RESEARCH

Design, Implementation and Evaluation of Core/Periphery-based Network-oriented Mixed Reality Services

Shiori Takagi*, Shin'ichi Arakawa and Masayuki Murata

Abstract

Many new network-oriented services have been developed in recent years, and Multi-access Edge Computing (MEC) has been standardized to improve the responsiveness of services. When deploying services in a MEC environment, it is necessary to consider a service structure that can flexibly switch service behaviors to meet various user requests and that can change service behaviors according to the real-world environment at a low implementation cost. In this paper, we introduce a core/periphery structure for service components, which is known as a model for flexible behavior in biological systems, and design and implement a network-oriented mixed reality service based on this structure. We investigate what kinds of functions should be developed to accommodate user requests in conjunction with various types of devices and real-world environments in which users and devices are located. To utilize the flexibility of the core/periphery structure, we regard core functions as those whose behaviors remain unchanged even when there are changes in user requests or the environment. In contrast, peripheral functions are those whose behaviors can change under such circumstances. Experiments reveal that implementation costs are reduced while retaining increases in service response time to less than 31 ms. These results show that taking advantage of the core/periphery structure allows appropriate division of service functions and placement of functions in a MEC environment, with only small penalties on latency and at a low implementation cost.

Keywords: Core/Periphery Structure; Multi-access Edge Computing (MEC); Mixed Reality (MR); Telexistence Service; Network Robot

1 Introduction

Many new network-oriented services have arisen with development of the Internet of things (IoT), and information networks are rapidly changing. Using these new services, we can send real-world information from cameras and sensors to the cloud, or perform high-load processing such as image recognition or voice and sound recognition. For example, telexistence services using robots and Virtual Reality (VR)or Mixed Reality (MR) technologies are now being developed. The ANA Avatar project [1] investigates use of robotics and techniques for transmitting tactile sensations to develop services through which users operate avatar robots to communicate at remote places as if they were actually there.

In such applications, application-level delay is a significant factor affecting service quality. However, communication distance and load concentrations can significantly increase application-level delay in cloud computing environments [2]. Recently, multi-access edge computing (MEC) [2–4] has been standardized to mitigate increases in application-level delay for delay-sensitive services. In an edge computing environment, computing resources and storage are allocated at the network edge so that processing required by end devices is performed at closer sites. This improves the responsiveness of applications by shortening communication distances and load distributions.

Because many new network-oriented services have developed to meet various user requests, it is important to consider service designs that can accommodate as many services as possible when deploying network services in a MEC environment. However, implementation costs increase if developers must reconstruct entire services to meet different user requests or to adapt to environmental variation such as device evolution. Moreover, MEC environment resources are not necessarily the same as those in a cloud computing environment. Specifically, MEC environment resources

*Correspondence: s-takagi@ist.osaka-u.ac.jp
Graduate School of Information Science and Technology, Osaka University, 1-5 Yamadaoka, Suita, Osaka, Japan
Full list of author information is available at the end of the article

are limited by spatial restrictions, making it difficult to locate all possible services, such those on the edge that can adapt to each user request and environmental variation. It is therefore necessary to consider service structures that can change service behaviors in a flexible manner. Service function placement in MEC environments has been studied in, for example, [5] and [6], but most of them correspond to user mobility. We consider a service design where the developers can modify or add service functions in a flexible manner with less cost against changes of real environment and user requirements.

We has been investigating a core/periphery structure [7,8] that allows service components to effectively adapt to each user request and environmental variation. The core/periphery structure is a model for flexible and efficient information processing mechanisms in biological systems. Information processing units in a core/periphery structure are classified as core or periphery units. Core units are densely composed with system constraints, and process information more efficiently. Periphery units, which are connected to core units, can have various configurations, allowing them to flexibly adapt to environmental changes surrounding the system and to build flexible, efficient information processing mechanisms with the core. The advantages of a core/periphery structure for accommodating information services, represented by chains of functions, were numerically investigated in Ref. [9], with the results showing that a core/periphery structure reduces developmental costs for accommodating various kinds of information services.

Based on these advantages of a core/periphery structure, in this study we realize a service system that adapts service behaviors to various user requests, devices, and real-world environmental changes where the end devices are located. Unlike model-based evaluations, we design and implement a service based on the core/periphery structure. We then consider a shopping service using MR devices and robots. We implement this service using actual devices, and experimentally evaluate the effect of designing services based on a core/periphery structure. This paper focuses on a shopping service, but service design based on a core/periphery structure is not limited to the shopping service and can be applied to other network services.

