




Article

Human-Centered Dynamic Service Scheduling Approach in Multi-Agent Environments

Yunseo Jung ^{1,2}, Hyunju Kim ³ , Kyung-Duk Suh ^{2,4}  and Jung-Min Park ^{2,*} 

¹ Department of Mechanical & Biomedical Engineering, Ewha Womans University, 52, Ewhayodae-gil, Seodaemun-gu, Seoul 03760, Korea

² Center for Intelligent & Interactive Robotics, Korea Institute of Science and Technology, 5, Hwarang-ro 14-gil, Seongbuk-gu, Seoul 02792, Korea

³ Department of Information Science, Cornell University, Ithaca, NY 14853-0099, USA

⁴ Department of Electrical & Computer Engineering (Control, Robotics and System), Korea University, 145, Anam-ro, Seongbuk-gu, Seoul 01841, Korea

* Correspondence: pjm@kist.re.kr

Abstract: As robots become more versatile and combined with a variety of Internet-of-Things technologies, they will be able to serve humans in their daily environments. To provide services by satisfying various human requests, several robots must take turns performing a series of tasks that constitute the service. Because the order of service delivery may differ according to user requests, sequential interdependencies between tasks should be considered. Therefore, we propose a dynamic service scheduler consisting of dynamic sequencing and allocation that can handle scheduling of tasks with user requests such as prioritizing certain tasks or actively changing their order in a multi-agent environment. We experimented with the proposed method in four situation scenarios by building a virtual reality smart office consisting of multiple robots with a robot arm, mobile robot, and smart lamp. The results demonstrated the feasibility and effectiveness of the proposed approach by satisfying the user requirements in different situations. The proposed approach constitutes a basis for further development of efficient in-office and at-home multi-agent environments.

Keywords: service scheduling; multiple agent; human-centered approach



Citation: Jung, Y.; Kim, H.; Suh, K.-D.; Park, J.-M.

Human-Centered Dynamic Service Scheduling Approach in Multi-Agent Environments. *Appl. Sci.* **2022**, *12*, 10850. <https://doi.org/10.3390/app122110850>

Academic Editors: Augusto Ferrante, Mihaiela Iliescu, Mingcong Deng and Tai Yang

Received: 20 September 2022

Accepted: 25 October 2022

Published: 26 October 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Service robots capable of assisting humans in their environments [1] are rapidly being developed to provide services such as household chores, in-home delivery, healthcare, and assistance in offices, hotels, hospitals, and educational environments [2,3]. Combining robotics with innovations in computer vision, sensors, and the Internet of Things (IoT) enables robots to interpret and monitor real-world settings contextually. Improvements in speech recognition and artificial intelligence facilitate intuitive human interactions with robots [4,5]. Accordingly, the application of human–robot interaction technology is gradually expanding to the daily service industry, and research on robots that provide highly complex and personalized services to users is in progress [6].

To provide services to users using service robots in daily life, various challenges need to be addressed. Because service robots should be able to accept and complete the user requests, service tasks must be appropriately scheduled so that they are allocated to a suitable robot and executed according to the user requirement. However, existing research on robot scheduling does not reflect these characteristics because human multi-robot system interaction was first introduced in the industry to reduce the workload of human workers and increase the productivity of manufacturing processes. In terms of efficiency, the parallel execution and synchronization of robots [7] have been a major concern for multi-robot systems. We thoroughly studied how the overall workload is distributed to ensure that the work is performed quickly. For example, Refs. [8,9] proposed a scheduling system that

dynamically changes the task distribution according to the variability of human workers so that multiple robots can safely and efficiently collaborate with humans. These studies focused on enabling industrial robots deployed in highly structured and well-controlled environments to work as efficiently as possible. Service robots, however, must be able to operate in a dynamic environment to satisfy the continuous and varying service requests of humans [1].

Moreover, humans have complex requirements for the services that they want to receive. These requirements typically include complex procedures (e.g., priority of service delivery) and guidelines that the robot should follow [10]. This results in services with sequential interdependencies. Tasks with multiple sequences are common in both natural and artificial systems. They must be consecutively completed to properly perform the entire task. Therefore, a single service can contain multiple individual subtasks that must be performed sequentially. For example, serving coffee in a kitchen may involve two separate subtasks: pouring coffee into a cup and delivering the cup to the user. If each subtask is performed by a different robot, the second robot must wait for the first robot to finish pouring coffee into the cup. In addition, we can consider a service that delivers coffee to a specific cup that the user wants. In this case, the robot should be able to bring a specific cup into the kitchen and then proceed with coffee serving. This implies that the detailed requirements of the user significantly change the sequential interdependence of the entire service. For this reason, multiple robots in a service environment need to collaborate, considering the correlation between services to fully accommodate the needs of users. This implies that the service robot must be able to dynamically accept user requests and provide services accordingly. However, current multi-robot systems do not consider the sequential relationship between services that occur according to the user's request, resulting in performance degradation or failure to fulfill. A given service can be divided into multiple sequential subtasks, each of which can vary in interdependencies owing to dynamic user requests, which are addressed by coordinating several heterogeneous robots with different capabilities. Therefore, for effective deployment of service robots, a systematic scheduling framework that can dynamically adjust and manage schedules according to user requests is required.

In this paper, we propose a novel scheduling framework in which multiple robots can effectively provide services through collaboration in response to dynamic requests of the users. The main contributions of this study are as follows.

1. We present a scheduler in which multiple heterogeneous agents perform services with sequential interdependencies.
2. The proposed scheduler can accept complex and dynamic user requests.

The remainder of this paper is organized as follows. Section 2 presents the related work, and Section 3 explains the proposed scheduling scheme. Section 4 describes several simulated scenarios and analyzes the results to evaluate the effectiveness of the scheduler. Finally, Section 5 concludes the paper.

