A Multi-Agent-Based Management System for Pervasive Collaborative Computing Environment

Sang Woo Han, JongWon Kim
Dept. of Information and Communications
Gwangju Institute of Science and Technology
Gwangju, Korea
{swhan, jongwon}@gist.ac.kr

Wolfgang Prinz
Fraunhofer FIT
Schloss Birlinghoven
53754 Sankt Augustin, Germany
wolfgang.prinz@fit.fraunhofer.de

Abstract

To support advanced collaboration among knowledge workers distributed geographically, there have been extensive researches under the scope of pervasive computing environment. Especially, to cope with difficult operation of traditional room-based collaboration environment, several conceptual frameworks are designed and prototyped. For easy operation, in this paper, we design a management system for multi-party collaboration environment. The proposed system manages a set of distributed software components belonging to meeting nodes by user-generated specification for a common collaborative task. Based on the proposed design, we implement a multi-agent-based management toolkit, which supports software agents to abstract the functionalities of meeting nodes, services, and tools and to configure and operate each collaboration nodes. Finally, we demonstrate how to operate meeting nodes according to user-generated specification for multi-party video conferencing.

1. Introduction

High-performance computing and high-speed networking technology has been promoting the development and deployment of advanced collaboration systems among industry and research institutes. Due to the specialized needs of each area, research-oriented and commercial-based multimedia collaboration systems have been released for public use in various fields such as scientific co-experiment, business meeting, employee training, and distance learning. For example, Cisco’s TelePresence and HP’s Halo support realistic (e.g., enabling eye-contact) remote meetings among distributed knowledge workers employing large-scale displays and high-quality video cameras. Also, ANL (Argonne National Laboratory) Access Grid and Microsoft Research’s ConferenceXP have proposed the designs of meeting node configuration with multicast-connected multiple hosts, so that users can easily develop and deploy customized services in order to build extended collaboration environments.

As an effort for more practical and interactive collaboration, the research described in [1] has built a multi-party interactive collaboration environment named as SMeet (Smart Meeting Space). The main features of the SMeet prototype system are a tiled display service efficiently visualizing high-resolution media data, a pointing service controlling visualization media at a near distance, and a hand-motion tracking service operating 3-dimensional medical contents. But, since this prototype consists of numerous hardware and software components, it is hard for unskilled users to manually operate all of them. To enable multi-party collaboration more conveniently, we should consider how to ease the configuration and use of distributed components for user-intended collaboration works.

Thus, in this paper, we design a management system, in order to easily configure those pervasive collaboration nodes to organize multi-party collaboration environments. The proposed system flexibly combines various networked devices and distributed tools to construct customized meeting nodes. The specific contributions of this paper are: 1) We introduce an architectural design of a management system for multi-party collaboration environment, which aims to dynamically organize meeting nodes according to user-generated specification. 2) We also present a prototype of the proposed system, which is based on JADE (Java Agent DEvelopment framework) [2]. To verify the result, we deploy it with a designated specification over
SMMeet testbed [3] and show the feasibility of intended operation for multi-party video conferencing.

The rest of this paper is organized as follows. System architecture is proposed in Section 2, and the software implementation and its verification are presented in Section 3. In Section 4, we discuss related work in the architectural aspect. Finally concluding remarks are given in Section 5.

2. System Architecture

In this section, we discuss some architectural requirements for multi-party collaboration environment, and propose the design of a multi-agent-based management system to satisfy these requirements.

2.1. Architectural Approach

We need to carefully consider architectural differences when adopting existing research works (i.e., general-purpose middlewares for pervasive computing environment) to the multi-party collaboration domain. To flexibly organize meeting nodes with heterogeneous components by considering different purposes and preferences, extensive research provides programming methods to construct meeting nodes. However, as meeting nodes have numerous hardware and software, node operators must describe highly sophisticated applications to configure and to compose numerous tools.

To ease this difficulty, we attempt to design a hierarchical abstraction structure for programmable collaboration environment, inspired from CDPS. The proposed structure can flexibly organize the meeting node from the viewpoints of users by selectively composing services and the associated tools. In detail, group collaboration tasks, describing how to allocate functional tasks to participating meeting nodes and how to interwork cooperatively, are specified by high-level descriptions. Participating meeting nodes decompose each group collaboration task into specific subtasks (e.g., media encoding/decoding, media transmission over IP networks, 2D/3D visualization data rendering on display devices, and pointing/hand-motion based user interaction). Based on the predefined service code matched to each subtask, services selectively use associated tools working on networked devices of a meeting room and performs each given subtask. Note that legacy tools (e.g., audio/video streaming tools, 3D graphics rendering tools, and file sharing tools) having proprietary interfaces are abstracted, in order to expose the tool functions via standardized interfaces. Without understanding of tools, node operators can easily construct task-oriented meeting nodes by using high-level specifications to combine distributed software components for user-intended collaboration environment.