When designing services based on a core/periphery structure, it is necessary to consider which functions should be implemented as core units and which should be implemented as periphery units. We first investigate what kinds of functions would be required in a shopping service. To utilize the flexibility of a core/periphery structure, we regard as core functions those whose behaviors remain unchanged under changes to user requests or the real-world environment, and peripheral functions as those whose behaviors can change under such circumstances. In this way, core functions allow adaptation to the emergence of new services by adding or changing some peripheral functions instead of recreating entire services. We next evaluate the design of a service based on the core/periphery structure in terms of implementation cost and service responsiveness. The results shows that as compared to a conventionally designed service, the implementation cost for adding new functions of a service design based on a core/periphery structure is reduced without increasing service responsibility. We close with a summary of the advantages of service design based on a core/periphery structure, which are not numerically verified but are experienced through the service implementation.

The remainder of this paper is organized as follows. Section 2 describes related work on services that are currently being developed or are expected to be developed in the future. Section 3 describes the services targeted in this paper and a service design based on a core/periphery structure. Section 4 describes details of the service implementation and evaluation. Section 5 describes lessons learned from service implementation based on a core/periphery structure. Finally, Section 6 describes our conclusions and future work.

2 Current and Future Network-oriented Mixed Reality Services

This section describes network-oriented services that have been developed recently or are expected to be developed in the future.

2.1 Current Services

Telexistence services have been actively developed in recent years, and momentum for their social implementation has been rising. Telexistence aims at allowing people to feel as if they are actually at a remote place. TELESAR V [10] is a telexistence master–slave system allowing users to feel present in a remote environment by transmitting not only video and audio, but also haptic sensations. ANA Avatar [1] is conceived as a new mode of instantaneous transportation allowing users to communicate and work as if actually present in remote places, using robotics and technologies for sending tactile sensations and allowing remote robot operations. ANA has begun testing via the ANA Avatar Museum, which allows users to view a remote aquarium, and ANA Avatar Fishing, through which users can remotely fish. A telexistence application using drones is also being developed [11].

2.2 Future Services

Sixth-generation (6G) networks will allow development of services using technologies that would be difficult to support over fifth-generation (5G) networks. Within ten years, current remote interaction technologies will become obsolete, and new forms of interaction such as holographic and five-sensory communication will allow immersion in remote places [12]. Tactile Internet and full-sensory digital reality can be realized by 6G networks [13]. It has also been suggested that 6G networks will further support underwater and space communications, allowing deep-sea sightseeing and space travel [13].

Application-level delay, which significantly increases in cloud computing environments with communication distances and load concentrations, will be a significant factor for service quality in these applications [2]. MEC is therefore expected to be further standardized [2–4]. In edge computing, computing resources and storage are allocated at the network edge, so that processing required by end devices is performed at closer sites. This improves the responsiveness of applications by shortening communication distances and optimizing load distributions. Our research group demonstrated that MEC environments improve the service quality of network-oriented MR services [14]. The ETSI Industry Specification Group [15] suggests video content delivery, video stream analysis, and Augmented Reality (AR) as key use cases for MEC, and suggests guidelines for software developers.

Current mainstream services include audio and video transmission, but realizing transmission of information for the five senses will require construction of service systems that can handle multiple inputs and outputs. In this paper, we propose guidelines for service function placement in a core/periphery structure, a biological model for flexibly and efficiently processing information.

3 Service Design Based on a Core/Periphery Structure

This section describes a service design based on a core/periphery structure.

3.1 Network-oriented d Reality Services under Study

We consider a shopping service using MR and robots. Robots are placed in an actual store to allow users to shop from home as if they were actually there. Robots provide a video feed while moving about the store under user operations. Real-world information on the store side is added to videos delivered to users. Users can move robots with gamepads, gestures, or gaze. Figure 1 shows an overview of the shopping service and its functions.

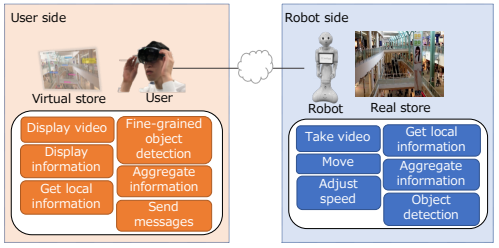


Figure 1: **The presumed service and its functions.**

Robot-side applications provide functions for moving, taking video, processing images, collecting and aggregating information around the robot, and adjusting movement speed so as not to collide with people or objects. User-side applications provide functions for displaying video, sending instructions to the robot, collecting and aggregating information around users, and detecting objects at a user-defined granularity.

3.2 Service Decomposition Based on a Core/Periphery Structure

Functions for our network-oriented MR services are categorized into those related to video transfer and those related to robot operations. This section describes functions and processing for video transfer and robot operations. We then decompose our service into core and peripheral functions based on a core/periphery structure.

