Agent-based management of support systems for distributed brainstorming

Yuki Kaeri, Kenji Sugawara, Claude Moulin, Thierry Gidel

Advanced Engineering Informatics 44 (2020) 101050

Contents lists available at ScienceDirect

Advanced Engineering Informatics

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

Full length article

Agent-based management of support systems for distributed brainstorming

Yuki Kaeri[a,*], Kenji Sugawara[b], Claude Moulin[c], Thierry Gidel[d]

[a] Faculty of Media Studies, Department of Media Studies, Mejiro University, Japan
[b] Faculty of Information and Network Science Chiba Institute of Technology, Japan
[c] Sorbonne Universités, Université de Technologie de Compiègne, JR Unit CNRS 7253, Heudiasyc, France
[d] Sorbonne Universités, Université de Technologie de Compiègne, COSTECH EA 2223, Centre de Recherche, France

ARTICLE INFO

Keywords:
Multi-agent system
Resource-oriented architecture
Distributed brainstorming
Flexibility
Usability

ABSTRACT

In this paper, we describe the design and the management of an agent-based system that supports distributed brainstorming activities. The support system is a highly coordinated IoT application composed of many locally installed interface devices, multimedia communication functions, and cloud functions that process application logic and store meeting data. The system is designed to support a variety of brainstorming sessions, so its functionalities must be modifiable and enable the system to be adapted to different environments and user requirements without any loss of performance. System accessibility should be also ensured from any location for any user. These constraints require a flexible and usable support system.

We further discuss the aspects of flexibility and usability that are important in a support system for distributed brainstorming, from which we propose a conceptual schema for flexible and usable support systems. To realize this schema, we present a resource-oriented architecture that can modify the brainstorming support system's structure and functions. Flexibility is achieved thanks to an agent-based system that manages resources and operates on them according to users' requests.

We also describe the system architecture, which is organized around a set of channels dedicated to different services proposed to the users. We present in detail a video channel that ensures user awareness during synchronized activities. We then conduct several experiments verifying the usability of important channels in the architecture and present the results of these experiments.

Finally, we discuss experimental scenarios that show how the system owes its adaptability to management based on an agent organization that supports distributed brainstorming and other activities.

1. Introduction

In the architectural reference model (ARM) defined in the Internet of Things Architecture (IoT-A) European Research Project, "things" consist of three types of devices—sensors, tags, and actuators—that can provide data to different external systems [1]. According to these reference models [2], an IoT application is largely composed of three elements: cloud resources, network resources, and edge resources [3]. Since edge resources depend on physical installation conditions and geographical constraints, IoT applications require connections between the edge, network, and cloud resources whose runtime matches the operation conditions. In order to build a complex IoT application, resource-oriented architecture (ROA) has been studied [4]. For example, Dar et al. proposed an ROA to connect front-end IoT devices with back-end business process applications, promising programmer-friendly access to IoT devices [5].

Agent-based development also provides a suitable and effective modeling method and programming paradigm for IoT systems [6,7]. For example, an agent-based cooperating smart object (ACOSO) approach—based on metamodels and exploitable at the system analysis, design, and implementation levels—has been studied for building and managing IoT systems [8,9].

On the other hand, collaborative work between distributed sites over the Internet has become important for companies and universities in the world. Brainstorming sessions are a good example of collaborative methods used for new product design and software development at different stages, such as planning or defining requirements. Our support system is based on the KJ method [10], developed by Jiro Kawakita, in which ideas are presented as Post-it notes and can be manipulated using large multitouch multi-user tabletop displays [11]. The results of meetings using such collaborative work methods are saved in a database and may be shared between participants located in the same place

* Corresponding author at: Mejiro University, 4-31-1 Nakaochiai, Shinjuku-ku, Tokyo, Japan.
E-mail addresses: yk.kaeri@gmail.com (Y. Kaeri), suga@net.it-chiba.ac.jp (K. Sugawara), claude.moulin@utc.fr (C. Moulin), thierry.gidel@utc.fr (T. Gidel).

https://doi.org/10.1016/j.aei.2020.101050
Received 29 December 2018; Received in revised form 7 September 2019; Accepted 28 October 2019

Fig. 1. TATIN: Brainstorming support system using a multitouch interface.

on a shared screen, but also between participants at different sites. The use of various devices with rich functions deployed within the system allow it to capture, capitalize on, and treat useful information [12].

In summary, a support system for distributed brainstorming is a highly coordinated IoT application composed of many locally installed interface devices, synchronous multimedia communication functions, and cloud functions that rapidly process and store meeting data. The functionality and performance of the support system must be changeable across sessions to adapt to various kinds of brainstorming, and within a session to adhere to the various environments and user requirements during each brainstorming session. Accessibility to the support system should be also ensured for any user from any location.

Operating a support system and adapting its structure according to the evolution of a session is a challenging topic. In order to easily use such a complex support system for brainstorming sessions, users should be supported by intelligent functions. In this paper, we discuss the flexibility and a usability of support systems for distributed brainstorming and propose a conceptual schema of flexible and usable support systems for distributed brainstorming (FUSSDBs). To realize this schema, we further propose an ROA and an agent-based system for managing resources.

In Section 2, we investigate the key concepts of "flexibility," "usability," and "ROA." In Section 3, we define the problem we aim to solve in this paper and provide an overview of our FUSSDB as well as an architecture for agent-based resource management. The FUSSDB is composed of a usability-aware service orchestration system (USOS) and a flexible support system (FSS) for distributed brainstorming. In Section 4, we describe the FSS's ROA-based design. In Section 5, we detail the design of the USOS, a multi-agent system that is composed of user agents and orchestration agents (OAs). In Section 6, we implement the FSS and USOS and conduct experiments aiming to validate their flexibility and usability. Finally, in Section 7, we discuss the flexibility and usability of the proposed system considering the experimental results.

2. Related work

2.1. Brainstorming support from multimedia devices

Brainstorming is a meeting facilitation technique that improves creative efficiency by sharing opinions among participants, allowing them to seek solutions to problems and generate ideas. Existing brainstorming support systems, such as TATIN-PIC [13], Groupgarden [14], GKJ [15], and ScriptStorm [16], mainly support design and project management processes. These systems assume that such processes can be improved if all users can simultaneously submit ideas via Post-it note, and then discuss, enrich, and organize these ideas (Fig. 1).

2.2. Distributed brainstorming support system

Distributed brainstorming exists as a method of brainstorming between distributed sites [17]. In a survey of distributed collaboration, Marlow found that "current tools and approaches are inadequate for meeting scenarios that require participants to not only converse but also to share and co-reference different types of multimedia content across distance," and that, "Participants use a mixed set of tools to support different needs within different types of meetings, as no single video conferencing tool includes sufficient functionality to address all their demands" [18].

Tools for brainstorming include those that support media with different functions (Skype, Google Hangouts, Google Docs[1], etc.) and must be accessed by users at the same time [19]. However, many audiovisual conference support applications often do not permit data accumulation and reuse, leading to insufficient asynchronous brainstorming support. Current research and Web services that support distributed brainstorming include IdeaStream [20], Bubbl.us[2], MindMeister[3] , Stormboard[4], XMind[5], and Mindomo[6].

2.3. Capitalizing on distributed brainstorming meeting resources

A meeting resource is defined as an object or information used or generated during brainstorming, including documents, video streams, sensor signals, and meeting records. In distributed brainstorming, users want to share meeting resources by activating applications necessary for sharing information [19]. However, it is difficult for users to launch these applications on demand during brainstorming.

Additionally, although Skype for Business[7] and Google Hangouts[8] can store meeting video on a local or cloud server, these applications were not intended to relate the video streams to other media or information generated during the meeting. If it could be combined with

[1] https://docs.google.com/.
[2] https://bubbl.us/.
[3] https://www.mindmeister.com.
[4] https://stormboard.com/.
[5] http://xmind.net/.
[6] https://www.mindomo.com/.
[7] https://www.skype.com/business/.
[8] https://hangouts.google.com/.