2. Related Work

Task scheduling is important for the effective service coordination of multiple service robots. So far, multi-agent task scheduling has been mainly studied in industrial robots, as multiple agents introduce parallel implementation into the workflow enabling tightly organized executions and thus promote efficiency and minimal cycle times (see, e.g., [7,9,11,12]). Many studies on industrial robots have only considered the presence of homogeneous agents [11] because industrial robots operate in a specific setting (e.g., assembly lines [7–9,13,14] and warehouse operations) to maximize productivity and efficiency. However, service robots have diverse forms, functions, and abilities [4,10]. Even if the same task is presented, the time and prerequisites required to complete the task vary depending on the type of robot responsible. The main purpose of a service robot is to satisfy service requests of a user. Therefore, it is important to properly incorporate agent heterogeneous variables, such as the ability of each agent and duration of task scheduling. To achieve this, first, a

given task must be modeled to recognize how the entire task is performed in terms of task component relationships and agent dependencies. The task model describes resources, task sequences, and procedural knowledge of tasks [15]. With an appropriate task model, agent heterogeneity can be introduced by mapping agent variables to each task component. A variety of useful task-modeling tools are available for designing and modeling specific processes. Many scheduling methods utilize precedence relationships [16–18], temporal constraint networks [19], hierarchical task decompositions (such as AND/OR trees [7,9,20]), and Petri nets [9,12], especially because they directly encode parallel implementations. Some studies considered individual agent capabilities and obtained the final schedule based on capability indicators which evaluate the extent to which an agent is suitable for a specific execution [13,16]. The decision-making algorithms are often based on a multi-criteria approach using a cost function. Frequently used optimization methods for selecting the optimal schedule include tree search algorithms [7], genetic algorithms [17,21], and mixed-integer linear programming [18]. Ref. [7] proposed a hierarchical framework and used an A* graph search algorithm to generate an optimal task sequence for agents in human–robot assembly. Ref. [18] found a final schedule using variants of mixed-integer linear programming in an optimization framework that generates task assignments and schedules for a human–robot team to improve both time and ergonomics. However, these studies do not consider system variability but primarily pursue optimization regarding ergonomics or cost-effectiveness. They lead to fixed schedules and do not allow flexibility for spontaneous decisions.

Dynamic task allocation for heterogeneous agents makes it possible to cope with systemic inconsistencies better than a single fixed schedule. Various methods have been proposed to estimate the duration time of robots (e.g., methods-time measurement strategy [14,22]). More recently, [8] used a directed acyclic graph to represent the dependency relationship between tasks and introduced a delay-predicting strategy that predicts human duration and adjusts the robots' schedules accordingly. Similarly, Refs. [9,20] applied an AND/OR tree to specify all asymmetric actions that an agent can perform in a single graph to enable just-in-time rescheduling of the assembly process according to agent volatility and robotic failures. Ref. [20] suggested an online perception-simulation planning framework to dynamically allocate tasks to robots or humans by always monitoring the manufacturing process. Ref. [12] defined time Petri nets for specific actions or part transports to handle multiple executions. The proposed adaptive scheduling scheme predicts the future evolution of a system based on the agent's previous performance. However, the above literature requires precise knowledge of the progress of a task at a specific point in time. This type of full observability can only be achieved by continuously monitoring the agents' actions or the state of the workspace with a wide range of sensor settings. Moreover, such approaches primarily rely on manually specified task descriptions for the deployment of static jobs and do not consider incorporating new incoming tasks into the system and dynamic changes in the execution order of tasks. Alternatively, Ref. [11] uses the largest total amount of processing time first strategy and schedules with a tabu-search algorithm to minimize the makespan of the jobs with tree-formed precedence constraints. However, the study did not consider the agent heterogeneity. The service robot must be able to accept new requests from the user and create a newly ordered task model in real time. Therefore, task scheduling for service robots must embody dynamic task sequencing (modeling) and dynamic task allocation. Ref. [23] suggests a heuristic algorithm based on a greedy for dynamic scheduling for tasks. It reduced the total workload but could not adjust the specific timing and task sequency.

Recently, studies have been conducted on service robots used in smart homes. A knowledge-based framework for object search [24] and ontology-based smart home architecture [25,26] have been proposed. Probabilistic inference has been applied to hybrid task planning to address task failures [27]. These studies combined intelligent space and semantic task models to dynamically create and sequence a new task but focused on

context-aware services and did not address the scheduling aspect. In addition, all the executions were dedicated to a single robotic agent.

In summary, the limitations of the previous robot task scheduling approach are as follows. (1) Existing methods determine an optimal schedule based on maximum productivity, but service robots must prioritize user demands and specifications over efficiency. (2) Only environmental variabilities such as delays and robotic failures are considered, and the user activity dynamics, such as user requests, are not considered. (3) They require precise knowledge of the overall progress via constant monitoring. (4) The previous robot task scheduling method could not accept new tasks or user requirements and build newly sequenced task models in real time. Therefore, we believe that a new approach to task scheduling for multiple service robots is essential to ensure the accessibility of service robots in daily environments. We propose a systematic scheduling framework that can dynamically adjust and manage a schedule according to user requests for the effective provision of services.

3. Human-Centered Dynamic Service Scheduling Approach

3.1. Proposed Scheduling Framework Overview

Humans can receive services from one or more agents (service providers) of different types and functions in their daily lives. There are two types of service providers. An active agent can perform a variety of tasks through their own processes with distinct mechanisms, whereas a passive agent is limited to specific processes, such as turning a smart lamp on and off in a home management system. In this study, both are referred to as agents and can share and collaborate on a given task according to their capabilities. These capabilities are assumed to be predetermined and stored in a database so that the scheduler can distribute tasks accordingly.

When a user requests a service, it is set as a single job. The job is then divided into an ordered set of operations, where each operation is characterized by its numbered sequence and the agents that can execute it. In this study, we define an operation as an essential action that is completed independently by a single agent. Thus, a job can be performed by multiple agents one after another completing sequential operations.

A user may have various service requirements, and each job may have additional constraints on when it should be provided to the user. A user may request that a specific service be provided before or after a different service, or only when a specific condition is met. For example, a user can ask for a meal and coffee to be delivered simultaneously or for coffee to be delivered first. Therefore, services must be provided in a manner that satisfies these requirements.

To provide the requested services, an appropriate agent must perform the task while satisfying the constraints requested by the user. That is, the job is decomposed into operations that each agent can perform, and these operations must be performed by the appropriate agent based on the agent's capability and duration while satisfying the constraints requested by the user.

In this study, a scheduling framework is proposed to allow multiple agents to properly perform the services requested by the user while satisfying dynamic requests (Figure 1). The proposed scheduling framework comprises three main components.

1. Job Decomposer: It decomposes the job requested by a user into a list of operations that an agent can perform.
2. Operation Manager: It sequences and organizes the operations using operation queues.
3. Operation Allocator: It allocates an operation to an appropriate agent.

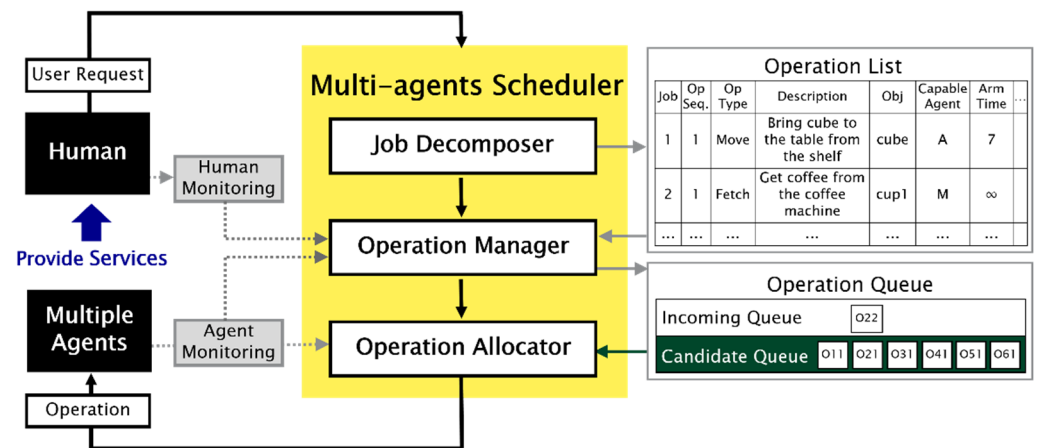


Figure 1. Diagram of the proposed scheduler framework.