3.2.1 Video Transfer

Functions for video transfer provide video capture and output, perform object detection, and distribute video to users. For video transfer, we consider two service scenarios depending on user requests, devices, and real-world environments. First scenario is a video rate adjustment. When a new camera or device is developed, the performance of the camera capturing the video may not match the performance of users' devices. In that case, the function to change the rate and resolution of the video is needed. Users can also change the video resolution and rate depend on the network environment they are placed in. Second scenario is the selection of object detection methods. Users switch object detection methods appropriate to the location of the robot or information the users want. For example, in a shopping service, when a robot is moving through the halls of a shopping mall, users may select fast but coarse-grained method, and when users want to know the detailed classification of a product, they select slow but fine-grained method. Also, when new object detection methods are developed, service

Table 1: Service functions for video transfer.

Function	User requests	Processing
Video I/O	Real time video High-resolution video	Change rates Change resoluti
Object detection	Fast but coarse-grained Slow but fine-grained	Adopt appropriate m
Video distribution	To one user	Use UDP
	On a large scale	Use HTTP

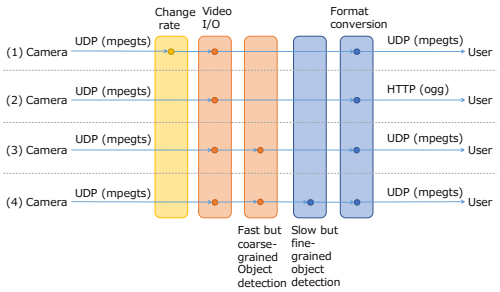


Figure 2: **Examples of processing in video transfer.** (1) Video providers change video frame or bit rates. (2) Video providers distribute video to multiple users. (3) Object detection with only a standard part is executed. (4) Object detection with a new part is executed.

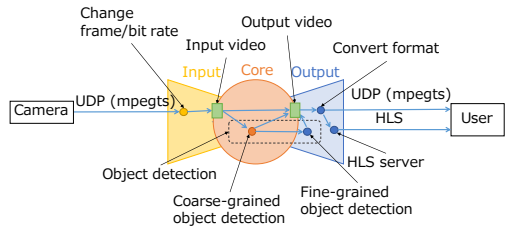


Figure 3: **Video transfer based on a core/periphery structure.**

developers implement additional functions to support them. Third scenario is the selection of video distribution protocols. A video transfer service which requires real-time video transmission from robots to users may use UDP and mpeg-ts, and other video transfer service may use HTTP for transmitting video to multiple users. To realize these service scenarios, our service provides functions for video I/O, object detection, and video distribution. Table 1 summarizes service functions and processing to adapt to the users' request for video transfer.

Then, our next step is to determine which are core functions when adopting a core/periphery structure for the service. We decompose the service into functions, and consider which of those described in Section 3.1 are core functions and which are peripheral functions, based on the concept of a core/periphery structure in which core functions processes information more efficiently, while periphery functions have various configurations for flexibly adapting to environmental changes.

Figure 2 shows examples of video transfer processing. When video providers want to change the video frame or bit rates to adapt to the amount of available resources, the video is processed before input. Users too can change the frame or bit rate. In that case, video is processed after output. Protocols and the video format can be changed at the video providers' request. For example, video providers use the UDP transfer protocol to send video to a single user, and HTTP otherwise. When users want to know what is in the video, object detection is executed. There are various object detection methods, such as YOLOv3 [16], which is fast and widely used, and Mask R-CNN [17], which provides more detail but is slow. Users can adopt their preferred method. Orange functions in Fig. 2 are common core functions, while light orange and blue functions are peripheral functions.

Figure 3 shows the core/periphery structure for video transfer, with orange fields indicating core functions, light orange fields indicating camera-side periphery functions, and blue fields indicating user-side periphery functions. Video is sent from the camera, whose frame and bit rates are adjusted based on provider requests as a peripheral function. The video then passes through core functions, including those for inputting video, outputting video, and the standard part of object detection. Finally, the video format and distribution protocol are selected and sent to users. By utilizing the flexibility of a core/periphery structure, all developers have to do is remake or add peripheral functions for adapting to different user requests, changes in the real-world environment where devices are placed, or device evolution.

3.2.2 Robot Operation

Robot operations provide functions for recognizing user actions, sending messages from users, accessing APIs, adjusting robot speeds to avoid obstacles, and collecting and aggregating information obtained from robots. For robot operations, we consider three service scenarios depending on user requests, devices, and real-world environments. First scenario is the selection of command interfaces based on the users' devices, which includes either or all of gamepads, gestures, and gaze. Users select how to operate remote devices depending on the users' device type and its specification. In the future, as new command interfaces or devices are developed, new functions to use the new devices

Table 2: Variations of service for robot operations.