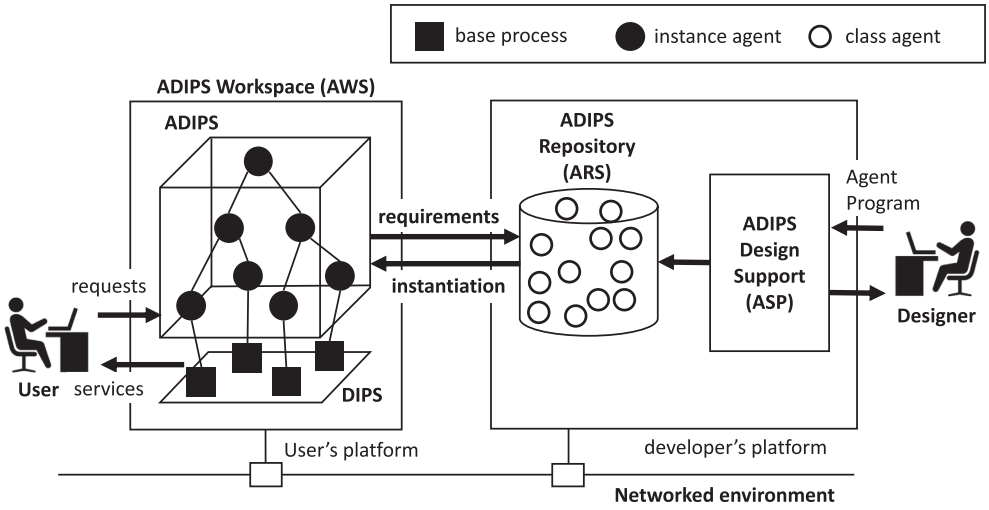


Fig. 2. The architecture of the ADIPS framework.

this additional information, the video information would be rendered more useful and reusable.

The concept of capitalizing on, accumulating, and using the results of brainstorming asynchronously is not new [12], but no operational system for reusing these data has yet been realized.

2.4. System flexibility

Flexibility is important for systems working in today's ever-changing, complex, dynamic, and global environment [21]. Software system flexibility can be understood in two contexts: structural flexibility and process flexibility. Structural flexibility is the capacity of the design and organization of a software application to be adapted to business changes. Process flexibility is the ability of users to make changes to the technology using management processes that support business changes [22].

Flexibility is judged using metrics such as the range of supportable changes, response time, sustainable duration of a stable state, and margin to the following change anticipated. Agent-based approaches have been explored for realizing system flexibility by calculating these metrics and changing the system structure and functions [23].

2.5. System usability

Usability is defined as the ease of use and acceptability of a system for a class of users carrying out specific tasks in a specific environment [24]. Ease of use affects user performance and satisfaction, whereas acceptability affects whether the product is used [25]. Heuristic evaluation (HE) is the most common informal method for inspecting system usability; disadvantages of HE include separation from the end user and an inability to identify or allow for unknown users' needs [25].

A collaborative usability inspection is a systematic examination of a finished system by a team of developers, users, domain experts, and usability specialists. One advantage of a collaborative usability inspection is that is allows developers to build skills and knowledge about the usability of developing systems [24]. Usability can be assessed by metrics, such as the usability metric for user experience, that are concerned with improving user experiences [26]. It is a challenging study to acquire these skills, know-how and metrics from interacting systems with people because it may be able to contribute building intelligent user interfaces.

2.6. Resource-oriented architecture

A service is a discrete unit of functionality that can be accessed remotely over the Internet. A service-oriented architecture (SOA) is a style of software design in which services are provided to users or other system components. In contrast, resource-oriented architecture (ROA) has been studied to provide an end-to-end integration architecture of front-end IoT devices with Web applications [27]. A system resource is any physical or virtual component that is deployed in an Internet environment. A virtual component is a discrete piece of code and/or data structure. An ROA is a style of software design where resources are connected using common application programming interfaces (APIs), such as REST, to reuse them for different purposes. The resources can be identified by universal resource identifiers on the World Wide Web.

Although the merits of SOAs and ROA have been debated, efforts have been made to unify their dominant implementations in cloud computing [4]. An ROA for the Web of Things (WoT) is designed as a refinement of the IoT by integrating smart things not only into the IoT but into the Web using a RESTful API [27].

An estimated 50 billion smart objects will be connected to the Internet by 2020. To provide value-added services to end users through IoT platforms, these devices must be discovered by resources in the cloud and by other devices. Services for resource discovery in the IoT are generally proposed using RESTful web services [28].

2.7. Agent-based system resource management

An agent is an object or a program that involves features of autonomy, reactivity, social ability, and pro-activeness [29]. An agent framework provides a working environment and development tools as well as an agent model for agent developers [30].

Agent-based computing (ABC) is suitable for implementing robust scalable systems and interoperable and virtualizable "things." ABC is suitable for supporting the design and implementation of autonomous IoT systems[31].

The agent-based distributed information processing system (ADIPS) framework is an agent framework developed to design and implement agent-based flexible systems based on a repository of agents, as shown in Fig. 2[32]. A repository of agents defined by the ADIPS framework stores a set of classes of agents (class agents) developed by agent providers using design support tools on the designer's platform.

The framework provides an ADIPS organization/reorganization protocol to generate a multi-agent system consisting of instantiated agents on a user's platform by class agents and an ADIPS communication/cooperation protocol to send and receive agent messages based on the knowledge query and manipulation language agent communication protocol [33]. Based on user requests, a multi-agent system is instantiated from the repository using the ADIPS organization/

reorganization protocol and ADIPS communication/cooperation protocol.

3. Approach to usable support systems for distributed brainstorming

3.1. Problems in developing support systems for distributed brainstorming

Marlow's statements regarding the inadequacies of current tools and approaches highlights the complexity of developing an IoT support system that realizes many different functions and can be used anytime, anywhere, and easily. If the support system can be developed to meet all requirements and adapt to all situations, it can become too difficult to use in typical work situation environments.

Usability is defined as the ease of use and acceptability of a system for a class of users carrying out specific tasks in a specific environment [24]. The problem explored in this paper is to create a usable architecture for support systems for distributed brainstorming.

3.2. Overview of a support system for distributed brainstorming

An overview of our proposed support system for distributed brainstorming is shown in Fig. 3. A distributed brainstorming environment consists of distributed sites $s[1]$, $s[2]$, ...,$s[N]$ and a cloud support system consisting of cloud support functions and a meeting resource storage. The cloud support functions that provide services, such as video conferencing, affinity diagrams for organizing ideas using notes, and document sharing for each site, are discussed in Section 4. A cloud support function also stores participant activities as data streams, such as video streams captured by cameras, into meeting resource storage. These functions help participants to use meeting records during or after a brainstorming session.

A local site consists of brainstorming participants and a local support system that is composed of interface devices and local support functions that brings services to the participants. The local support systems are connected to the cloud support system via the Internet. The cloud and the local support systems are explained in Section 4.

3.3. Conceptual schema of a flexible and usable support system for distributed brainstorming

A distributed brainstorming session (DBS) is a meeting for brainstorming conducted by users at different locations using an Internet-connected system. In this section, we formalize a model to define the flexibility and usability of a DBS support system.

Let i be a DBS identifier; then, $DBS(i)$ is denoted as follows:

$$DBS(i) = \langle S(i), F(i), U(i) \rangle. \quad (1)$$

$S(i)$ is a set of places $s(i, j)$, $(j = 1, ..., n)$, where j identifies the place at which a user participates in the $DBS(i)$. $F(i)$ is a set of functions $f(i, j, m)$ that provide services for users at $site(i, j)$, where m identifies a function. $U(i)$ is a set of user properties $up(i, p)$, where p identifies a user who participates in the $DBS(i)$.

To introduce usability to our support system, as in Fig. 3, we propose a conceptual FUSSDB schema in Fig. 4. The FUSSDB is composed of usability-aware service orchestration system (USOS) and flexible support system (FSS) for distributed brainstorming. We assume that many sensor devices are deployed at any site $DBS(i)$ and data from the sensors are captured by the USOS as meeting resources.

An FSS for $DBS(i)$ gives services provided by the functions $F(i)$ to $S(i)$. An FSS is flexible for a $DBS(i)$ if the $S(i)$ and $F(i)$ can be modified by operations performed by the USOS. The two kinds of flexibility (structural and functional) of an FSS are defined as follows, according to Nelson and Cooprider [22]: structural flexibility is defined by the ability to change the number of connected places and participating users during DBS(i) and process flexibility is defined by the ability of users to change the structures and functions of support systems to adapt to evolving brainstorming sessions.

A USOS is defined by its ability to orchestrate a series of operations to change the structure and functions of an FSS according to changes in user requirements, as shown in Fig. 4. A USOS supports unskilled users in operating an FSS during evolving brainstorming sessions. The USOS acquires users' requirements through a user interface or by analyzing user activity stored in meeting resource storage, as in Fig. 3; then, it orchestrates a series of FSS operations.

3.4. Agent-based management of a flexible and usable distributed brainstorming support system

A new FUSSDB conceptual schema is proposed in Section 3.3. To develop a FUSSDB based on the schema in Fig. 4, we designed a FUSSDB architecture (Fig. 5), which is composed of an agent subsystem (AS) to develop the USOS and a resource subsystem (RS) to develop the FSS, based on the ROA explained in Section 2.6.