3.2. Multi-Agent Scheduler

3.2.1. Job Decomposer

The service is independent and has no correlation with other services. However, the service job requested by a user may have specific requirements regarding when it should be provided. We categorized these additional user requirements as follows:

- Perform the service first;
- Perform the service last;
- Perform the service in a specific order;
- Perform multiple services simultaneously;
- Perform the service only when specific conditions are met.

These additional requirements contribute to sequential interdependencies among different services. As we have defined a service as a single job, a job may contain a certain sequence of activities and prerequisites, detailing how it should be delivered. The activities are transcribed into operations, and prerequisites are stored for a specific time and situational condition that can be followed by multiple agents. For agents to successfully provide services, a job must be translated into an executable one, considering the agents' capabilities and action units. We divide a job into operations, which are a collection of essential actions completed independently by a single agent. This allows multiple agents to perform a job by completing sequential operations. For example, when a person asks for coffee, the job "Bring coffee" can be divided into two operations: "Get coffee from the coffee machine" and "Bring coffee to the table". Each operation has its own order because bringing coffee should only be executed after getting coffee is complete.

The job decomposer decomposes the requested jobs into operations and creates an operation list for user requests. When the user requests M jobs for the agents to perform, each job J_1, J_2, \dots, J_M is decomposed into a sequence of one or more operations. For example, job J_1 is decomposed into $(O_{11}, O_{12}, \dots, O_{1n})$. We assumed that there is a job-to-operation database for properly mapping jobs to operations. The operation list (Table 1) consists of variables such as the operation type, object, capable agents, and the agent's operation completion time, etc. Operation types can be "fetch", "drop", or "move" etc. The object is the target of the operation. The start and end positions are where the operation starts and ends, respectively, and information is obtained from the target object. A capable agent lists the agents that can perform the operation. The duration of each agent is a predetermined nominal number and represents the execution time of the operation. If the agent is unable to perform the operation, the duration time is set as infinite. As previously described, the job decomposer is presented as pseudocode in Algorithm 1.

Algorithm 1: JobDecomposer()

```

//Given Variables
 $J_M = \{j_1, j_2, \dots, j_i, \dots, j_M\}$  //Set of Requested Jobs
 $A = \{a_1, a_2, a_3\}$  //Set of Agents
//Initialization
 $O_i$  //Set of Operations of Job  $j_i$ 
 $R_i$  //User Requirement values of Job  $j_i$ 
 $C_{ij,ak}$  //Capability of Agent  $a_k$  performing Operation  $o_{ij}$ 
 $d_{ij,ak}$  //Duration time of Agent  $a_k$  performing Operation  $o_{ij}$ 
 $C_i = \{C_{i1}, C_{i2}, \dots, C_{ij}, \dots, C_{iN}\}$  where  $C_{ij} = [C_{ij,a1}, C_{ij,a2}, C_{ij,a3}]$ 
 $D_i = \{D_{i1}, D_{i2}, \dots, D_{ij}, \dots, D_{iN}\}$  where  $D_{ij} = [d_{ij,a1}, d_{ij,a2}, d_{ij,a3}]$ 
if New Jobs ( $J_M$ ) are Requested then
    for  $i = 1$  to  $M$  do
         $O_i \leftarrow$  JobDecomposer( $j_i$ )
         $R_i \leftarrow$  getUserRequirements( $O_i$ )
        ( $C_i, D_i$ )  $\leftarrow$  getNominalValues( $O_i, A$ )
        ( $O, R, C, D$ )  $\leftarrow$  pushback( $O_i, R_i, C_i, D_i$ )
         $i \leftarrow i + 1$ 
    end
end

```

Table 1. Example of an operation list.

Job	Op Seq.	Op Type	Description	Obj	Capable Agent	Arm Time	Mobile Time	Light Time
Bring cube	1	Move	Bring cube to the table from the shelf	Cube	A	75	∞	∞
Bring coffee	1	Fetch	Get coffee from the coffee machine	Cup1	M	∞	50	∞
	2	Move	Bring coffee to the table from the coffee machine	Cup1	A, M	55	40	∞
Bring water	1	Fetch	Bring water to the table	Bottle	A, M	50	35	∞
Take back empty cup	1	Fetch	Take empty cup from the table	Cup2	A, M	50	50	∞
Throw away trash	1	Move	Take trash from the table and throw it away in the bin	Trash	M	∞	50	∞
Turn on light	1	Turn on	Turn on light	Lamp	L	∞	∞	5

Managing user requests of service with additional requirements and complicated user requirements are addressed in the next section.

3.2.2. Operation Manager

The operation manager is responsible for carefully sequencing the order of operations based on the operation list created by the job composer. The operation manager also reorganizes the order of operations by considering the user requirements. In environments

in which multiple agents perform sequential operations simultaneously, it is important to consider the relationships among operations in terms of overall execution. This implies that the scheduler must monitor all executions to avoid possible failures. The order of operations determined by the operation manager affects how agents are utilized. For example, delivering coffee to a person involves two separate and sequential operations: pouring coffee into a cup and delivering the cup full of coffee to the person. Two different agents may be assigned to perform each operation; therefore, to avoid failure to complete the job, it is very important that the second agent does not deliver the cup before the first pours the coffee into the cup.

To monitor these sequential correlations between operations, the operations are managed by queue types to prevent them from mixing. To sequence and organize operations from the original order in which jobs were requested, we created two types of operation queues: candidate and incoming. The candidate queue represents operations that can be executed immediately and do not require waiting for other existing operations. By contrast, the incoming queue is a bundle of subsequent operations that must be held waiting until the previous queue is executed or user-specific conditions are met. The candidate queue is created by enqueueing the first operation of each job in the order in which they are requested. The remaining operations are then stacked on top of the candidate queue and named the incoming queue because they are waiting to enter the candidate queue.

The operation manager also rearranges the queues to be rescheduled according to the dynamic requests of a user. If a job is canceled before execution, all related operations are removed from the candidate and incoming queues. The user requirement for the order of service is resolved by three functions of the operation manager that adjust the stacked queue: freezing, stacking, and combining.