2.2. Meeting Node Structure

To ease the management of multi-party collaboration environments, we suggest an agent-based meeting node structure that includes node mediator, service agents, device agents, and tool agents. Fig. 1 illustrates the structure.

Node mediator is a control point to enforce group collaboration tasks in a meeting node. The specific functions of node mediator consist of three parts that includes node registry, agent matchmaking, and connection management. Note that these functions are released through node access API presented in Table 1.

Table 1. Node access API.

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodes</td>
<td></td>
<td>Get a list of participating nodes</td>
</tr>
<tr>
<td>startup</td>
<td>properties</td>
<td>Launch an agent matching with the given properties</td>
</tr>
<tr>
<td>shutdown</td>
<td>agent ID</td>
<td>Shutdown an agent having the given agent ID</td>
</tr>
<tr>
<td>search</td>
<td>node ID, service ID, properties</td>
<td>Search agent(s) having given properties</td>
</tr>
<tr>
<td>request</td>
<td>agent ID, a request message</td>
<td>Send an agent request message to call a interface</td>
</tr>
</tbody>
</table>

Node registry aggregates meeting node context that is characterized by general node information (e.g.,
node name, virtual meeting place, and contact information), the state information of available service/device/tool agents (e.g., agent identifier, agent properties, and operational stage), and public data to be shared by all participating nodes. Agent matchmaking discovers the most appropriate tool agent(s) that can satisfy service-specific requirements. Note that each tool agent has functional properties\(^3\) and non-functional properties\(^4\), called as tool properties. Consider the tool properties, agent matchmaking can be effectively conducted by extending matchmaking algorithms\(^5\) in order to find the best matched tool agent(s). At this stage, we have simply implemented agent matchmaking by using exact matching of tool properties. Finally, connection management handles multi-party communications among meeting nodes participating in a common collaboration session. A new node contracts a group collaboration task with participating nodes by the following request-response protocol. A new node mediator sends JOIN_REQUEST message to participating nodes through centralized conference membership control\(^6\) or P2P-based node lookup mechanisms (e.g., using JXTA). If the new node can support service(s) for an ongoing collaboration, the existing nodes permit its participation and return ACCEPT message. Finally, they add the new node name into the list of participants and allow the new node to access their service agents.

**Service agent** discovers, configures, and executes tool agents by using node access API, in order to perform the requested subtask. To cope with the subtask, the service agent carries out the predefined service code of how involved tools work with each other having mutually dependent functionalities. Based on the predefined service code, the service agent\(^5\) coordinates tool agents for actual interface binding and inter-working procedures. As meeting nodes may have heterogeneous capabilities (e.g., available collaboration tools), the service code has to be instantiated by selective use of the most appropriate tools among available tools. For this, the service code should be described with tool properties demanded by the service. Note that capacity of a meeting node is characterized by different capability (e.g., processing power, network bandwidth, and available tools) as well as time-varying system performance. To support services continuously in such a situation, it is recommended that the service code be designed by using strategy-based adaptation algorithms\(^7\). However, designing the adaptation algorithms is out of scope for this paper.

**Device agent** controls each device holding versatile collaboration tools. Working on a device, a device agent registers the device as a manageable device in the meeting node and registers available tool agents into the node registry. As requested by the node mediator, the device agent opens or closes selected tool agents. It also monitors whether tool agents are normally working or not.

**Tool agent** wraps a tool to provide elementary and unique function to assist users in doing collaborative works. For a versatile meeting node, a variety of open-source collaboration tools having different capabilities should be easily deployed and be manageable. As the tool agents already know the predefined tool properties such as role, type and proprietary execution commands, they can response when the service agents request to access tools identified by given conditions. Thus, the tool agents basically allow other components (i.e., node mediator, service agents, and tool agents) to access and control useful functions of tools over networks, although the actual access and control may be restricted by the allowed privileges.

### 2.3. User-generated Specifications

The management system provides description methods to specify service composition among participating nodes (described by composition pattern), tool composition in a meeting node (described by service code), and tool abstraction (by tool descriptor) over hierarchical abstraction structure of meeting nodes. With these descriptions, meeting nodes combine composable resources (i.e., service agents and tool agents) to achieve user-intended group collaboration.

