE International Symposium on Mixed and Augmented Reality (ISMAR). pp. 479–488. IEEE (2021)
21. Radford, A., Kim, J.W., Xu, T., Brockman, G., McLeavey, C., Sutskever, I.: Robust speech recognition via large-scale weak supervision. Electrical Engineering and Systems Science (2022)
22. Red Hat, I.: Keycloak. https://www.keycloak.org/ (2020), accessed: September 2023
23. Ros2: Ros - robot operating system. https://www.ros.org/ (2023), accessed: October 2023
24. Rose, S., Borchert, O., Mitchell, S., Connelly, S.: Zero trust architecture (no. nist special publication (sp) 800–207). National Institute of Standards and Technology (2020)
25. Serras, M., García-Sardiña, L., Simões, B., Álvarez, H., Arambarri, J.: Dialogue enhanced extended reality: Interactive system for the operator 4.0. Applied Sciences **10**(11), 3960 (2020)
26. Swagger Community: Swagger. https://swagger.io/ (2011), accessed: September 2023
27. Team, S.: Speechbrain. https://speechbrain.github.io/ (2022), accessed: September 2023