The RS is a subsystem consisting of physical and virtual components, where virtual components are discrete pieces of code and/or data structures, as stated in Section 2.6. There are four categories of RS resources:

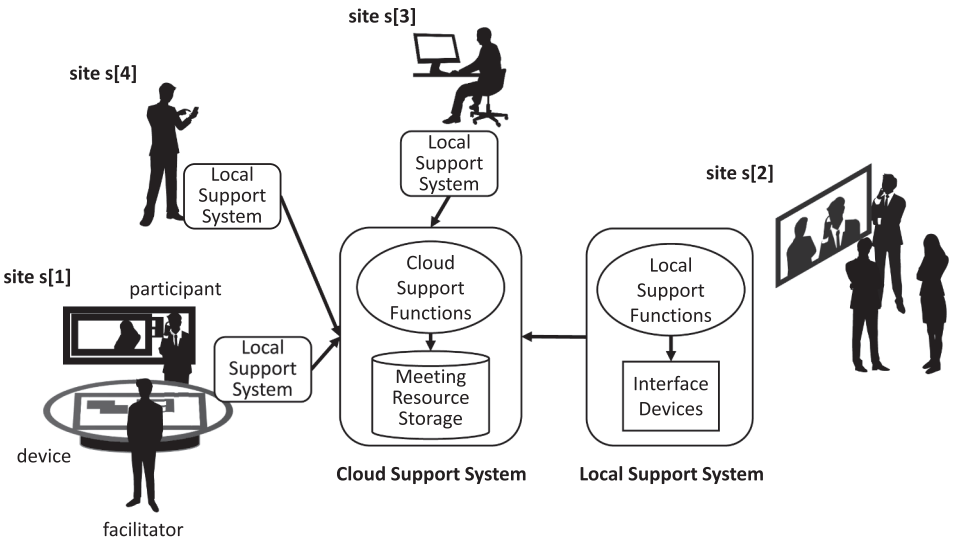


Fig. 3. Overview of a proposed support system for distributed brainstorming.

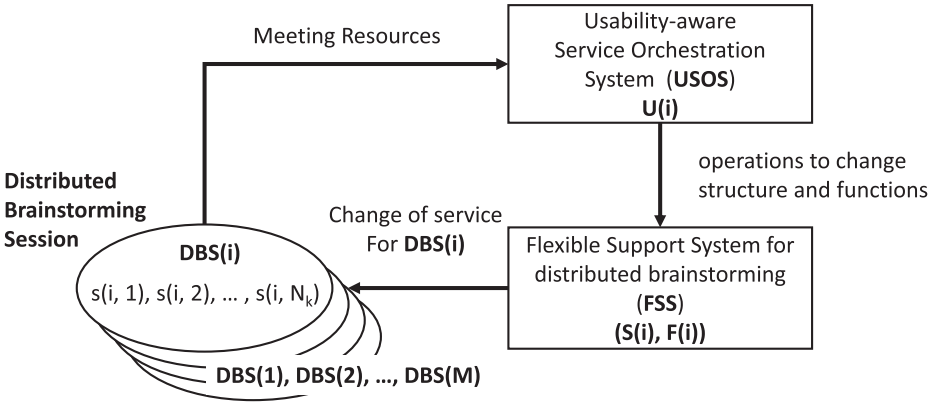


Fig. 4. Conceptual schema of a FUSSDB.

- A cloud resource is a piece of code or data in the cloud used to process application logic directed to edge resources.
- A network resource provides communication and networking services for cloud and edge resources
- An edge resource is a device that interacts with a user and its environment and the set of programs that control it and communicate with other edge resources
- A meeting resource is a collection of data and streams acquired by sensors in a DBS, used by AS agents to analyze user activity, and capitalized for future exploitation

A Linux operating system is used for the cloud resources. Network resources are services that may use different networks, such as the Internet, LANs, Wi-Fi, or 5G. Edge resources run on a PC or small devices and benefit from communication services provided by the network resources.

The AS is a subsystem consisting of agents to develop the USOS in the FUSSDB conceptual schema (Section 3.3). The AS is composed of the following types of agents:

- A user agent interacts with a user to acquire his/her requests
- An OA organizes and communicates with a collection of RS resources
- A resource connector directly controls an assigned resource from OA messages
- A meeting resource analysis agent extracts explicit/implicit user requests from meeting resources

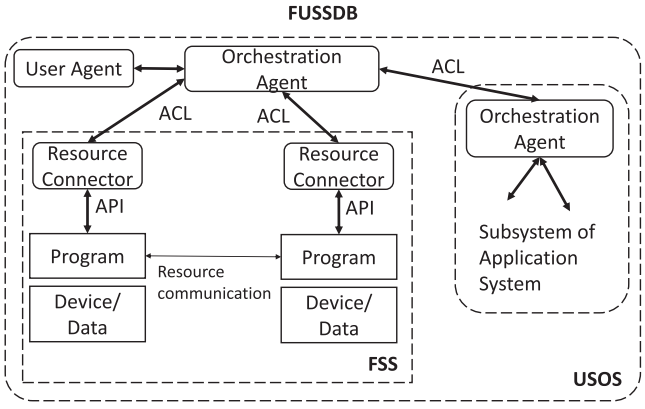


Fig. 6. Agent-based resource management in FSS for usable FUSSDB.

More precisely, a resource connector (see Fig. 6) is an agent that controls a device and uses network resources to send and receive data from other cloud and edge resources. A resource connector also sends and receives agent messages in agent communication language. We assume that the API for any resource is prepared by the provider who installs the resource in a DBS environment. We also assume that a resource connector is installed on any edge resource.

A user agent requests an OA to launch an application system to support brainstorming activities. The OA invokes an organization of agents, including resource connector agents, and cooperates with it

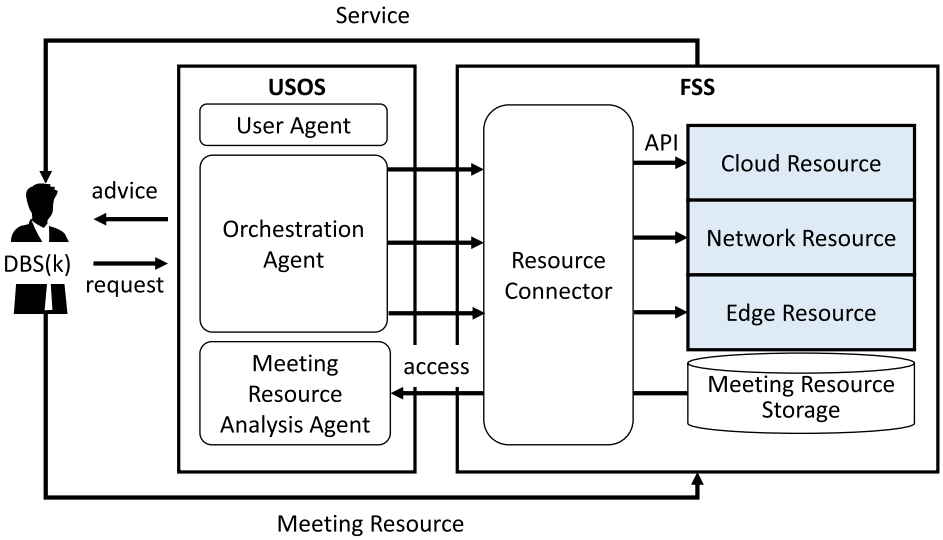


Fig. 5. Agent-based and resource-oriented FUSSDB architecture.

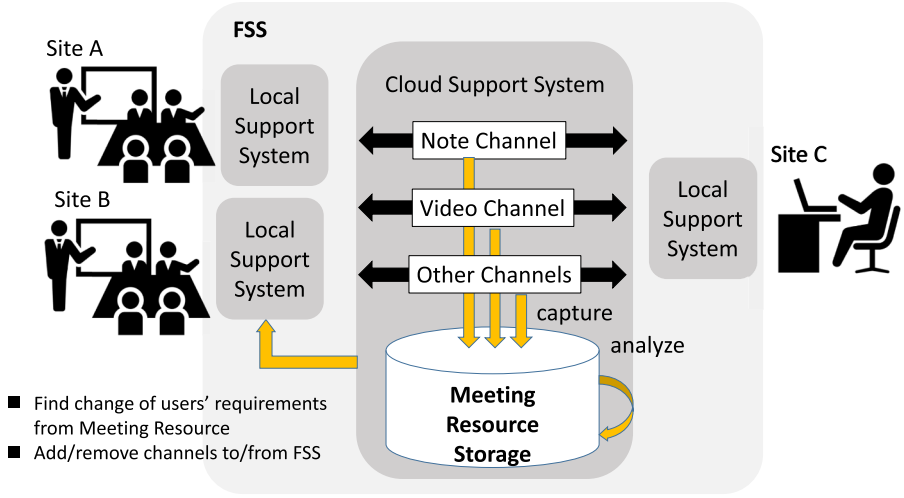


Fig. 7. Channel-based FSS architecture consisting of resources in local support systems and in a cloud support system.

