Design and Implementation of Agent APIs for Large-scale Social VR Platforms

Ryutaro Kurai*
Cluster, Inc.
Nara Institute of Science and Technology

Takefumi Hiraki†
Cluster Metaverse Lab

Yuichi Hiroi‡
Cluster Metaverse Lab

Yutaro Hirao§
Nara Institute of Science and Technology

Monica Perusquia-Hernandez¶
Nara Institute of Science and Technology

Hideaki Uchiyama‖
Nara Institute of Science and Technology

Kiyoshi Kiyokawa**
Nara Institute of Science and Technology

Figure 1: Conversation between an agent and a user at cluster [2], a social VR already in operation. In this situation, three hunmans are controlling the humanoid avatar and one agent is controlling the boxy avatar. The female avatar on the far right is asking the box y avatar to translate from Italian to Japanese. The boxy avatar is connected to OpenAI's GPT-4, which performs the translation task. Using the API proposed in this paper, we can implement the above agent in a few lines of Python script.

Abstract

Implementing an autonomous agent on a social VR platform where many users share space requires diverse information. In particular, it is required to recognize the distance from other users, their orientation toward each other, the avatar's pose, and text and voice messages, and to behave accordingly. This paper proposes an API to obtain the above information on "Cluster," a multi-device social VR platform in operation, and an agent that uses the API. We have implemented this API using a network proxy. The agent using this API can connect to ChatGPT [7] and have a conversation in real time. We measured the latency required for the conversation and confirmed that the response time was about 1 second.

Index Terms: Human-centered computing—Human computer interaction (HCI)—Interaction paradigms—Virtual reality; Software and its engineering—Software organization and properties—Contextual software domains—Virtual worlds software

*e-mail: r.kurai@cluster.mu
†e-mail: t.hiraki@cluster.mu
‡e-mail: y.hiroi@cluster.mu
§e-mail: yutaro.hirao@is.naist.jp
¶e-mail: m.perusquia@is.naist.jp
‖e-mail: hideaki.uchiyama@is.naist.jp
**e-mail: kiyo@is.naist.jp

1 Introduction

Social VR platforms have become central to the creation of virtual communities. They attract large numbers of users who engage in deep communication and interactive gaming experiences. These platforms are redefining how we connect and play in VR and shaping a new frontier for human interaction and entertainment. If bot agents that behave indistinguishably from humans in such VR space can be realized, realism and immersion in VR space will be improved.

For example, in entertainment, the quality of the experience would be improved by allowing people to enjoy interacting with realistic characters in VR environments [3]. In education and training, this approach will allow for more realistic practice, particularly in developing interpersonal skills [5]. Furthermore, in psychological support and therapy, a bot agent that enables realistic interaction could prove to be highly beneficial [4].

The construction of a VR experimental environment is one of the scopes in the above realization of bot agents inseparable from humans in the VR space. The development and widespread use of game engines such as Unity and Unreal Engine and head-mounted displays (HMDs) such as HTC Vive and Meta Quest have made designing and implementing experiences in VR environments easier. Therefore, if it is possible to activate the agents mentioned above

in a VR environment with many participants, it can be used as a promising field of experimentation for developers and researchers. However, there are technical difficulties in creating a VR environment in which many people can participate, such as synchronizing information between users and difficulties in securing the absolute number of users to participate in the first place.

Some social VR platforms solve information synchronization problems. Many people are already interacting with such services. VRChat is a well-known example. VRChat has about 100 million monthly active users [9], has few restrictions on the representation of avatars and the structure of the virtual space, and is a platform that we can use as an experimental field. However, VRChat does not provide Application Programming Interfaces (APIs) for communicating information in the virtual space to agents or controlling avatars outside VRChat. To the best of our knowledge, such APIs have not been made available on any social VR platform other than VRChat.

The popular game Minecraft [6] provides such APIs by a third party [8]. Voyager [10] is an agent that runs on top of Minecraft and connects the game character to the Large Language Model (LLM) to control the character. However, Minecraft is a specific game title, and the avatars that run on it have limitations, such as few types of avatars, they can express few poses, and we cannot use voice calls. In contrast, social VR platforms allow users to freely select and use uploaded avatars with fewer restrictions and high expressive power in the virtual space.

To address this problem, we propose an API that enables agents to operate on the Cluster [2], a social VR platform where many people can participate, and avatars have high expressive power, e.g., appearance, gesture, and voice. Using the proposed agent API, developers can implement agents with minimal prerequisite knowledge of HTTP communication. In addition, we prepare agent software that can be called and controlled from our API and realize avatar operation. By connecting to other web services via this agent software, it is possible to make the agent behave in various ways.