Function	User requests/ Real-world environments	Variations of service
Command interface	Use gestures Use gamepad Use gaze	Change interface
Device control	Operate robots Operate drones	Switch/Add APIs
Adjust robot speed	Some obstacles Slippy	Move robot slowly
	No obstacles	Move robot speedily

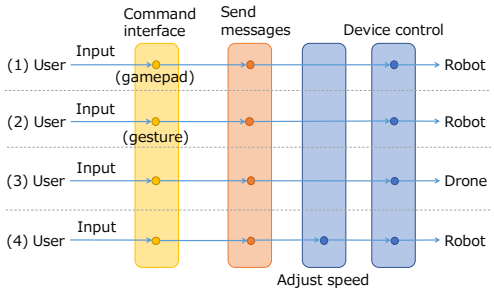


Figure 4: **Examples of processing for robot operations.**(1) User operates a robot with gamepad. (2) User operates a robot with gestures. (3) User operates a drone. (4) Robot speed adjusted based on the environment.

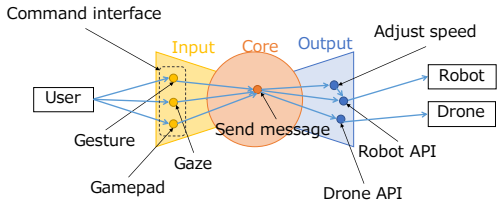


Figure 5: **Robot operation based on a core/periphery structure.**

are developed and provided to users. Second scenario is the selection of a remote device to operate. Users select the remote devices e.g. robots and drones to operate based on the remote environment or users' requests. The APIs used in the service are switched accordingly. When new remote devices are developed, users can use the new remote devices. Third scenario is adapting to changes in the real environment in which the robot is located. For example, when the robot is located in a crowded area, it moves slower, and when it is in a large area, it moves faster. To realize the service scenarios, our service provides functions for command interface, object device control, and adjustment of robot speed. Table 2 summarizes variation of service on different user requests/real-world environments for requests for robot operations.

We decompose the service into functions and determined core functions as in Section 3.2.1. Figure 4 shows examples of processing for robot operations. When users operate a robot with gamepads or gestures, their device recognizes instructions and send messages based on the selected method. When users operate different devices such drones, service behavior after receiving user messages will change to access the robot or drone's API. Furthermore, robot speeds are adjusted based on the surrounding environment. When there are no obstacles or crowds, users can speedily move robots. Otherwise, robots slow down to avoid collisions.

The function for send messages in robot operations is a common function, and therefore should be divided as a core function, rather than the whole service being performed as an all-in-one function. Figure 5 shows the core/periphery structure for robot operations. Functions for sending instruction messages from users and aggregating information obtained from robots are common, so they are core functions. Functions for adapting to user requests and changes in the real-world environment, such as how to input user instructions, are peripheral functions. Functions for accessing robot APIs, collecting information such as the current robot position and adjusting movement speed are peripheral functions, because they change according to device type and real-world environment. Flexibility of the core/periphery structure allows developers to simply remake or add peripheral functions to adapt to varying user requests, environmental changes, and device evolution.

3.3 Service Scenarios

We expect designing services based on the proposed core/periphery structure to reduce implementation costs, because developers need only add peripheral functions to implement additional services. Ideally, we would implement all service functions and present their implementation costs. In practice, however, time limitations make it difficult to implement and evaluate all service scenarios. In this study, we implement some of the service scenarios described in Section 3.2 to evaluate reductions in implementation cost and penalties for application-level delay.

We prepare two service scenarios for implementation. In the first, we modify robot behavior according to its real-world environment. This scenario realizes communication between robots' peripheral functions for adjusting speed and core functions related to robots, object detection, and messaging. In the second scenario,

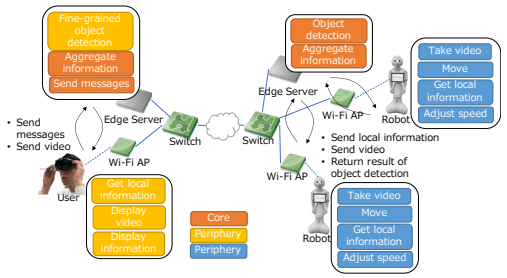


Figure 6: **Service scenario.**

we modify behavior of a user application based on the user's real-world environment. This scenario realizes communication between user-side peripheral functions for displaying information and core functions related to users, information aggregation, and messaging.

3.3.1 Behavior Based on the Real-world Robot Environment

The following describes a scenario in which robots modify their behavior based their real-world environment. Functions for robot operation and core/periphery functions are as follows:

- Core: Functions for transmitting instructions from the user to the robot and functions for object detection.
- Periphery: Functions for obtaining information near the robot, adjusting the robot movement speed, and aggregating information sent from multiple robots.