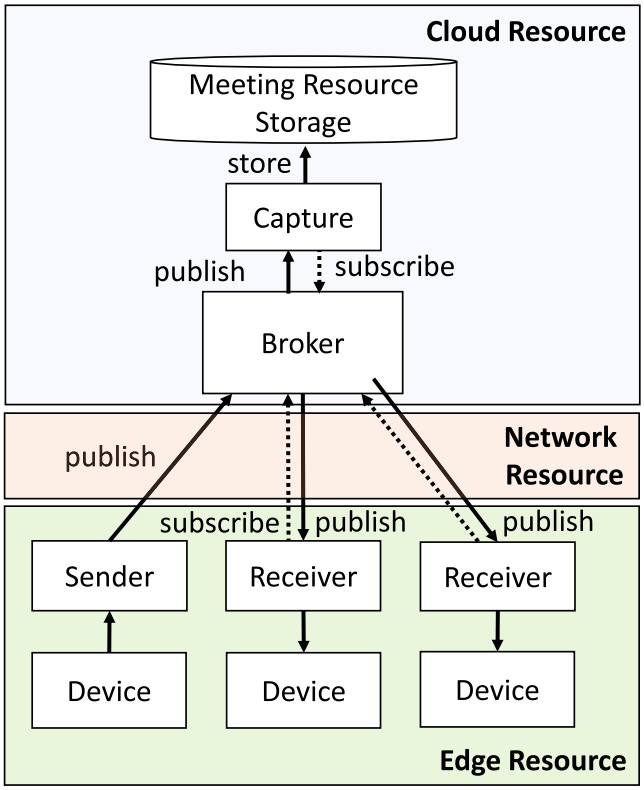


Fig. 8. Structure of a primitive channel based on the publish/subscribe model.

using the ADIPS protocol [32] (see Fig. 6).

4. Channel-based system architecture of a flexible support system for distributed brainstorming

4.1. System architecture of a multimedia support system for distributed brainstorming

In this paper, we propose an innovative system architecture for multimedia systems, composed of an FSS and a USOS, to support distributed brainstorming. In this section, we present an FSS design that uses the concept of a channel that is composed of a cloud resources, network resources, edge resources, and meeting resources, as shown in Fig. 5. Each channel, depicted in Fig. 7, is an autonomous application that allows people at distributed sites to either discuss and exchange ideas (via a video channel) or perform specific activities (via the note and document channels) during brainstorming.

Each channel stores the data that crosses over it in a cloud database. The video channel stores the video streams captured by cameras installed at each site. The note channel stores notes containing ideas written down during a session. Meeting data can easily be consulted after the session by participants and other project members. The FSS in Fig. 7, consisting of channels that connect resources in local support systems via resources in a cloud support system, supports several activities occurring during distributed meetings.

Another advantage of this architecture is that it allows for analysis of the captured data by programs during or after a meeting and can produce meeting records for participants. As mentioned in Section 2.3, the concept of capitalization[12] is important for brainstorming. In such a system, data from channels can be transformed by artificial intelligence-based programs into high-level knowledge.

4.2. Primitive channel design

A primitive channel is defined by a system of devices and cloud programs for sending and receiving data or streams among distributed clients, such as cameras and displays. We refer to the set of devices, cloud programs, and middleware that connects devices and programs involved in this application as primitive channel resources; we refer to the collection of resources that works on the resource platform as an flexible support system [34]. A primitive channel is designed and implemented as a system that works autonomously after it is launched on a resource platform; we provide an API to reconfigure these channels in this paper.

The primitive channel is based on the publish/subscribe model [35], as shown in Fig. 8. In this model, a sender transmits data for a publish operation to a server program called a broker, which broker transmits data to clients that have performed subscribe operations. Using these operations, clients of a primitive channel can transmit meeting data from a sender to receivers and, with the same principles, save data into cloud storage.

One advantage of this model is that it enables the design and implementation of a primitive channel that serves distributed clients and records data in the cloud. Another advantage is in the flexibility to add and remove clients dynamically [36] to and from the primitive channel. These modifications change the structure and function of the communication middleware. Obviously, it would be complicated for users who are not familiar with Internet middleware operations to operate the primitive channel. To overcome this challenge, our architecture

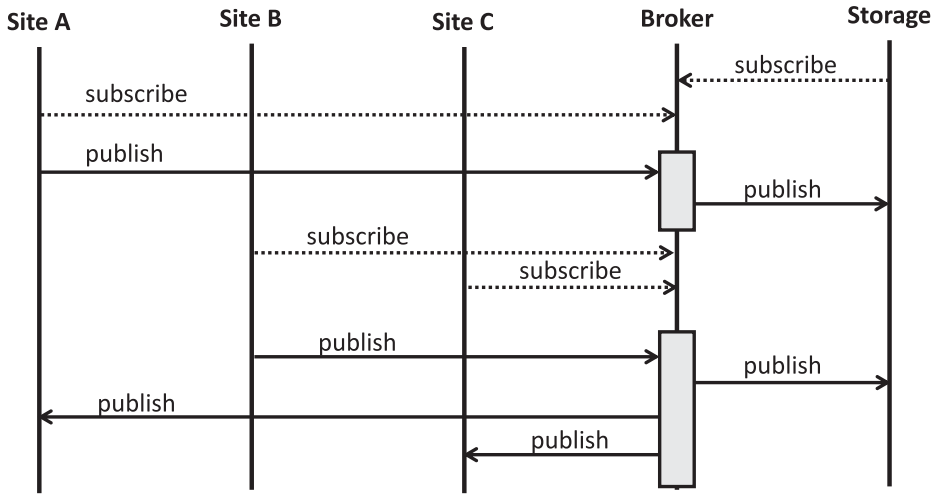
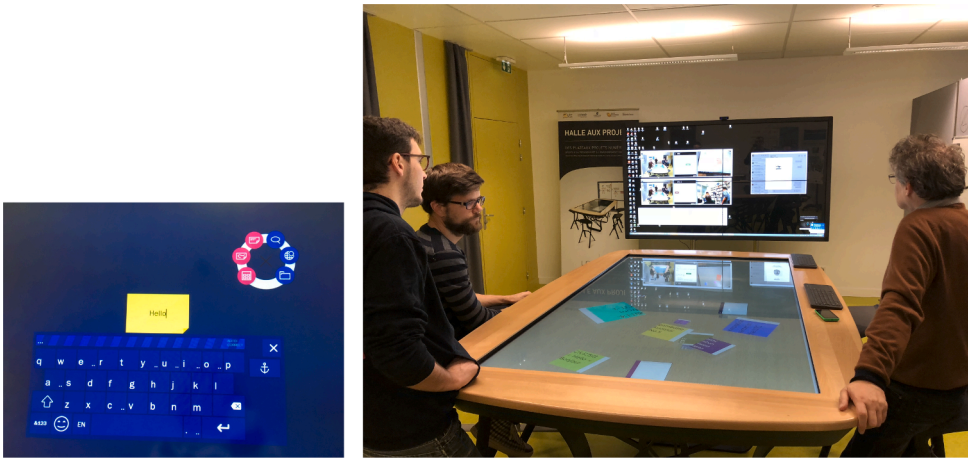


Fig. 9. Sequence diagram of the publish/subscribe model.



(a) Keyboard - menu (b) Tabletop

Fig. 10. Note channel viewer.

proposes a transparent agent-based support for users, which is presented in Section 5.

Fig. 9 shows a sequence diagram that details the publish/subscribe mechanism of a primitive channel. It is composed of three clients, a broker, and a "storage" program that saves data in a cloud database. When a primitive channel is launched, the storage program first sends a "subscribe" message to a broker. For example, as shown in Fig. 9, Site A sends a "subscribe" message to a broker upon launching. Site A then sends a "publish" message to the broker; the published message is sent to the storage program only because it is the unique subscriber at this point. After the event, suppose that Site B and Site C send a "subscribe" message to the broker. When a publish message is sent to the broker from the Site B, the message can be received by Site A, Site C, and the storage client. When a new client sends a subscribe message to the broker, it is accessible and involved in the primitive channel and will automatically publish new data.

4.3. Note channel design

A note channel supports idea generation during brainstorming sessions. A multitouch display device is used at each site and a specific program has been developed to capture users' gestures on this device. Fig. 10 shows the multitouch tabletop used at the University of Technology of Compiègne, in France (Fig. 10b), where some notes are displayed. On such a device, it is necessary to use a virtual keyboard to edit notes and virtual menus to select actions (Fig. 10a). Created objects are graphically represented as virtual Post-it notes and clusters on the multitouch display.