We designed and implemented the proposed agent API and agent software and confirmed their operation. We show the actual operation of the agent on the Cluster in Fig. 1. We also implemented an agent connected to ChatGPT [7] as an application of the agent API and confirmed the feasibility of a conversational agent on a social VR platform.

2 METHODOLOGY

In this study, we propose an agent API that operates on a social VR platform where multiple people can share the same VR space and communicate in a VR environment. As a social VR platform, we use Cluster, which Cluster Inc. operates. It have been downloaded more than 1 million times and the total number of users has exceeded 8 million. It is the largest social VR platform in Japan.

This section describes the communication required to realize an agent operating in a VR environment. Communication is necessary to convey information in the virtual space to the agent and to give instructions from the agent to the avatar in the virtual space.

2.1 Terminology

First, we explain the terminology used in this paper.

2.1.1 User

We define a user as a person who uses Cluster. Users have a unique ID and can communicate with other users in the VR space using an avatar of their choice.

2.1.2 Cluster Server

A Cluster server is the software that shares avatar motion information between users in the VR room. It runs on a computer managed by the service provider (Cluster, Inc.). Its primary role is to receive avatar movements through communication from Cluster clients and share them with other Cluster clients.

2.1.3 Cluster Client

A Cluster client is the software that runs the Cluster. It can run in multiple environments, including Windows, OS X, Android, and iOS. This paper is intended for Windows and OS X clients. The Cluster client receives keyboard and mouse input from the user and expresses the input as avatar movements within the Cluster space. Users sent the represented motion to the Cluster server to share with other users.

2.1.4 Comment

Text can be shared within the VR space by input from the keyboard. Users can share the text with all users in the same space as fig. 1. Cluster provides this function to users as a method of verbal communication.

2.1.5 Emote

A communication feature that displays the user's emotion as an emoticon. By selecting an emoticon with the mouse, the avatar will pose the same as the emoticon or display a heart icon directly above the avatar.

2.1.6 Agent API Module

The Agent API module proposed in this paper is added to the Cluster client, which intervenes in the communication between the Cluster client and the Cluster server and provides the API required to create the agent software.

2.1.7 Agent Software

Agent software denotes software that implements agent movements using the API provided by the agent API module. Specifically, it is software that obtains the movements of other users from the API module and directs the response by the avatar. We use HTTP to communicate from the agent API module to the agent software. Therefore, we implement the agent software as an HTTP server and HTTP client.

2.2 Implementation

2.2.1 Monitoring of Communication by the Agent API Module

First, we consider the pathways that convey information to the agent in the virtual space. We show a schematic diagram of an agent API operating in a VR environment in Fig. 2. Information in the virtual space includes the avatar's position, posture, comments, voice, etc., of users simultaneously present in the same space. Typically, when using Cluster, all of this information is represented as rendered media, such as virtual scenes, UIs, and audio. However, when considering agent control, this information should be available as numerical data that can be directly analyzed, e.g., spatial coordinates and audio signals.

Therefore, we have developed an API module that provides this information in space in a desirable form to the agent. As fig. 2a shows, client and server are normally connected directly. This module exists between the Cluster client and the Cluster server in Fig. 2b and runs on the computer on which the Cluster client runs. This module monitors the communication between the Cluster client and the Cluster server. Also, it provides the agent software with the information the agent needs in an easy-to-use form.

The Cluster client and server are constantly adding and changing functionality daily. Therefore, this API module is separated from the client and server and monitors their communication. If we tightly couple a module to the Cluster client or Cluster server implementation, the module must be updated daily to keep up with the changes.