Figure 6 shows this scenario. There are users with MR headsets, robots, cameras, and edge servers on robot side. Orange functions are core functions. Blue functions are peripheral functions on robot side, and light orange functions are peripheral functions on user side. Users send instructions to robots, moving their bodies and heads by gamepads, gestures, and gaze. Robots can detect nearby obstacles and stop using sensors. Video captured by robot-mounted cameras are sent to the edge servers, which perform object detection to recognize objects and persons around the robots. Object detection, a core function, needs to be performed in real time and requires powerful servers. These functions should thus be deployed on edge servers, not on end devices. The results of object detection are returned to robots. For example, when robots know that there are many people around them, they can reduce speed to avoid collisions. Moreover, information from robots can be aggregated on edge servers and shared with other robots for collision avoidance and the like.

3.3.2 Behavior Based on the Real-world User Environment

The following describes a scenario in which behavior is based on the real-world user environment. Core/periphery functions are as follows:

- Core: Functions for sending user instructions to robots and for aggregating information from multiple robots.
- Periphery: Functions for displaying video, detailed object information, and information about each robot.

Figure 6 shows this scenario. As a core function, store and robot information such as product information or communication status is collected at user-side edge severs. Users select which robot to operate only by communicating with an edge server while viewing aggregated information about stores and robots. Video sent from cameras is roughly classified by object type on robot-side edge servers. These functions perform real-time image processing and information aggregation, and thus are inappropriate for execution on end devices. To improve responsiveness, core functions should be performed on edge servers instead of the cloud. Then, detailed object detection is performed as a peripheral function on a user-side edge server. User devices collect personal information such as user tastes, what the user already owns, and purchase history, and this information is aggregated on an edge server. Using this personal information, the system can display content most appropriate for the user. For example, the application can recommend commodities based on previous frequent purchases, or can warn users of impending expiration dates for food.

4 Implementation and Evaluation of a Service Based on a Core/Periphery Structure

This section describes implementation details and evaluates the service based on Section 3.3.

4.1 Implementation of a Service Based on a Core/Periphery Structure

4.1.1 Video Transfer

Video from cameras is sent to a robot-side edge server. Video is captured using OpenCV [18], then object detection is performed using a PyTorch implementation of YOLO v3 [16]. For video processing, mask R-CNN (Region-based Convolutional Neural Networks) [17], an algorithm that surrounds detected objects with a rectangle and recognizes the object type for each pixel and colors it accordingly, can be used. The processed video is transmitted via UDP using ffmpeg [19] to HoloLens [20], an MR headset worn by users, for

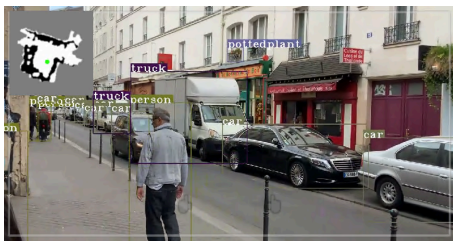


Figure 7: **Screenshot of the HoloLens application.**

display. HoloLens is a standalone head-mounted computer made by Microsoft that displays holograms and recognizes user gaze and gestures to provide a MR experience.

4.1.2 Robot Operation

HoloLens controller information is transmitted via Message Queuing Telemetry Transport (MQTT), a publish/subscribe-type protocol developed for frequent message exchange between IoT devices. Users use an Xbox controller that can connect to HoloLens. We develop an MQTT messaging system on a user-side edge server with mosquitto [21], an open-source message broker, and Node-RED [22], a programming tool for event-driven applications. The MQTT broker receives controller commands via HoloLens and sends them to a program running on the robot. The robot is a Pepper [23] running a program developed using the programming tool Choregraphe. This program converts messages from the MQTT broker to the Pepper API.

Figure 7 shows a screenshot of the HoloLens application. Users can see video with the object detection results and a map made by Pepper displayed at the top left. The green dot represents Pepper's position.

4.2 Evaluation Metrics and Measurement

This subsection describes the evaluation metrics, namely implementation cost and service responsiveness, and how we measure those metrics.

4.2.1 Implementation Cost

Using the implemented service, we show that adopting a core/periphery structure lowers implementation costs.

We evaluate the number of lines of source code as the implementation cost, comparing source code size when the service is designed based on a core/periphery structure with the case where the service is not designed based on a core/periphery structure and implemented on an end device. Number of lines of source code is a numerically representable metric indicating implementation complexity. While knowledge and preparation of the development environment is also part of the implementation cost, such factors are difficult to numerically evaluate. Section 5 describes these and other lessons regarding service implementation.

4.2.2 Service Responsiveness

Because sending user instructions via an edge server can increase application-level delay compared with the case of directly sending instructions to robots, we measure and evaluate application-level delay as a penalty for using edge servers.