**Composition Pattern:** The composition pattern is a template that is logically described in order to compose service agents having capabilities to perform specific functions (i.e., subtasks) for a group collaboration task between meeting nodes. Based on the composition pattern, the node mediators discover the service agents that can be used in their meeting nodes and delegate the responsibilities of subtasks to them. The specification of the composition pattern comprises three parts: **identifier** identifying the group collaboration task, **service agents** to be launched, **DAG (directed acyclic graph)** describing the functional dependency graph. The functional dependency graph includes a set of logical paths that

---

\(^3\) Functional properties represent the description of the tool actions in terms of tool operation (e.g., execution command) and functional description (e.g., input/output data format).

\(^4\) Non-functional properties describe the tool characteristics (e.g., reliability, availability, and interoperability) that are not directly related to the tool functionality.

\(^5\) To naturally communicate with tool agents, service agents can use agent interaction protocols and service-specific context ontology (e.g., media player ontology, network monitoring tool ontology, and so forth).
indicate the service agents to send requests (described as sources) and other service agents to response to the requests (described as destinations).

*Service Code:* The service code is user-generated program for a service agent, which is primarily used to employ the tool agents of a meeting node directly with the purpose of supporting a given subtask. Thus, the service agent tailors the meeting node to meet the users’ specific needs. To express procedural operations of tool agents, the service code is written by Java and node access API. The structure of the service code consists of three distinct parts:

- **Initialization:** Tool agents that are necessary to perform a subtask are declared by calling `startup` method. By this declaration, the node mediator can know the tool agents to be executed at start-up time.
- **Main procedure:** The main procedure sequentially invokes the selective interfaces of tool agents according to service-specific business logic. For more convenient service programming, node access API allows to search (by `search`) and invoke (by `request`) the interfaces of the tool agents. Also, we can interact (e.g., inform, query, or request) with the tool agents of remote nodes (discovered by `nodes`) in doing the task.
- **Shutdown:** Before ending the subtask, all the running tool agents have to be completely closed (by `shutdown`).

*Tool Descriptor:* The tool agent should easily accommodate selected variety of (open-source) collaboration tools having different capabilities and proprietary interfaces. The tool agent needs to understand tool properties, executable options, and command line interfaces to execute its tool according to pre-defined configuration. Thus, a tool descriptor includes 1) the basic properties that include `role` meaning whether the tool is producing or consuming contents, `type` specifying the kind of content format, and `max` representing the permissible number of processes of the tool and 2) the executable configuration that describes tool-dependent information such as execution command, working directory, and optional parameters.

### 3. Implementation Results

To explore the possibility of the proposed architecture, we are currently developing the multi-agent-based management system for SMeet, based on JADE (Java Agent DEvelopment Framework) multi-agent middleware [2]. Currently the system comprises about 13K source line of code (SLOC) and is being tested under SMeet prototype. All the agents (i.e., node mediator, service agents, and tool agents) of a SMeet node are using FIPA-compliant agent communication protocol [8] and can semantically communicate by using ontology-based context description. Now we have preliminarily designed node ontology for agent operation in a meeting node and space ontology for multi-party service composition between meeting nodes.

For the verification, we have described sample specifications for realizing multi-party video conferencing over SMeet testbed as follows:

- **Composition pattern:** Fig. 2 shows an example to present a task of multi-party video conferencing. Node mediators first launch service agents identified as `smeet.service.media`, a Java class to provide media service. They then connect the media service agents belonging to participating meeting nodes together according to `<PATH>` of `<DIRECTED_ACYCLIC_GRAPH>`.

- **Service code:** Fig. 3 describes a sample service code of the media service. In `init()`, we declares a start-up of the tool agents for video receiving and video display. We also register an event, `RECEIVE` to be arisen when users want to receive media. Corresponding to these events, event handlers are also written with the interfaces of service agents. For example, in `receive()`, this service code requests the receiving tool to decode video and then requests the display tool to show video. In `shutdown()`, we have to declare to shutdown running agents when the service code closes.