The freezing function is used when a user wants the service to be provided first or last. To perform the job first, the remaining jobs are frozen until the requested job is completed. If a job needs to be completed last, it will be frozen until all other jobs have been completed.

The stacking function is used to insert related operations into the requested position when a user wants to receive the service in a specific order. For example, if a user wants job J_4 performed immediately after job J_1 , the operation manager stacks the operations of J_4 on top of J_1 so that J_4 can be executed immediately after J_1 is completed.

The combining function is used when a user wants to receive multiple services simultaneously. If a user wants jobs J_1 and J_4 to be executed together, such as by asking for a meal and coffee to be delivered together, the service can be performed separately by two different agents simultaneously or by a single agent. If two agents are running in parallel, the operation manager uses the freezing function and freezes the last operation of J_1 and J_4 until both are in the candidate queue, and the two agents are available for simultaneous delivery. The combining function is used in a single-agent execution, and a new operation is created that combines the last operations of J_1 and J_4 to be performed by the agent as one. When all the previous operations are completed, the new operation O_{14cb} is fetched and replaces the last two operations in the candidate queue. The combining function can only be applied when the two operations have the same operation type, object, start position, and end position. In addition, only agents that can perform both operations can execute a combined operation. The duration of the combined operation was arbitrarily set to be longer than that of both operations.

The scheduler also utilizes human monitoring and agent monitoring, which allows multiple agents to provide services at user-specified preferred times. For instance, the operation manager freezes the operation in the candidate queue until the human monitoring condition is met. The operation status in the candidate queue can be "ready", "frozen", and "execute". There is no "finished" status for operations because when an operation is complete, the agent status changes to "finished", and the operation is dequeued from the candidate queue by the operation manager. As described above, the operation manager is implemented according to the pseudocode reported in Algorithm 2.

Algorithm 2: OperationManager()

```

//Initialization
InQ //Array of Incoming Queues where InQ[n] is the n-th Operation in the Incoming Queue of
Job ji
CanQ //Array of Candidate Queues where CanQi[n] is the n-th Operation in the Candidate
Queue of Job ji
OA = {Oa1, Oa2, Oa3} //where Oak is the Operation Allocated to Agent ak
SA = {Sa1, Sa2, Sa3} //where Sak is the current state of Agent ak; 'Idle' or 'Execute' or 'Finished'
H ← HumanMonitoring(O)
if O is Updated then
    for i = 1 to M do
        Oi ← getOperations(O)
        CanQi ← Oi[0]
        Oi ← remove(Oi[0])
        InQi ← Oi
        i ← i + 1
    end
    (InQ, CanQ) ← RearrangeQueue(InQ, CanQ, H, R)
    break;
end
SA ← AgentMonitoring(A)
Afin ← getFinishedAgents(SA)
if Afin ≠ ∅ then
    for ak ∈ Afin do
        CanQi ← pop(Oak)
        O ← remove(Oak)
    end
    (InQ, CanQ) ← ManageQueue(InQ, CanQ)
end
if H is True then
    (InQ, CanQ) ← UnfreezeQueue(InQ, CanQ, H, R)
end

```

3.2.3. Operation Allocator

The operation allocator allocates operations according to the order specified by the operation manager. The operation allocator allocates operations from the candidate queue to available agents. In this study, the problem of allocating operations to each agent is addressed by solving the following optimization problem (1):

$$\min_x (\max_{a_k} \sum_{i=1}^M \sum_{j=1}^N (w_{ija_k} \cdot x_{ija_k})) \quad (1)$$

subject to

$$a_k \forall k \in \{1, \dots, K\}$$

$$\begin{aligned}
 &O_{ij} \quad \forall i \in \{1, \dots, M\}, \forall j \in \{1, \dots, N\} \\
 &x_{ij,a_k} [\text{boolean}] \quad \forall O_{ij}, \forall a_k \in A \\
 &W_{ija_k} = C_{ija_k} \cdot D_{ija_k} \quad \forall O_{ij}, \forall a_k \in A \\
 &\sum_{a_k=1}^A x_{ij,a_k} = 1 \quad \forall O_{ij}, \forall a_k \in A
 \end{aligned}$$

The term W_{ija_k} represents the weight of the allocation operation O_{ij} to agent a_k . C_{ija_k} and D_{ija_k} are the capability and duration, respectively, and the weight is calculated as their combination. If agent a_k was capable of performing O_{ij} , C_{ija_k} was given a constant 1, and if the agent was incapable, C_{ija_k} was given an infinite cost. Boolean variable $x_{ija_k} \in \{0,1\}$ is a decision variable for detecting whether operation O_{ij} is assigned to each agent. According to the last constraint, the sum of the decision variables for a single operation is always equal to 1, only one of the W_{ija_k} is true, and the agent is decided for the allocation. Because we adopted parallel execution of multiple agents, we modified the multi-objective mixed-integer linear program from [8] to a min-max optimization to minimize the longest completion time among the entire agents. We use decision variables to determine an appropriate agent for each operation to minimize the overall time. As shown in Function CostCalculator, this calculation process determines the number of cases of the decision variable by repeated permutation and compares the time required for each case to find the combination of agents with minimum cost. The result of the optimization problem (1) is an operation for idle agent a_k to perform. When the operation is allocated, the agent status changes to “executing” and when the agent is completed, it changes to “completed”, signaling the operation manager to dequeue the completed operation from the candidate queue. As previously described, the dynamic allocator is presented with its pseudocode in Algorithm 3.

It should be noted that our framework separates the sequencing and allocation components of the scheduler. The order of operations is already in place because the operation manager has utilized the user requirements in the operation queue and handled the relationship between the requested services. This means that the allocation is executed simply, as the operation allocator only needs to carefully time the execution of each operation to minimize the total cost.