We measure times from when the HoloLens application publishes a message to return of robot sensor data to HoloLens directly and through the MEC environment. Then, we compare these times to evaluate the effect of allocating core functions on an edge server. We regularly sent messages about 20 times from the HoloLens application and saved each message return time as $t_1, t_2, \ldots, t_{20}$. We also record times when Pepper returned sensor data as $t'_1, t'_2, \ldots, t'_{20}$ in the HoloLens application. Then, we calculate the average of $t'_1 - t_1, t'_2 - t_2, \ldots, t'_{20} - t_{20}$ as the application-level delay.

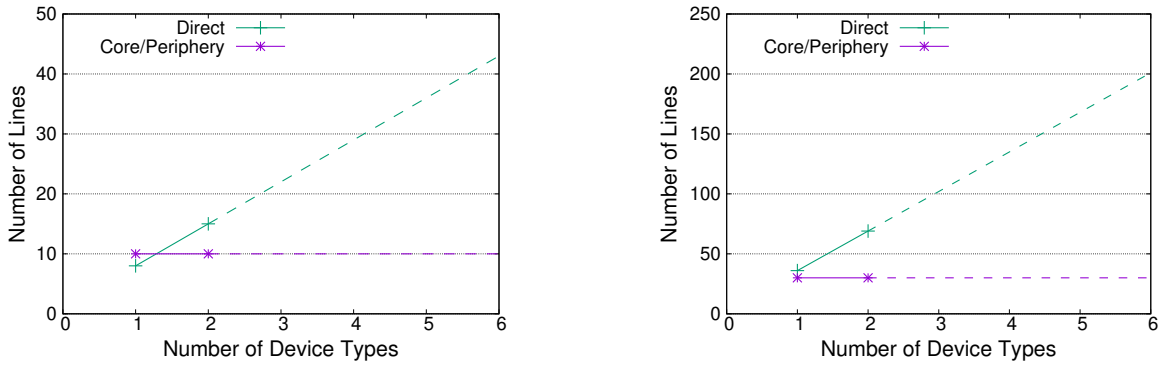
Application-level delay is a one-way delay. However, since there are different system clocks between HoloLens and Pepper, accurate comparison of one-way delay is difficult. We therefore measure round-trip delay.

We construct a MEC environment using OpenStack located in Osaka.

4.3 Results

4.3.1 Implementation Cost

Figure 8 shows the relation between the number of device types at remote sites and the number of lines of source code for the connection establishment part (Fig. 8(a)) and for the messaging part (Fig. 8(b)). We omit the complete source code due to space limitations, but it is available at our GitHub repository [24]. The "Direct" represents the design not based on a core/periphery structure, and Core/Periphery represents the design based on a core/periphery structure. Solid lines in the figure represent the number of lines for two robot types, a Pepper and a presumed robot, and dashed lines represent the number of lines when using more than two robot types. We have not implemented the application with more than two robots, but predict that the number of lines will linearly increase because



(a) **Connection establishment part** (b) **Messaging part**

Figure 8: **Number of lines of source code.**

applications not based on a core/periphery structure require source code for establishing connections and messaging for each device API, resulting in a constant additional number of IF statements for each.

Figure 8 shows the effect of a design based on a core/periphery structure increases as the number of device types increases. Note that when a single type of device or a single type of service is implemented, the design based on a core/periphery structure is less effective. When additional type of remote devices are deployed for the service, developers need to prepare service functions, i.e., write code, to establish connections and operations for the remote devices. Writing the code is necessary for both the design based on a core/periphery structure and the design not based on a core/periphery structure. However, in the application designed not based on the core/periphery structure, the amount of source code increases linearly against the increase of remote device since the service functions are dependent each other. In the application designed based on the core/periphery structure, developers can reuse these functions as core functions, and the amount of code is constant or increases marginally.

We considered both variation of devices at remote sites and variation of user-side devices. Both in services based on a core/periphery structure and those not based on this structure, developers must add source code for obtaining controller information, because this is a peripheral function. However, increasing the number of controller types also increases the number of source code parts to be added on the remote side, an effect that is mitigated by designing services based on a core/periphery structure. Therefore, developers can implement applications more easily by adopting a core/periphery structure.

Table 3: Results of experiments measuring penalty of using an edge server.

	Direct	Core on edge
Average [ms]	21	52
Max [ms]	24	263
Min [ms]	0	18
Ping RTT [ms]	-	1

4.3.2 Service Responsiveness

Table 3 shows average, maximum, and minimum values for application-level delay, along with ping round-trip time (RTT) when the HoloLens application directly connects to the Pepper and when the HoloLens application connects to Pepper via edge servers. Figure 9 shows the average application-level delay in these three cases. The results show that application-level delay when using MQTT on an edge server is about 31 ms. A 31 ms delay is tolerable because humans' reaction time is around 190 ms for light stimuli [25–29]. In combination with the results presented in Section 4.3.1, therefore, a service design based on a core/periphery structure reduces implementation costs without significantly deteriorating service responsiveness.