The program on the device sends events ("create note," "move note," etc.) resulting from user's actions, associated with time stamps, to a broker. In another site, the same kind of program, which has subscribed as a data recipient, receives the events and can display objects on the multitouch surface, reproducing the original actions. Locking events prevents moving the same object in two different sites. The MQTT[9] protocol is used to transfer events between sites.

4.4. Video channel design

A video channel is a synchronous video distribution application that delivers a video stream captured from a camera at one site to be displayed at distributed sites. Such channels aim to improve the awareness of distributed teams during brainstorming sessions by dynamically setting cameras and displaying and storing video streams. The WebRTC selective forwarding unit (SFU)[10] protocol is used for synchronous video stream delivery.

[9] https://www.iso.org/standard/69466.html.
[10] https://www.w3.org/TR/webrtc/.

4.5. Document channel design

A document channel is an application that ensures a synchronous delivery method for exchanging documents useful for discussion and presentation during brainstorming. The system posts document files in the cloud and can refer to them at any time according to the needs of participants. Events ("change page," etc.) are sent by the application and received at other sites to reproduce the original actions. The WebSocket[11] protocol is used as a network resource.

5. Agent-based management of FSS resources by the USOS

5.1. Architecture of an agent-based support system

The proposed FSS architecture is a channel-based system, as shown in Section 4 is complicated because each channel in the system consists of various kinds and versions of resources provided by different producers, as shown in Figs. 11 and 17. For example, if a meeting site involves several participants who play important roles in a distributed brainstorming session, many cameras and an interaction tool such as a big multitouch table should be launched and connected to other sites. A fully integrated support system at a primary meeting site generally consists of many channels to exchange information with many sites. On the other hand, a mobile participant of distributed brainstorming may need a compact system to join in using a mobile phone or small PC.

Therefore, unskilled users feel burdened if they must set up channels by themselves according to their own situations and requirements. The burden of launching the system can be reduced if an operator helps users set it up before they begin and while they use the system.

We proposed a USOS in Section 3.3. The USOS supports unskilled users in operating the FSS throughout the evolution of brainstorming sessions. We use an agent-based architecture to develop the FSS and USOS because the ADIPS framework, explained in Section 2.7, is expected to be suitable for promoting flexibility and orchestration ability.

5.2. Agent-based design for a channel-based support system

A primitive channel is designed and implemented as a system that works autonomously after it is launched on a resource platform, and that provides a reconfiguration API, as described in Section 4.2. A primitive channel is also a communication system among clients at distributed sites via a broker in the cloud. For example, a video channel in the FSS is composed of a sender program, a broker, and a receiver program. A sender program controls a camera device and publishes the captured data to a broker, and a receiver program subscribes to the service of the broker to receive the published data, as shown in Fig. 8.

A collection of resources that work on a computer or a device such as a smart phone is wrapped into a module using a resource connector. Resources in a resource connector provide APIs for the resource connector and for other resources in other resource connectors. A resource connector is a wrapper program that translates an agent message to a command for a resource and translates a series of data to inform agents in the USOS of the working state of a device, such as whether it is available for agents.

A primitive channel agent is an agent that directly controls some resource connectors by sending agent messages. By using resource connectors, a primitive channel agent in the USOS can use resources in the FSS, as shown in Fig. 12. A primitive channel agent is an autonomous system because a primitive channel of the agent is implemented as an autonomous and reconfigurable system. A channel agent controls some primitive agents and channel agents by sending messages. For example, a video channel agent can set up an identifier for a broker to subscribe and publish data for a sender resource and a receiver resource to build a channel service for a video channel, as shown in Fig. 12. A channel agent is an autonomous system if primitive channel agents of the multi-agent system are autonomous. Channel agents and primitive channel agents are OAs in the USOS, as shown in Fig. 5. The OAs are reusable components that constitute channel-based support systems using the ADIPS framework.

A primitive channel is a component that provides a communication service among sites that is controlled by a primitive channel agent. For example, a video sender agent sends video streams from one site to other sites. Two video sender agents send video streams to each other. A channel agent, as in Fig. 13, can be dynamically composed of different primitive agents, based on the distributed meeting environment, by changing the structure of OAs in the USOS.

Channel agents can also be organized hierarchically according to service structure. The top agent in this hierarchy is called a system agent. For example, a Distributed Brainstorming Support System Agent in Fig. 13 is a system agent. A system agent includes a collection of user agents normally corresponding to registered users. A user agent sends the user's requirements to a system agent that retrieves organizations of agents by cooperating with orchestration agents in USOS. System agents and user agents are the OAs in the USOS in Fig. 5.

5.3. Agent-based management of system structures by channel activation and deactivation

A channel agent is a multi-agent system that is presented by an organization of channel agents and primitive channel agents, as shown in Fig. 13. If a system agent invokes a new channel agent as per a user's request, a new service for the user is added to the system. In a similar way, a system agent can remove an unneeded channel from the system or replace an inadequate channel with an adequate one. We propose a mechanism for managing channel agents to restructure an agent-based system, as shown in Fig. 14, which we call a channel repository.

The channel repository is a collection of channel agents that can be reused by agent-based systems working in the USOS. Every channel agent has a state—either "active" or "inactive"—and the respective replacement operations of the states are called "activation" and "deactivation."

The channel repository is a collection of channel agents and primitive channel agents whose state is "inactive." A resource management agent (see Fig. 14) manages the profiles of the agents in the channel repository. A profile is a set containing an identifier, functions, performance values, a structure, and other features. The resource management agent performs "activation" and "deactivation" operations on channel agents working in the USOS to control resources. It retrieves the proper channel agent via usability-aware service orchestration using agent rules, according to system agent and user requests. The protocol between the resource management agent and agents in the channel repository is based on the ADIPS framework [32], which provides an agent repository and cooperation protocols for reusing the services realized by multi-agent systems according to requests sent by the user agents.

To meet these user requirements, the proposed agent-based architecture is composed of different spaces for local and cloud resource management and for system function orchestration by agents. The FSS of an agent-based channel contains modules with rich graphical user interfaces that fit the participants' tasks and support distributed brainstorming activities.

Thanks to the user interface and transparency, when people want to launch a new channel to support an activity, they can activate a resource connector that is kept ready for user agents and future operations generated by primitive channel agents. This functionality is made possible because we maintain the elements necessary for activating a new channel in a channel repository (shown in Fig. 14); the channel repository is a cloud database that manages unused resource connectors, primitive agents, and channel agents. An application agent

[11] https://html.spec.whatwg.org/multipage/web-sockets.html.

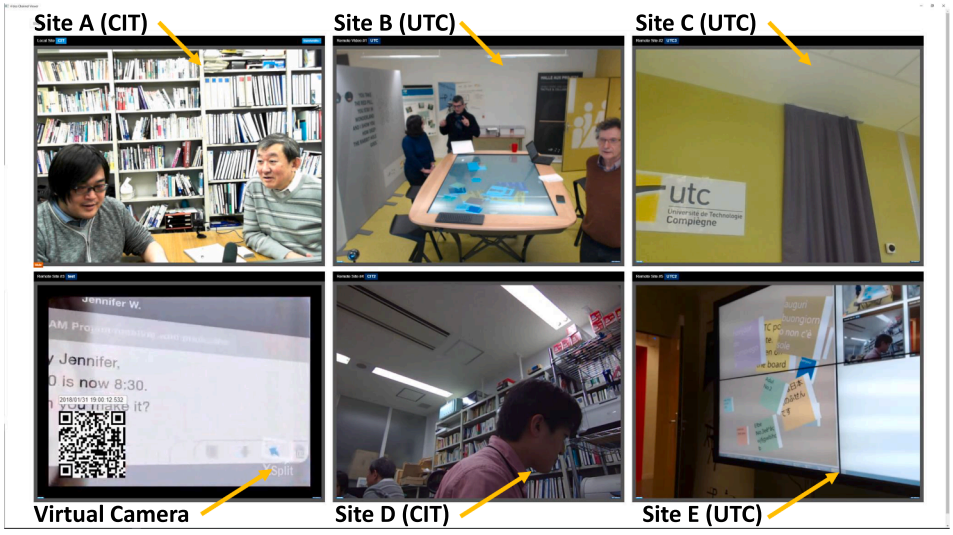


Fig. 11. Video channel viewer.