Algorithm 3: DynamicAllocator()

```

 $S_A \leftarrow \text{AgentMonitoring}(A)$ 
 $A_{fin} \leftarrow \text{getFinishedAgents}(S_A)$ 
 $A_{idl} \leftarrow \text{getIdleAgents}(S_A)$ 
if  $A_{fin} \neq \emptyset \parallel (\text{CanQ is Updated} \ \& \ A_{idl} \neq \emptyset)$  then
  for  $a_k \in A_{fin}$  do
     $O_{ak} \leftarrow \emptyset$ 
     $S_{ak} \leftarrow \text{Idle}$ 
  end
  while  $O \neq \emptyset$  do
     $A_{idl} \leftarrow \text{getIdleAgents}(S_A)$ 
    if  $\text{CanQ} \neq \emptyset$  do
       $X_{real} \leftarrow \text{CostCalculator}(x, w)$ 
      for  $a_k \in A_{idl}$  do
         $O_{ak} \leftarrow \text{AllocateOperation}(X_{real})$ 
        if  $O_{ak} \neq \emptyset$  then
           $S_{ak} \leftarrow \text{Execute}$ 
          break
        else
           $S_{ak} \leftarrow \text{Idle}$ 
          break
        end
      end
    end
    else
       $S_{ak} \leftarrow \text{Idle}$ 
      break
    end
  end
end

```

```

Function CostCalculator( $x, w$ )
//Initialization
 $W_c = 0$  //Total Cost of a case
 $W_{min} = Inf$  //Minimum Cost amongst all cases
input:  $x_{i,j,ak}$  //Decision Variable [Boolean]
            $w_{i,j,ak} = C_{ij}[k] \cdot D_{ij}[k]$ 
 $N_A$  //Number of Agents
 $N_{cq}$  //Number of 'Ready' Operations in the Candidate Queue
 $X = \{x_{ij,a1}, x_{ij,a2}, x_{ij,a3}, \dots, x_{ij',ak}\}$  //A case of an Allocation
where  $\forall O_{ij} \in CanQ, \sum_{a_k=1}^A x_{ij,a_k} = 1$ 
 $X_p$  //Set of Possible cases of Allocation
 $N_{X_p}$  //Number of Set of Possible cases of Allocation
 $X_p[N_{X_p}] \leftarrow \mathbf{GetPermutation}(CanQ, A)$  where  $N_{X_p} = (N_A)^{N_{cq}}$ 
for  $n = 1$  to  $N_{X_p}$  do
     $X_n \leftarrow X_p[n]$ 
    for  $k = 1$  to  $N_A$  do
         $W_{a_k} = \sum_{i=1}^N \sum_{j=1}^M (w_{i,j,a_k} \cdot x_{i,j,a_k})$ 
        if  $W_c < W_{a_k}$  then
             $W_c = W_{a_k}$ 
        end
         $k \leftarrow k + 1;$ 
    end
    if  $W_{min} > W_c$  & Executable then
         $W_{min} = W_c$ 
         $X_{real} \leftarrow X_n$ 
    end
     $n \leftarrow n + 1;$ 
end
return( $X_{real}$ )

```

4. Experiments and Results

To evaluate the proposed scheduler, we built a virtual reality (VR) smart office for simulation (Figure 2) and used it to experiment with four service scenarios.

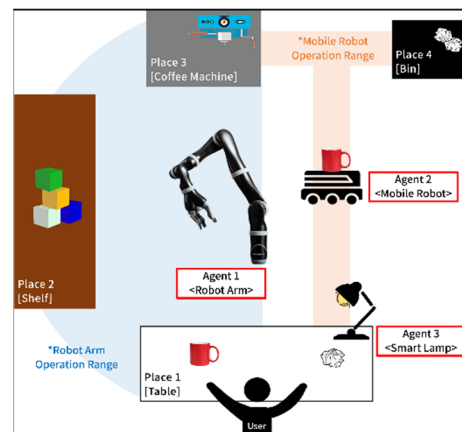


Figure 2. VR smart office setting.

4.1. Smart Office Prototype

A VR smart office, in which multiple robots provide daily services to a user, consists of a user sitting at a table and three agents with different functions. For the active agents, KINOVA Jaco [28], a robotic arm capable of object manipulation (e.g., pick-up, transfer, put down), and a simulated mobile robot were used. A smart lamp, an integration of an IoT system, was used as the passive agent. The operations used in this experiment are listed in Table 1.

4.2. System Setup

The simulated environment for providing services to a user in a VR smart office is illustrated in Figure 3a. The VR smart office was displayed on a 55-inch flat 3D display with full HD (1920 × 1080). An RGBD camera was mounted on the top to track the user's hand and show that the user received a service from the agent (Figure 3b).

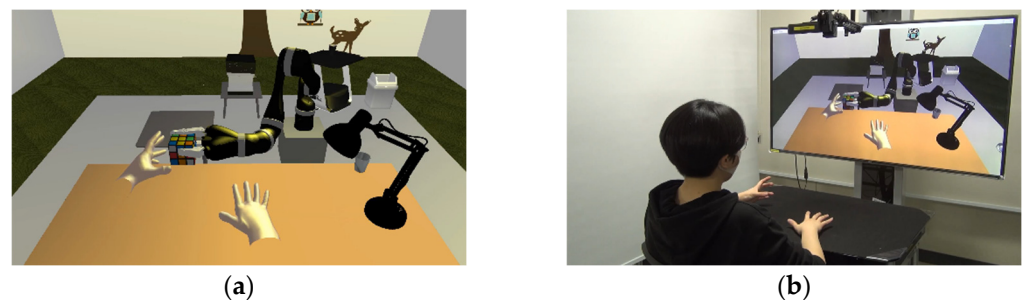


Figure 3. VR smart office: (a) robots provide service to user and (b) experimental setup.

The experimental system was implemented in Openframeworks from OpenGL to C++ and Bullet Physics 2.82 for the physics simulation. The robotic arm was planned and simulated on the ROS Moveit and the simulated mobile robot on Bullet Physics. The experiments were performed on an Intel Core i7-3770 3.5 GHz computer with 16 GB memory.

4.3. Experimental Scenarios

We designed four service scenarios to demonstrate the proposed scheduling method in dynamic situations of our daily life environment. The dynamic situation that occurs when a user requests and receives services is composed of the following scenarios. First, the user requests a new task while the agents are still performing the previous task (Scenario 1). The second scenario includes a delay situation. Agents interact with humans in real service situations, and although we can estimate the average execution time of the agents, it is very difficult to predict it accurately; therefore, delays cannot be avoided (Scenario 2). The

following are additional dynamic situations caused by a user. A user may request a service, including specific requirements regarding when it should be provided. These requirements may include “request to change the order of a service,” “simultaneous execution of services,” or “preferred time for the execution of service.” Therefore, we selected two different scenarios for the various user requests. The third scenario occurred when a user requested a change in the execution order of the service (Scenario 3) and the fourth when a user designated a preferred time for a specific service (Scenario 4).

4.4. Results and Discussion

The proposed scheduler was compared with a baseline scheduler, which represents a basic scheduling method that exemplifies how the existing scheduling solutions would work for service robots. Both schedulers determine the optimum schedule that minimizes the overall time span based on the agent capability and duration. The baseline scheduler executes jobs on “first-in-first-out” basis and reschedules them whenever a new job is requested.