5 Lessons from Service Implementation

This section presents lessons learned from service implementation based on a core/periphery structure, including factors that cannot be numerically represented.

First, developers do not need to consider device APIs and specifications. When a service is not based on a core/periphery structure, functions are not divided and user-side devices directly establish connections with remote devices. Developers need to know the APIs of many remote devices to write many parts of source code, including how to establish connections,

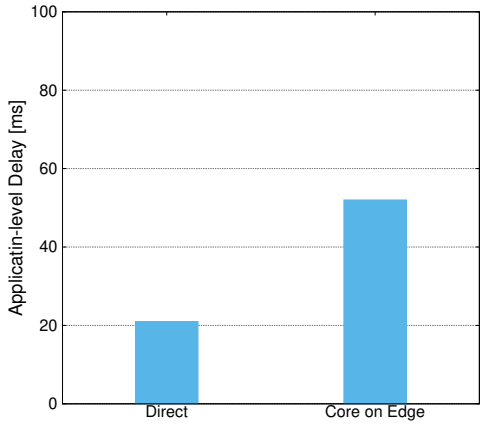


Figure 9: **Application-level delay.**

how to move devices at remote sites, and the parameter settings such as the sensitivity to user operations, which are depend on the speed and other features of the remote device. By dividing functions based on a core/periphery structure, user-side developers need to know only user-side device APIs, and do not need to consider remote device APIs.

Furthermore, adopting a core/periphery structure absorbs differences in development environments. We implement the service using Unity. A 32-bit version of Unity is required to directly use the Pepper API from a user-side application, but 32-bit versions of Unity are no longer being developed. To develop for Pepper, therefore, we must use an old version of Unity and modify the source code as appropriate. When developing for devices that require an old development environment and those that require new ones, we need to know the APIs provided by both. When designing services based on a core/periphery structure, however, developers need to prepare an environment for the user-side device only, because core functions absorb differences in device specifications.

Second, we consider the implementation cost for sharing information among robots. A non-core/periphery service structure does not have edge servers. To share information such as positions, robots must establish connections with each other. Therefore, each time a new robot appears, developers must modify source code to allow other robots to connect with the new one. By designing services based on a core/periphery structure, since robots send information to only edge servers, where that information is aggregated, source code does not need to be changed even when new robots appear.

Third, we derive guidelines for service function placement in a core/periphery structure. Taking advantage of a core/periphery structure allows appropriate division of service functions and deployment of those functions to different servers or devices. If no functions are divided and deployed in the cloud or on end devices, new services must be entirely recreated to adapt to various user requests or device evolution. Furthermore, allocating core functions on edge servers and peripheral functions on end devices is the most effective in terms of service responsiveness and implementation cost, because it is possible to form feedback loops by short-distance communication between end devices and edge servers located near those devices and to adapt to real-world environmental changes.

6 Conclusion

In this paper, we introduce a core/periphery structure for service components, which is a known model for flexible behavior in biological systems, and we designed and implement a network-oriented MR service based on this structure.

In the presumed service, we investigate what kinds of functions can be developed for user requests, real-world environments where devices are located, and the development of new devices. To utilize the flexibility of the core/periphery structure, we regard core functions as those with unchanging behaviors even when there are changes in user requests or the real-world environments, and peripheral functions as those whose behaviors can change under such circumstances. We also implement applications based on the scenarios described in Section 3.3 and evaluate the effects of a design based on a core/periphery structure under an experimental laboratory environment. These experiments showed that application-level delay due to MQTT on an edge server is about 31 ms. Taking advantage of the core/periphery structure allowed us to appropriately divide service functions and locate functions in a MEC environment, thus reducing implementation cost for adding new functions with little penalty.

In future work, we will evaluate implementation costs for object detection and feedback to robots, and for sharing information among robots. There is also a need for implementation and evaluation of the service using robots other than Pepper. Service design based on a core/periphery structure is more efficient when there are various devices, but in this paper we implement and evaluate a service using only one kind of headset and robot.

Availability of data and materials

The datasets generated and/or analysed during the current study are available in the GitHub repository, https://github.com/s-takagi15/HoloLens-Peppercontroller.

Competing interests

The authors declare that they have no competing interests.

Funding
This research was funded by the National Institute of Information and Communications Technology (NICT) in Japan.

Author information
Affiliations
Graduate School of Information Science and Technology, Osaka University, 1-5 Yamadaoka, Suita, Osaka, 565-0871 Japan.

Contributions
MM have contributed to the conception of this study. ST and SA designed our application. ST performed the experiments. ST is the main contributor and writer of this manuscript. All of authors have read and approved the final manuscript.