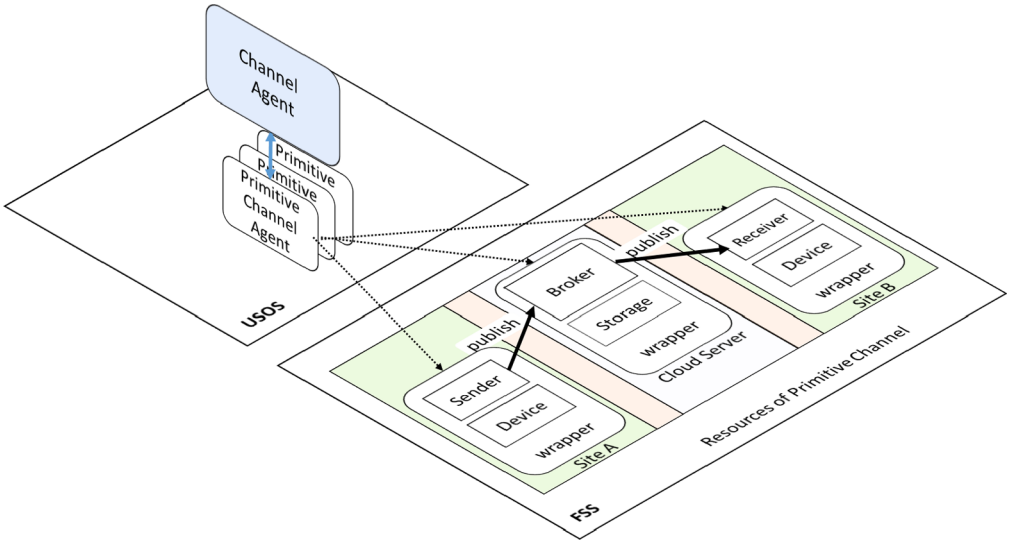


Fig. 12. Channel design consisting of an FSS and a USOS.

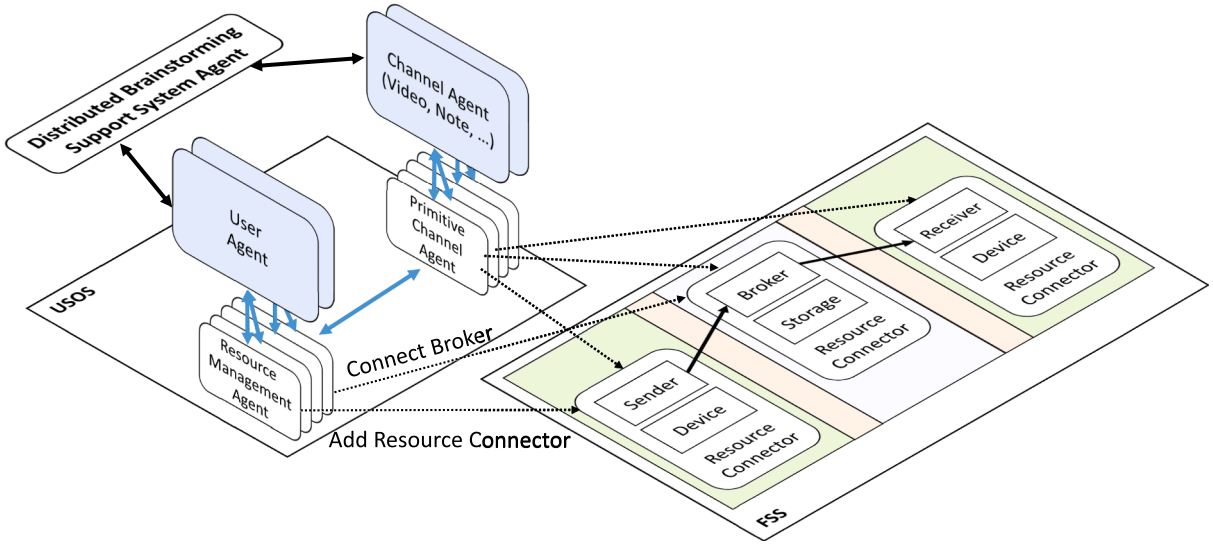


Fig. 13. Distributed brainstorming support system agent and resources.

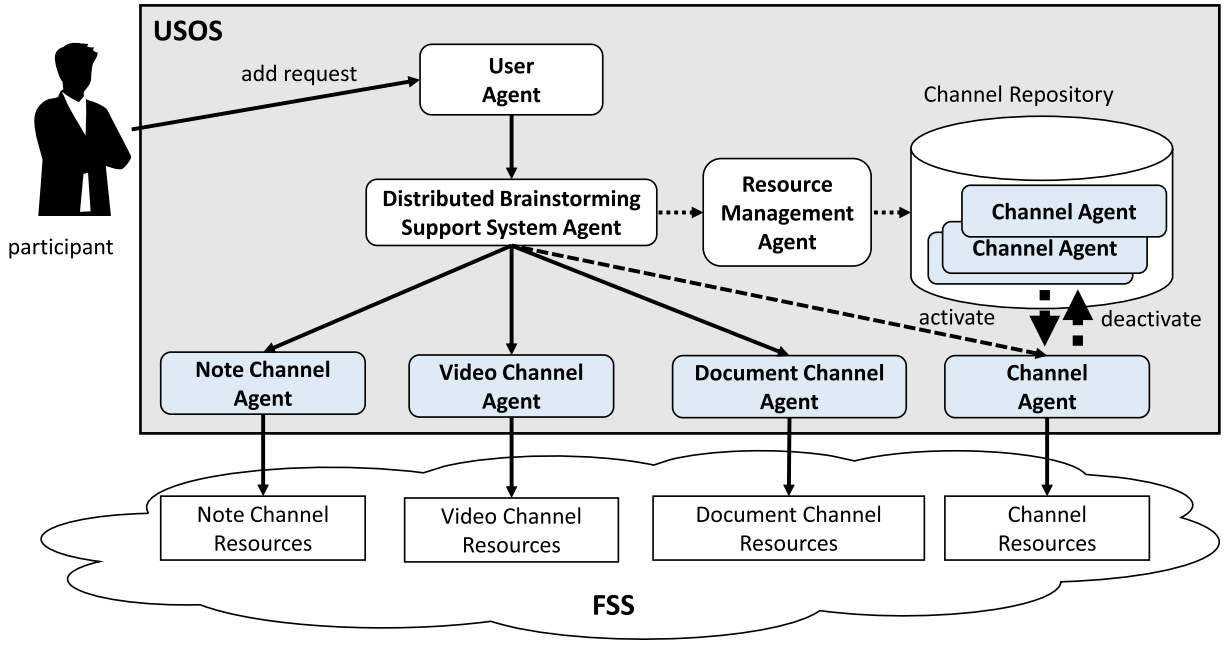


Fig. 14. Agent architecture of activation and deactivation of agent-based channels.

goes into the channel repository to retrieve the components for activating a channel, i.e., the components for its FSS and USOS.

6. Implementation and experiments

In order to test the proposed concept, we conducted the following two experiments: (1) We implemented the whole system according to the design proposed in this paper and confirmed whether the whole system works properly. (2) We checked whether each channel started up normally and if the operation succeeded. Moreover, we checked the system's flexibility and usability by testing the channel's operation.

In the first experiment, we tested several prototypes of the distributed brainstorming support systems proposed in this paper. The experiments involved a site at the University of Technology of Compiègne, France (UTC) and a site at the Chiba Institute of Technology, Japan (CIT). The experimental system is composed of several video channels and a note channel. Fig. 15 shows a snapshot of a small multitouch display and a large multitouch display at one CIT site. The small display shows a note channel that serves as a common interface between the two sites, used to exchange ideas via notes and clusters. The large multitouch display is composed of six video channels. In that display, each pane shows video streams captured by a camera installed at a site.

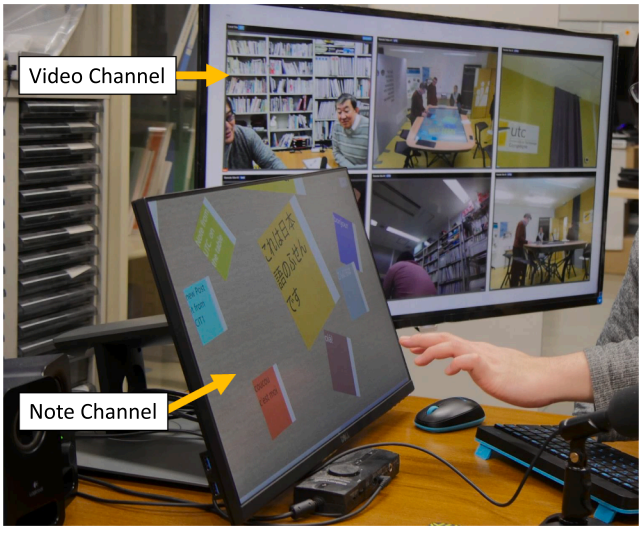


Fig. 15. Multichannel experiment.