4.4.1. Scenario 1: Additional Job Request

In Scenario 1, a user asks for an additional job “Bring water” while the agents are executing jobs J1 and J2. When a new request is made in the form of job J4, the baseline scheduler reschedules them because jobs J3 and J4 have not been yet executed; however, the proposed scheduler dynamically accepts and reschedules the new request, including operation O₂₂ of Job 2. In other words, the proposed scheduler divides the job into more basic units of operation, so that multiple agents share and complete the job more efficiently. In Figure 4, the block representing the execution of each job (operation) is shown based on the execution start and end times of the job by the scheduler. Figure 4 shows a 7.14% reduction in the total completion time when using the proposed scheduler compared with the baseline. The proposed method provides services in the order of the user’s requests whereas the baseline does not.

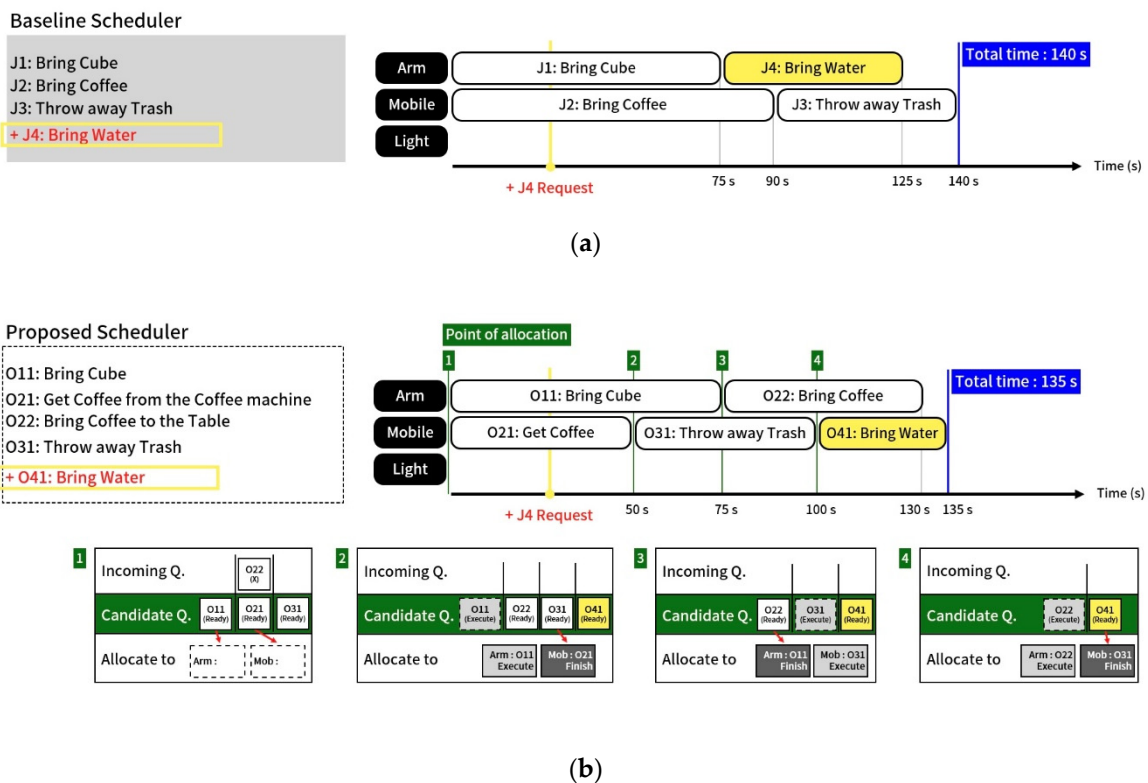


Figure 4. Additional job request: (a) the baseline scheduler and (b) the proposed scheduler.

4.4.2. Scenario 2: Delay Occurs

Scenario 2 is a delay situation that occurs when an agent executes a service. The baseline cannot handle systemic changes because it can only be rescheduled on a new request. If there is a delay, the baseline encounters problems in all subsequent jobs. As shown in Figure 5a, if the robot arm is delayed in bringing the cube, Job 3 will also be delayed, and the mobile robot will not be able to take over Job 4 even after it has finished performing the job. This delays the provision of the service, and the user waits for it for a long time.

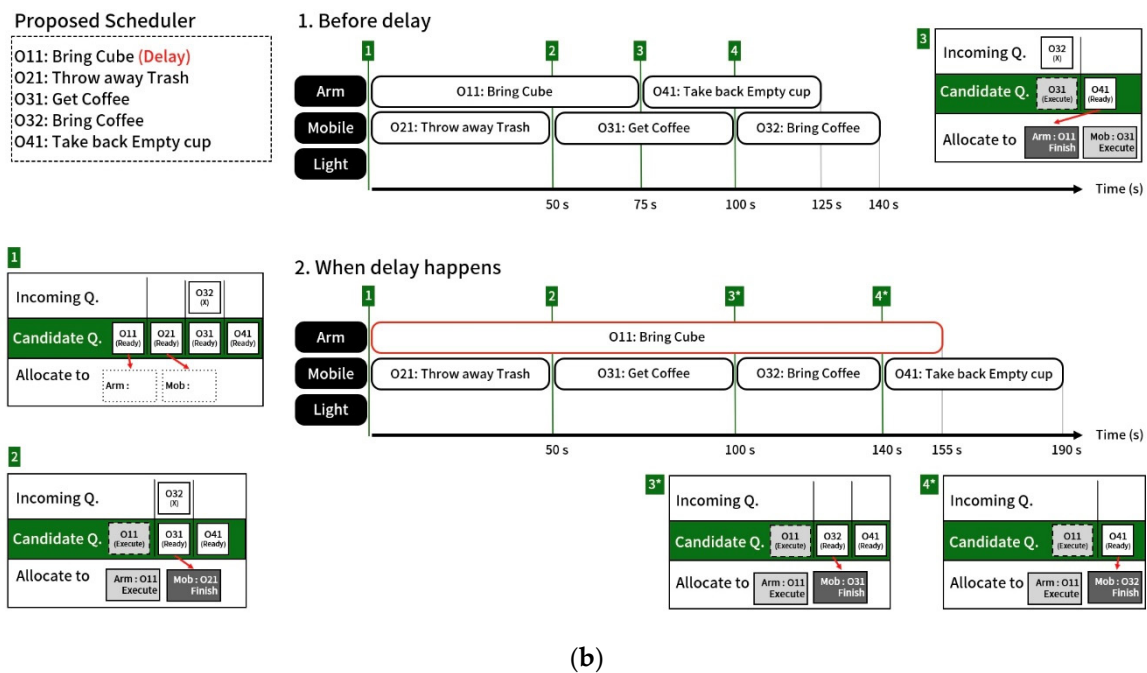
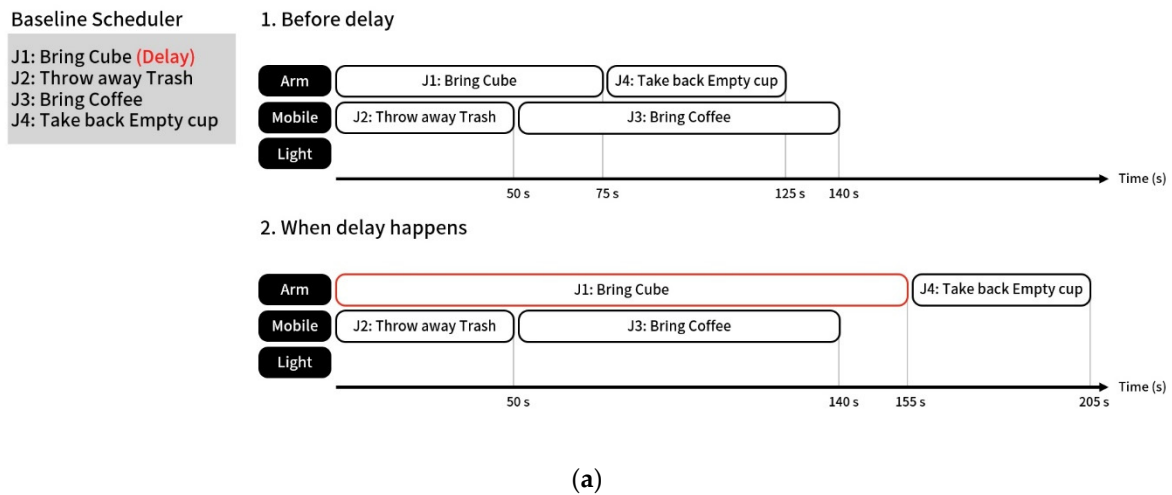