Acknowledgements
This work was partly supported by the National Institute of Information and Communications Technology (NICT) in Japan. . . .

References

1. "ANA Avatar," https://ana-avatar.com.
2. A. C. Baktir, A. Ozgovde, and C. Ersoy, "How can edge computing benefit from software-defined networking: A survey, use cases, and future directions," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2359–2391, Jun. 2017.
3. Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing a key technology towards 5G," *ETSI White Paper*, no. 11, Sep. 2015.
4. T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1657–1681, May 2017.
5. T. Ouyang, Z. Zhou, and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, 2018, pp. 1–10.
6. G. Liu, J. Wang, Y. Tian, Z. Yang, and Z. Wu, "Mobility-aware dynamic service placement for edge computing," *EAI Endorsed Transactions on Internet of Things*, vol. 5, p. 163922, 07 2018.
7. P. Csermely, A. London, L.-Y. Wu, and B. Uzzi, "Structure and dynamics of core-periphery networks," *Journal of Complex Networks*, vol. 1, pp. 93–123, Sep. 2013.
8. V. Miele, R. Ramos-Jiliberto, and D. P. Vázquez, "Core–periphery dynamics in a plant–pollinator network," *bioRxiv*, Jul. 2019.
9. Y. Tsukui, "On network function virtualization for dynamically changing service requests based on a core/periphery structure," Master's thesis, Graduate School of Information Science and Technology, Osaka University, Feb. 2020.
10. S. Tachi, "Telexistence: Enabling humans to be virtually ubiquitous," *IEEE Computer Graphics and Applications*, vol. 36, no. 1, pp. 8–14, Jan. 2016.
11. X. Xia, C. Pun, D. Zhang, Y. Yang, H. Lu, H. Gao, and F. Xu, "A 6-DOF telexistence drone controlled by a head mounted display," in *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, Mar. 2019, pp. 1241–1242.
12. E. C. Strinati, S. Barbarossa, J. L. Gonzalez-Jimenez, D. Ktenas, N. Cassiau, L. Maret, and C. Dehos, "6G: The next frontier: From holographic messaging to artificial intelligence using subterahertz and visible light communication," *IEEE Vehicular Technology Magazine*, vol. 14, no. 3, pp. 42–50, Sep. 2019.
13. Z. Zhang, Y. Xiao, Z. Ma, M. Xiao, Z. Ding, X. Lei, G. K. Karagiannidis, and P. Fan, "6G wireless networks: Vision, requirements, architecture, and key technologies," *IEEE Vehicular Technology Magazine*, vol. 14, no. 3, pp. 28–41, Sep. 2019.
14. S. Takagi, J. Kaneda, S. Arakawa, and M. Murata, "An improvement of service qualities by edge computing in network-oriented mixed reality application," in *2019 6th International Conference on Control, Decision and Information Technologies (CoDIT)*, Apr. 2019, pp. 773–778.
15. D. Sabella, V. Sukhomlinov, L. Trang, S. Kekki, P. Paglierani, R. Rossbach, X. Li, Y. Fang, D. Druta, F. Giust, L. Cominardi, W. Featherstone, B. Pike, and S. Hadad, "Developing software for multi-access edge computing," *ETSI White Paper*, no. 20, Feb. 2019.
16. J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," *CoRR*, vol. abs/1804.02767, Apr. 2018. [Online]. Available: http://arxiv.org/abs/1804.02767
17. K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proceedings of 2017 IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017, pp. 2980–2988.
18. "OpenCV," https://opencv.org.
19. "FFmpeg," https://www.ffmpeg.org/.
20. "Microsoft HoloLens," https://www.microsoft.com/ja-jp/hololens.
21. "Eclipse Mosquitto," https://mosquitto.org.
22. "Node-RED," https://nodered.org.
23. "Pepper the humanoid robot - SoftBank Robotics," https://www.softbankrobotics.com/emea/en/pepper.
24. "Hololens-peppercontroller," https://github.com/s-takagi15/HoloLens-Peppercontroller.
25. F. Galton, "On instruments for (1) testing perception of differences of tint and for (2) determining reaction time." *Journal of the Anthropological Institute*, vol. 19.
26. K. von Fieandt, A. Huhtala, P. Kullberg, and K. Saarl, "Personal tempo and phenomenal time at different age levels," *Reports from the Psychological Institute*, vol. 2.
27. A. T. Welford, "Motor performance. in j. e. birren and k. w. schaie (eds.), handbook of the psychology of aging," *Van Nostrand Reinhold, New York*.
28. ——, "Choice reaction time: Basic concepts." *A. T. Welford (Ed.), Reaction Times. Academic Press, New York*.
29. J. T. Brebner and A. T. Welford, "Introduction: an historical background sketch," *A. T. Welford (Ed.), Reaction Times. Academic Press, New York*.