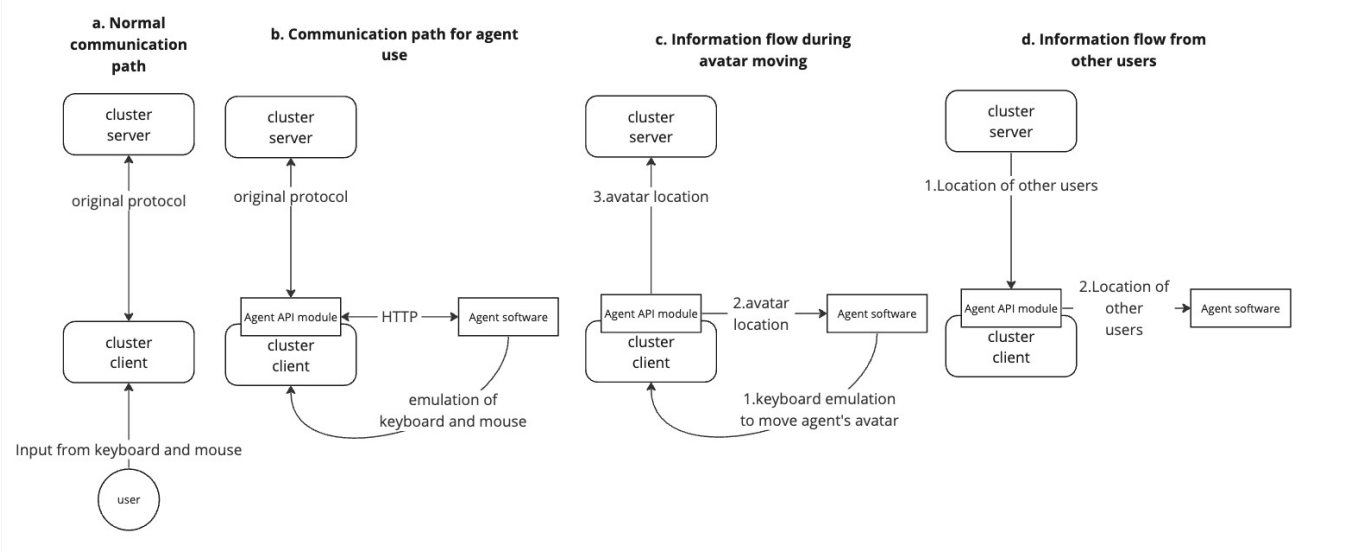

Figure 2: Schematic diagram of communication when using the agent API proposed in Cluster. Figure a illustrates the communication method between client and server in a normal cluster. Figure b illustrates the communication method when Agent Software calls Agent API Module, and figures c and d illustrate the communication method of Avatar's location from Agent API Module to Agent Software.

Changes in the communication protocols between server–clients are relatively infrequent compared to changes in the main server and client units. Therefore, it is easy for the module to track changes in the Cluster.

In addition, since development and updates can be performed independently of the development and release of Cluster, there is less conflict between service development and research activities.

We will describe the data flow through this agent API module using location information as an example. This flow is divided into three parts: (i) Avatar operation through Cluster client, (ii) Monitoring communication from the Cluster client to the server, (iii) Monitoring communication from the Cluster server to the client. Note that other types of data, such as voice and avatar posture, can also be updated by the same procedure.

Avatar operation through Cluster client In this system configuration, the agent software operates the avatar via the Cluster client (Fig. 2c-1). As the same software as the Cluster client used by ordinary users, the agent software can only manipulate the mouse and keyboard controls. Therefore, we designed the agent software to use PyWinCtl and PyAutoGUI, libraries that emulate mouse and keyboard operations from Python, to control the Cluster client. Since the keys "w", "a", "s", and "d" are assigned to move the avatar in Cluster, we can move it in any direction by operating these keys. The avatar can also face the desired direction by emulating the mouse operation in the Python control.

Monitoring communication from the Cluster client to the server The Cluster client sends the avatar movements manipulated by the Agent software to the server as avatar location information (Fig. 2c-3). Our Agent API module monitors this communication to know which coordinates the avatar has moved to (Fig. 2c-2). For example, if there are target coordinates to move the avatar to, the difference between the target coordinates and the current coordinates is calculated to determine the next Cluster client operation.

Monitoring communication from the Cluster server to the client By monitoring messages from the Cluster server to the Cluster client, the agent API module can know at which coordinates the other users are located (Fig. 2d-2). Using the other user's location information thus obtained, the agent software can move the operating avatar closer to the other user's position or change its orientation in that direction.

Table 1: The Agent API requests agent software to respond to the these pates. Each path corresponds to a type of data in VR space.

Path	Contents of data to be sent
/location	Avatar position of other users, Avatar location information operated by the agent
/emote	Information about emotes displayed by other users
/comment	The contents of comments entered by other users are POSTed at the time of commenting.

2.2.2 Communication from the Agent API module to the agent software

The Agent API module selects only the information necessary for the agent's operation from the content of the monitored communication, converts it into an easy-to-use form, and sends it to the agent software. Specifically, this information includes location information of other users in space, comments, and emotional expressions called emotes. We show the list of HTTP requests from the Agent API module to the agent software and their detailed contents in Table 1.

The Agent API module communicates with the agent software via HTTP. We make all HTTP requests using the POST method and store the necessary data in the payload in JSON. We choose HTTP for communication from the API module to the agent software to reduce the learning costs for developers implementing the agent software. Although we can consider alternatives such as using a proprietary protocol instead of HTTP, HTTP server implementations are available in many programming languages, and developers can concentrate on implementing agent control rather than communication protocols. We plan for Cluster to provide the agent API module so developers can experiment with various software-only agent implementations.