Figure 5. Delay occurs: (a) the baseline scheduler and (b) the proposed scheduler.

However, our scheduler can handle delays by adjusting the schedule based on these system changes and dynamically allocating the rest of the job to account for the overall outcome. The operation allocator allocates operations whenever a previous operation is completed, and the other available agents can take over the remaining operations, even if there is a delay. As shown in Figure 5b, the proposed method reduces the delay of the overall service by dynamically allocating Job 4 to a mobile robot when the robot arm is occupied by the delay of Job 1. In addition, it is noteworthy that the computational load of

the allocation is similar to that of the baseline. This is because instead of calculating the entire job each time, it selectively computes the optimal allocation only from the candidate queue. The computational cost of the allocation is considerably reduced because the number of operations in the candidate queue is always constant. Therefore, our scheduler can conveniently solve a systemic change, such as a delay, through dynamic allocation.

4.4.3. Scenario 3: User Requests to Change the Order of Service

In Scenario 3, the execution order of services is changed according to the user request. The user requested jobs in the following order: Job 1 (Bring a cube), Job 2 (Bring a cup of coffee), Job 3 (Take back an empty cup), and Job 4 (Bring water). Suppose the user wants to get coffee (Job 2) after removing the empty cup from the table (Job 3) because the table is full.

In the baseline scheduler, Job 3 is only assigned after all previous requests have been executed; therefore, it does not meet the user requirement. However, the operation manager in our scheduler handles sequential relationships between jobs so that it can accommodate and follow user requirements. As shown in Figure 6b, the operation manager stacks the operations of Job 2 on top of Job 3. Operations O₂₁ and O₂₂ are allocated only after O₃₁ is completed and the service is successfully provided as requested by the user. As a result, while the baseline scheduler fails to satisfy user requests by bringing coffee before an empty cup is removed from the table, the proposed method succeeds.

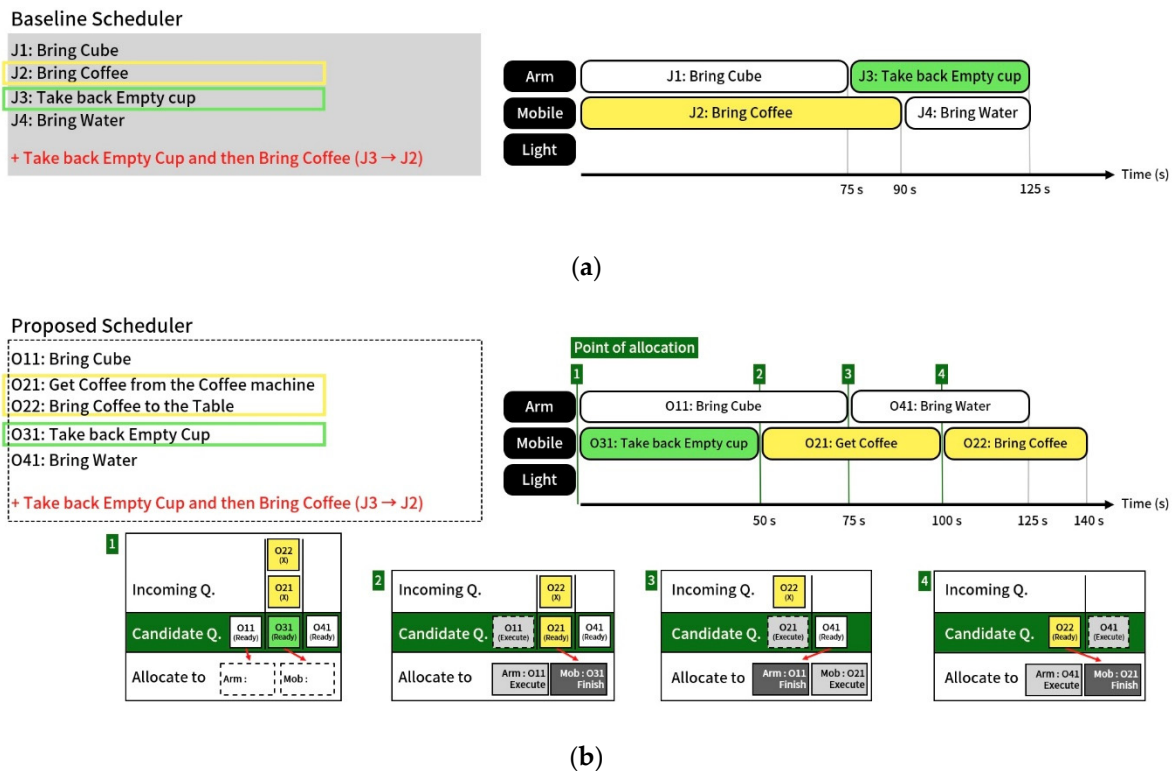


Figure 6. User requests to change the order of service: (a) the baseline scheduler and (b) the proposed scheduler.

4.4.4. Scenario 4: Preferred Time and Simultaneous Request

Scenario 4 includes a user requesting two jobs to be performed simultaneously or having a preferred execution time for the service. For example, a user may ask for two empty cups to be simultaneously brought from a table and a reading light to turn on when the user performs a specific action. To visually demonstrate the execution of the service in the VR smart office, the specific action is lifting a cup.