- **Tool descriptor:** Fig. 4 presents a sample tool descriptor about VLC media player as a video receiving tool. `<BASIC_PROPERTY>` indicates that this tool can receive video, and `<CONFIGURATION>` specifies the details of execution commands such as file name, working directory, and executable options.

```
<COMPOSITION_PATTERN name="multi-party video conferencing">
  <SERVICE_LAUNCH>
    <SERVICE name="smeet.service.media"/>
  </SERVICE_LAUNCH>
  <DIRECTED_ACYCLIC_GRAPH>
    <PATH src="smeet.service.media" dst="smeet.service.media"/>
  </DIRECTED_ACYCLIC_GRAPH>
</COMPOSITION_PATTERN>
```

Figure 2. Composition pattern.

We now describe the details of the operation steps that mainly include two steps:

1. **Organize a meeting node.** To organize a meeting node, a node mediator (to manage a meeting node) and
device agents (to manage devices under the control of the node mediator) must be executed. Then, the device agents will enroll their functional capabilities such as services and tools to be supported in the node mediator. The enrolled information is immediately presented in node mediator GUI depicted in Fig. 5. In the tab, denoted as my meeting room, of the GUI, node operators who operate a meeting node are allowed to manually add, start, stop, and remove agents.

2) Begin collaborating with participating nodes. For multi-party collaboration, all meeting nodes have to join a common collaboration session that has a composition pattern to describe functional dependency among participating nodes and participating node information (e.g., node names, contact information, and the addresses of node mediators). In this prototype, we provide session directory agents to hold collaboration sessions like centralized virtual venues. To connect a session directory agent, node operators have to enter the IP address of the session directory agent to connect in the tab, denoted as interactive meeting space, of the GUI. Then, node operators can watch a list of participating nodes and select a composition pattern. Here, video conference is selected. Then, node mediators begin to combine service agents based on the selected composition pattern, and the service agents also run tool agents in accordance with their predefined service code.

4. Related Work

In this section, we introduce related work in conceptual framework, software infrastructure, and middleware to support system management and application development in pervasive computing environment.

Interactive Room Operating System (iROS) is a meta-OS used for iRoom in order to tie together all devices in the level of low-level OS [9]. iROS enables interactions between devices placed in the local physical space by supporting three subsystems such as shared data storage (Data Heap), service advertisement and information (iCreafter), and event notification (Event Heap). Using these subsystems in iROS, developers can implement a couple of user modalities of moving data, moving control, and dynamic application coordination in iRoom.

Aura architectural framework is to enable mobile users to seamlessly perform user tasks, represented as the union of distributed services (named as Service Suppliers), in mobile computing environment [10]. Generally, when users are moving, they want to seamlessly do ongoing tasks. Environment always watches task performance by task monitors and renegotiate task support in the presence of resource and capability variations. In Aura, Prism infrastructure supports interactions between all components, built on top of distributed communication middleware such as CORBA or RPC (remote procedure call).

Task computing enables a user to compose entities (e.g., applications, devices, and contents) and execute complex tasks in pervasive computing environment [11]. Task computing provides semantic service description to abstract the functionalities of computing entities and middleware to discover, compose, execute, and manage these services based on user-defined task.
For the purpose of services, users represent tasks by a variety of clients.

In GAIA, an application framework for pervasive computing applications has been developed, which is a distributed component-based software infrastructure [12]. This framework abstracts devices and applications in Active Space as software components. Using MPCC (model, presentation, controller, and coordinator) model that extends MVC (model, view, and controller) of object oriented programming, this application framework enables developers to easily combine the software components in order to create new application in Active Space. GAIA also provides another framework (named as Olympus) to abstract physical entities and allows programmers to develop applications for Active Space configurations and operations through high-level API [13]. Olympus framework provides discovery process which chooses the most suitable entities for given search conditions. For this, this framework first discovers all possible entities (i.e., matching with the search conditions) and then calculates the best entities providing higher utility for user applications.

5. Conclusion

In this paper, we have explained the design of a management system to flexible organize multi-party collaboration environment and shown the implementation result. The proposed system has suggested the architectural design of meeting nodes that are comprise of software agents (i.e., node mediator, service agents, and tool agents) so that meeting nodes can be customized by selectively using services and tools based on group collaboration tasks. To explore a possibility of the proposed system, we have developed the prototype based on JADE multi-agent system. Using this, we have shown its operations for realizing multi-party video conferencing over SMeet testbed. In the future, we plan to improve the proposed system into workflow-based management framework in order to cope with dynamical change of meeting nodes during group collaboration. Thereby, we would realize rich and comfortable collaboration work environment with minimizing users’ efforts.

Acknowledgment

This work was supported in part by the BK21 global internship program of Korea Research Foundation (KRF) and the basic research project through a grant provided by the Gwangju Institute of Science and Technology in 2008.

References