2.2.3 Communication from the agent software to the agent API module

The agent software manipulates the Cluster client using mouse and keyboard emulation to control the avatar. However, entering com-

ments and selecting emotes using mouse and keyboard emulation is complex and increases input uncertainty. Therefore, the Agent API module has a function to accept the input of comments and emotes. This feature is implemented as an HTTP server, allowing the agent software to send requests to the agent API module via HTTP. This will enable comments and emoticons to be sent to the Cluster server without mouse or keyboard emulation.

We implement the agent API module and agent software designed as described above using Python.

3 Implementation and evaluation of an agent in a VR environment

Using the proposed agent API module, we implement an agent to communicate with other Cluster users. We use the Large Language Model (LLM) to generate the text necessary for communication and to control some of the avatars, and we use OpenAI's get-3.5-turbo as the LLM. The agent software can detect when a user is approaching the agent-operated avatar by measuring changes in user location information sent from the agent API module. We could implement the above agent in about 200 lines of Python script. Figure 1 shows the interaction between the user and the agent in the user's first-person viewpoint image, where the agent is programmed to return a greeting in the comment field in response to the user's approach.

Here, we did not prepare the greeting as a canned text, but we generated it each time using Open AI's API. We show an example of a prompt used below.

- A person by the name of <username > approached me. Please say something to him.
- A person by the name of <username > said <comment >. Please reply in as short a sentence as possible.
- A person with the name <username > moved away. Please say something to them.
- A person by the name of <username > show the emotion <emotion icon >. Please say something to them.

We confirmed that the system generates appropriate greetings by describing the avatar's situation and distance from the user, as in the prompt example. In other words, the user could spontaneously greet other avatars as they approached and say goodbye to avatars who were moving away. The system could also comment on the emote. We also found that the user's history and the avatar's approach and departure are stored, so the greeting text reflects this behavior.

As the speed of the conversational response significantly impacts the user experience, we measured the time required for the agent to respond to the conversation. We prepare 110 quizzes from the official AI King distribution dataset [1] as the question text to the agents. We asked the agents about the above quizzes using comments and measured the time from when the question remarks were posted to when the answers from the agents reached the clients. As we measured the client's time on the questioner's side, this time also included the round-trip communication time via the Internet. The results show that the average time to respond to the conversation was 1.34 s, with a standard deviation of 0.27 s. From this response time, the implemented agent has a sufficient response speed as a communication response, and we expect the conversation to be natural for the user.

4 Conclusion

In this paper, we proposed an agent API to realize a system in which an agent freely designed by the user can operate in a VR environment. We designed and implemented an agent API module in the Cluster, a social VR service that many users use. We also implemented an agent that runs in a VR environment using the API and showed that natural communication with users is possible using LLM.

Prospects include API modules and software openness. In the current implementation, it has not yet been possible to open the software widely due to the security risks for the Cluster companies. It is necessary to establish an environment where many researchers can use this API in the future through a simple contract with Cluster. In addition, to reach users widely as part of the service, having an agent on the Cluster server side will be necessary rather than having the agent software operate Cluster clients.

Acknowledgments

This work was partially supported by JST ASPIRE Grant Number JPMJAP2327.

References

[1] abc/EQIDEN Executive Committee. Ai king official distribution dataset. `https://sites.google.com/view/project-aio/dataset`.

[2] Cluster, Inc. Metaverse platform - cluster, 2017. `https://cluster.mu/en`.

[3] W. Hai, N. Jain, A. Wydra, N. M. Thalmann, and D. Thalmann. Increasing the feeling of social presence by incorporating realistic interactions in multi-party VR. In *Proceedings of the 31st International Conference on Computer Animation and Social Agents*, CASA 2018, pp. 7–10, May 2018.

[4] G. Kaimal, K. Carroll-Haskins, M. Berberian, A. Dougherty, N. Carlton, and A. Ramakrishnan. Virtual reality in art therapy: A pilot qualitative study of the novel medium and implications for practice. *Art Therapy*, 37(1):16–24, Jan. 2020.

[5] G. Makransky and L. Lilleholt. A structural equation modeling investigation of the emotional value of immersive virtual reality in education. *Educ. Technol. Res. Dev.*, 66(5):1141–1164, Oct. 2018.

[6] Mojang. Minecraft. `https://www.minecraft.net/`, Dec. 2023. Accessed: 2024-1-11.

[7] OpenAI. ChatGPT (Mar 14 version) [Large language model]., 2023. `https://chat.openai.com`.

[8] PrismarineJS. mineflayer: Create minecraft bots with a powerful, stable, and high level JavaScript API.

[9] Reuters. ChatGPT sets record for fastest-growing user base - analyst note. *Reuters*, Feb. 2023.

[10] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv: Arxiv-2305.16291*, 2023.