At the beginning of the experiment, we started with two video channels, in order to make both sites visible during work. Then, the distributed meeting's mediator, a human participant located at CIT, added new channels one by one to the experimental system via a user agent. All six video channels were successfully launched while the system was running. A note channel also worked as a bidirectional channel. We experimented with a note channel and connected it to three sites (two sites at CIT and one at UTC). The common view of the note channel was visible at each site by subscribing via a common broker.

The components of resources in the FSS in Fig. 8 were implemented using C# and the .Net Framework. Resource connectors were developed in C#. A video channel broker was implemented in C using Janus[12]. Cloud storage was implemented in Python and C (using Redis[13] and a MariaDB[14] server). The WebRTC SFU protocol was used to implement connections between senders and receivers in the FSS to realize synchronous communication for video channels. Note channel communication was performed with the MQTT protocol. A note channel broker was implemented using VerneMQ[15].

The agents in the USOS were implemented using Python, based on the OMAS agent model [30,37]. The agents work and communicate in coteries connected by OMAS X-agents. Their skills were programmed in Python and activated by agent messages described in JSON form, which can handle FIPA-compatible [38] messages and can be saved in and used as agent knowledge. Communication between agents was implemented using the MQTT protocol.

In the second experiment, we confirmed that the system could dynamically add a new video channel simply by an unskilled user clicking on the Add Video Channel button. In this experiment, the USOS and FSS must work together to activate a new video channel.

Therefore, each agent has implemented rules for acquiring the

[12] https://janus.conf.meetecho.com/.
[13] https://redis.io/.
[14] https://mariadb.org/.
[15] https://vernemq.com/.

Fig. 16. Experiment demonstrating the "Add Channel" action.

status of the video channel and each resource (camera, display, and connection) and orchestrating them according to the user's requests.

Rule 1 shows an example of a rule of resource management agent, an orchestration agent that controls channel agents. This rule describes the state of each resource and the action to be taken when the user's request changes.

Next, Fig. 16 shows the experiment's result. Triggered by a GUI request from the user, the system was able to add a new video channel using cameras and displays existing at each site (without the user indicating the connection destination, etc.). The left shows the result before adding a channel, and the right shows the result of adding a new video channel via orchestration. At this time, agents work cooperatively. In addition, the agents could cooperate through the network, even at remote sites, to acquire and control data from resources. In addition, the user could simply request with a simple GUI to add channels, and the system can easily and dynamically add the necessary video channels.

7. Discussion

Generally, IoT applications are huge open systems that are affected by factors such as Internet connections, devices and PCs at each local site, and user mobility. These applications may be changeable and thus adapted to support users. A distributed brainstorming support system is a complex IoT application that uses the Internet to combine cloud resources such as web service components and databases, edge resources such as cameras, multitouch displays, and sensors.

During brainstorming sessions, the number of participants and distributed teams can change at each meeting. Further, the number and performance of video channels added to the system depend heavily on the objectives and types of meetings, from small or large groups. Alternatively, during a meeting, it may become necessary for participants to activate new video channels, activate new note views, or to bring new documents into the meeting support system.

```
rules = [
{ "conditions": { "all": [
      { "name": "current_video_channel_count",
        "operator": "less_than_or_equal_to",
        "value": 1,
      },
      { "name": "deployed_display",
        "operator": "greater_than_or_equal_to",
        "value": 1,
      },
      { "name": "deployed_camera",
        "operator": "greater_than_or_equal_to",
        "value": 1,
      },
      { "name": "connection_count",
        "operator": "greater_than_or_equal_to",
        "value": 1,
      }, ...
  ], ...},
  "actions": [
      { "name": "add_channel",
        "params": {"name": "video_channel"}},
      }, ...
  ],
}, ... ]
```

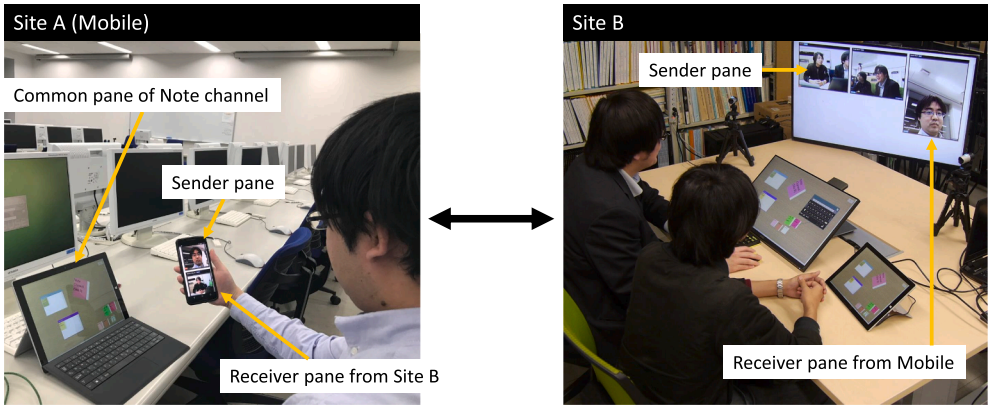


Fig. 17. Asymmetric scenario.

During experiments, the imperceptibly short delay between sites was appreciated by users; channel manipulation was simple and could be performed spontaneously. It was easy to determine how to access the functionalities of each channel via appropriate menus.

During brainstorming meetings, we believe that no more than three channel interfaces need to be accessed simultaneously by users: a video channel for people awareness, a note channel for producing ideas, and a document channel for presenting documents. What devices can be used to best support user interactions? We recommend two devices: a large tabletop device for the note channel and a large board device for the two other channels.

It is possible to add new channels to our system to support activities outside the scope of brainstorming, such as simulation or risk analysis. In this case, people would switch from one activity to another, which would not necessitate other devices.

This paper aimed to describe the architecture for supporting synchronous activities. Since data produced or consulted during meetings are stored in the cloud database, it is easy to reuse these data for other activities. For project members who cannot attend group meetings, it would be easy to access the meeting results. However, several applications should be added to our system in order to provide useful summaries of what happened during meetings.

Channels presented in this paper are essentially used in a symmetric way, i.e., people in each location have the same activities and the same role in performing them, and they also benefit from the same types of devices. Teams are in similar rooms and do not move from these rooms during the meeting. However, the architecture we designed also allows for asymmetric activities, which are slightly different from brainstorming. For example, as the video channel can be accessed from any device, including mobile devices, scenarios in which one team moves to show real situations to the static teams are possible without any modification of the architecture. Fig. 17 shows a team using a mobile phone to film a scene and the corresponding video channel view of the other teams.

In the same vein, it is possible for a meeting participant to connect to the video channel with a smartphone and produce a short video for the group. For example, somebody may record a short video summary at the end of a meeting.

In the second experiment, we confirmed that this system can orchestrate the necessary services and resources according to user requests and dynamically add the necessary channels, which provides flexibility.

Additionally, when the user clicks a button on the GUI, a request is transmitted and the necessary video channels can be activated by the collaboration of many agents. Therefore, even for an unskilled user, the system dynamically changes the environment based on a simple request, thereby facilitating information sharing between sites and achieving better communication. This result indicates the usability of the system.

8. Conclusion

Collaborative work between overseas organizations has become common and is mainly supported with video chat and information sharing tools. Because meeting requirements may vary (in the number of participants or quality of supporting functions, for example), users should be able to modify these tools as needed.

In this paper, we have proposed a distributed brainstorming support system that can be altered in structure and quality based on users' requests. In Section 3, we proposed a conceptual FUSSDB schema that is composed of an FSS and a USOS. The FSS, designed using channel-based architecture, was presented in Section 4; the USOS, designed using agent-based architecture, was described in Section 5. Then, experiments were conducted among distributed sites, as explained in Section 6.

When distributed teams collaborate, having a good video conference system that allows awareness between participants is crucial. The video channel we presented consists of several streams that can be easily activated and deactivated on demand. In the discussion, we also showed that the video channel may have uses beyond communication support. For example, it allows live transmissions during meetings and personal short sequences that are useful for data capitalization.

Each application is distributed because its elements physically belong to different computers, installed in the teams' locations and in the cloud. It consists of two logical spaces: USOS and FSS. FSS is an IoT environment consisting of devices, local control programs, cloud programs, and Internet communication services for distributed activity support. Agents in USOS manage components of FSS to launch, activate, deactivate, and modify channels according to users' requests during brainstorming meetings.

Declaration of Competing Interest

None.

References

[1] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of things (iot): a vision, architectural elements, and future directions, Future Gener. Comput. Syst. 29 (7) (2013) 1645–1660, https://doi.org/10.1016/j.future.2013.01.010.
[2] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, M. Ayyash, Internet of things: a survey on enabling technologies, protocols, and applications, IEEE Commun. Surv. Tutor. 17 (4) (2015) 2347–2376, https://doi.org/10.1109/COMST.2015.2444095.
[3] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, E. Riviere, Edge-centric computing: vision and challenges, SIGCOMM Comput. Commun. Rev. 45 (5) (2015) 37–42, https://doi.org/10.1145/2831347.2831354.
[4] H. Overdick, The resource-oriented architecture, in: 2007 IEEE Congress on Services (Services 2007), 2007, pp. 340–347. https://doi.org/10.1109/SERVICES.2007.66.
[5] K. Dar, A. Taherkordi, H. Baraki, F. Eliassen, K. Geihs, A resource oriented

integration architecture for the internet of things: a business process perspective, Pervas. Mobile Comput. 20 (2015) 145–159, https://doi.org/10.1016/j.pmcj.2014.11.005.
[6] Y. Kaeri, C. Moulin, K. Sugawara, Y. Manabe, Agent-based system architecture supporting remote collaboration via an internet of multimedia things approach, IEEE Access 6 (2018) 17067–17079, https://doi.org/10.1109/ACCESS.2018.2796307.
[7] C. Savaglio, G. Fortino, M. Ganzha, M. Paprzycki, C. Badica, M. Ivanovic, Agent-based computing in the internet of things: a survey, IDC (2017) 307–320.
[8] G. Fortino, A. Guerrieri, W. Russo, C. Savaglio, Towards a development methodology for smart object-oriented iot systems: a metamodel approach, 2015 IEEE International Conference on Systems, Man, and Cybernetics, 2015, pp. 1297–1302, , https://doi.org/10.1109/SMC.2015.231.
[9] G. Fortino, W. Russo, C. Savaglio, W. Shen, M. Zhou, Agent-oriented cooperative smart objects: from iot system design to implementation, IEEE Trans. Syst., Man, Cybernet.: Syst. 48 (11) (2018) 1939–1956, https://doi.org/10.1109/TSMC.2017.2780618.
[10] J. Kawakita, The Original KJ Method, Kawakita Research Institute, 1991.
[11] A. Jones, A. Kendira, D. Lenne, T. Gidel, C. Moulin, The TATIN-PIC Project, A Multimodal Collaborative Work Environment for Preliminary Design, in: 15th International Conference on Computer Supported Cooperative Work in Design, CSCWD 2011, June 8–10, 2011, pp. 154–161.
[12] C. Moulin, Y. Kaeri, K. Sugawara, M.-H. Abel, Capitalization of remote collaborative brainstorming activities, Comput. Stand. Interfaces (Elsevier) 48 (C) (2016) 217–224.
[13] A. Jones, C. Moulin, L.-P. Barthes, D. Lenne, A. Kendira, T. Gidel, Personal assistant agents and multi-agent middleware for cscw, in: Computer Supported Cooperative Work in Design (CSCWD), 2012 IEEE 16th International Conference on, IEEE, 2012, pp. 72–79.
[14] S. Tausch, D. Hausen, I. Kosan, A. Raltchev, H. Hussmann, Groupgarden: supporting brainstorming through a metaphorical group mirror on table or wall, in: Proceedings of the 8th Nordic Conference on Human-Computer Interaction Fun, Fast, Foundational - NordiCHI '14, ACM Press, 2014, pp. 541–550.
[15] M. Miura, T. Sugihara, S. Kunifuji, GKJ: Group KJ Method Support System Utilizing Digital Pens, IEICE Transactions on Information and Systems E94-D (3) (2011) 456–464.
[16] A. Clayphan, J. Kay, A. Weinberger, ScriptStorm: scripting to enhance tabletop brainstorming, Pers. Ubiquit. Comput. 18 (6) (2014) 1433–1453.
[17] K. Fujita, S. Kunifuji, A realization of a reflection of personal information on distributed brainstorming environment, in: T. Masuda, Y. Masunaga, M. Tsukamoto (Eds.), Worldwide Computing and Its Applications, Springer, Berlin Heidelberg, Berlin, Heidelberg, 1997, pp. 166–181.
[18] J. Marlow, S.A. Carter, N. Good, J.-W. Chen, Beyond Talking Heads: Multimedia Artifact Creation, Use, and Sharing in Distributed Meetings, in: Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing - CSCW '16, ACM Press, 2016, pp. 1701–1713.
[19] C. Wilson, Brainstorming and Beyond: A User-Centered Design Method, Morgan Kaufmann, 2013.
[20] F. Forster, Improving creative thinking abilities using a generic collaborative creativity support system, Res., Reflect. Innov. Integr. ICT Educ. 1 (2009) 539–543.
[21] A. Gorod, S.J. Gandhi, B. Sauser, J. Boardman, Flexibility of system of systems, Global J. Flexible Syst. Manage. 9 (4) (2008) 21–31, https://doi.org/10.1007/BF03396548.
[22] K. Nelson, J. Cooprider, The relationship of software system flexibility to software system and team performance, ICIS 2001 Proceedings, 2001, p. 4.
[23] T. Suganuma, T. Oide, S. Kitagami, K. Sugawara, N. Shiratori, Multiagent-based flexible edge computing architecture for iot, IEEE Network 32 (1) (2018) 16–23, https://doi.org/10.1109/MNET.2018.1700201.
[24] X. Ferré, N. Juristo, H. Windl, L. Constantine, Usability basics for software developers, IEEE Softw. 18 (1) (2001) 22–29, https://doi.org/10.1109/52.903160.
[25] A. Holzinger, Usability engineering methods for software developers, Commun. ACM 48 (1) (2005) 71–74, https://doi.org/10.1145/1039539.1039541 URL: http://doi.acm.org/10.1145/1039539.1039541.
[26] K. Finstad, The usability metric for user experience, Interacting with Computers 22 (5) (2010) 323–327, modelling user experience - An agenda for research and practice. https://doi.org/10.1016/j.intcom.2010.04.004. URL: <http://www.sciencedirect.com/science/article/pii/S095354381000038X>.
[27] D. Guinard, V. Trifa, E. Wilde, A resource oriented architecture for the web of things, Internet of Things (IOT) 2010 (2010) 1–8, https://doi.org/10.1109/IOT.2010.5678452.
[28] S.K. Datta, R.P.F. Da Costa, C. Bonnet, Resource discovery in internet of things: Current trends and future standardization aspects, in: Proceedings of the 2015 IEEE 2Nd World Forum on Internet of Things (WF-IoT), WF-IOT '15, IEEE Computer Society, Washington, DC, USA, 2015, pp. 542–547. https://doi.org/10.1109/WF-IoT.2015.7389112.
[29] M. Wooldridge, N.R. Jennings, Intelligent agents: theory and practice, Knowledge Eng. Rev. 10 (2) (1995) 115–152, https://doi.org/10.1017/S0269888900008122.
[30] J.-P.A. Barthès, Omas — a flexible multi-agent environment for cscwd, Future Gener. Comput. Syst. 27 (1) (2011) 78–87.
[31] G. Fortino, R. Gravina, W. Russo, C. Savaglio, Modeling and simulating internet-of-things systems: a hybrid agent-oriented approach, Comput. Sci. Eng. 19 (5) (2017) 68–76, https://doi.org/10.1109/MCSE.2017.3421541.
[32] T. Kinoshita, K. Sugawara, Adips framework for flexible distributed systems, in: T. Ishida (Ed.), Multiagent Platforms, Springer, Berlin Heidelberg, Berlin, Heidelberg, 1999, pp. 18–32.
[33] T. Finin, R. Fritzson, D. McKay, R. McEntire, Kqml as an agent communication language, Proceedings of the Third International Conference on Information and Knowledge Management, CIKM '94, ACM, New York, NY, USA, 1994, pp. 456–463, , https://doi.org/10.1145/191246.191322.
[34] Y. Kaeri, Y. Manabe, K. Sugawara, A platform for prototyping an agent space interacting with real space and digital space, in: The 2nd International Workshop on Smart Technologies for Energy, Information and Communication (IW-STEIC), 2013, pp. 113–120.
[35] P.T. Eugster, P.A. Felber, R. Guerraoui, A.-M. Kermarrec, The many faces of publish/subscribe, ACM Comput. Surv. 35 (2) (2003) 114–131.
[36] N. Shiratori, K. Takahashi, K. Sugawara, T. Kinoshita, Using artificial intelligence in communication system design, IEEE Softw. 9 (1) (1992) 38–46, https://doi.org/10.1109/52.108779.
[37] A. Jones, A. Kendira, T. Gidel, C. Moulin, D. Lenne, J.-P. Barthès, Personal assistant agents and multi-agent middleware for cscw, 16th IEEE International Conference on Computer Supported Cooperative Work in Design (CSCWD), Wuhan, China, 2012, pp. 72–79.
[38] P.D. O'Brien, R.C. Nicol, Fipa—towards a standard for software agents, BT Technol. J. 16 (3) (1998) 51–59.