The baseline scheduler does not accept these requests; therefore the two cups are fetched separately, and the reading light is on regardless of the user’s status, as shown in Figure 7a. The proposed scheduler combines O_{31} and O_{51} using the combining function of the operation manager to change the new operation O_{31-51} to be performed by a single agent so that the mobile robot simultaneously takes back empty cups 1 and 2, as shown in Figure 7b. Additionally, as shown in Figure 8, O_{41} “Turn on light” is performed when the user raises the cup.

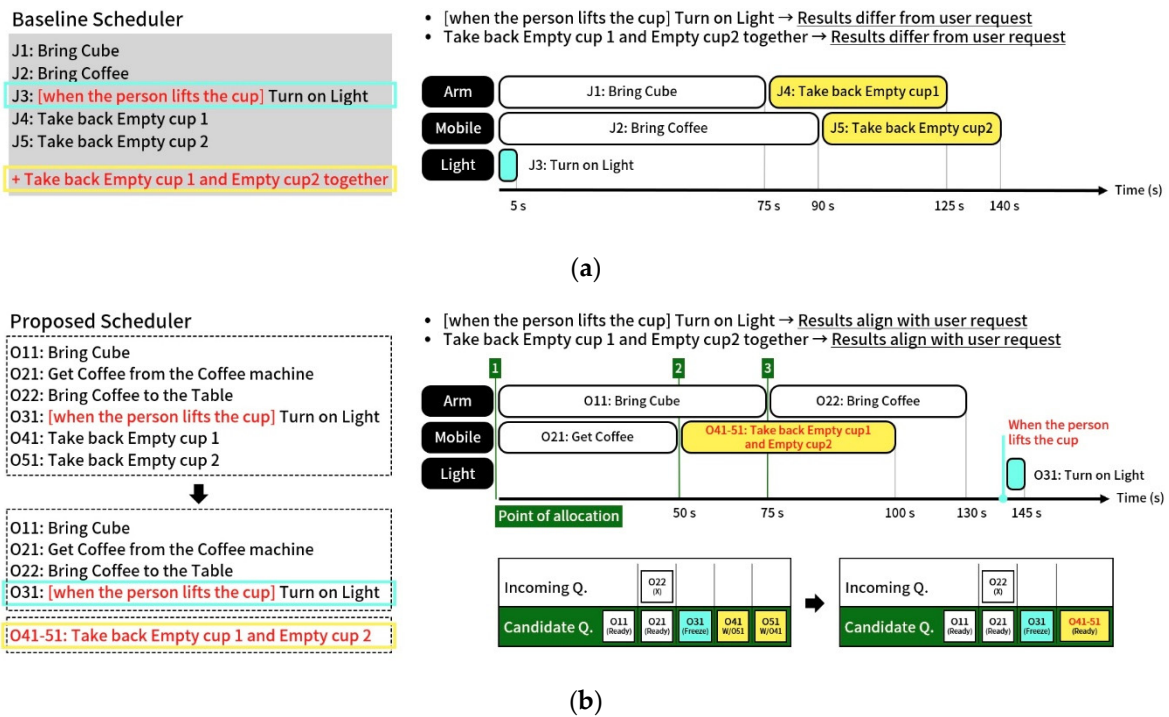


Figure 7. Preferred time and simultaneous request: (a) the baseline scheduler and (b) the proposed scheduler.

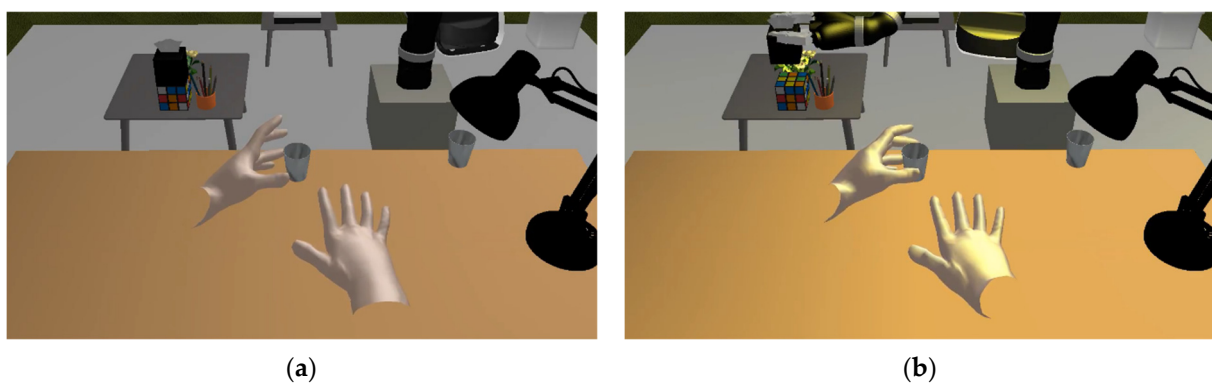


Figure 8. Execution result of the proposed scheduler: (a) before the user lifts the cup and (b) after the user raises the cup (light turns on).

5. Conclusions

We propose a novel scheduling framework for heterogeneous multiple agents that can actively accept dynamic user requests and provide services in daily life. In this study, we focused on the sequential and temporal aspects of services that occur owing to user requests. Therefore, the aim was to dynamically accept new requests and reschedule services so that multiple robots could satisfy these sequential interdependencies or complex prerequisites

of a requested job. To validate the technical reliability of the proposed scheduler, we developed a VR simulation and tested it in four smart office scenarios with interactive and dynamic requests. The results indicated that our scheduler was effective in all the scenarios by providing appropriate services according to dynamic user requests. In future research, we plan to refine our optimization solution for improved computational efficiency. We plan to increase the complexity of the user request of service, such as different robots performing a single service by operating simultaneously. For real-world applications, the service setting could be broadened from a home to a wider and more complex multi-purpose facility with multiple users. We also plan to further expand the human monitoring function with various other IoT technologies and use the proposed system in real-world applications.

Author Contributions: Conceptualization, J.-M.P.; methodology, Y.J. and H.K.; software, H.K. and K.-D.S.; validation, Y.J., H.K. and K.-D.S.; writing—original draft preparation, Y.J.; writing—review and editing, J.-M.P.; supervision, J.-M.P.; project administration, J.-M.P.; funding acquisition, J.-M.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Korea Institute of Science and Technology (KIST) Institutional Program under Project 2E31581.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Collins, G.R. Improving human–robot interactions in hospitality settings. *Int. Hosp. Rev.* **2020**, *34*, 61–79. [[CrossRef](#)]
2. Lai, C.J.; Tsai, C.P. Design of introducing service robot into catering services. In Proceedings of the 2018 International Conference on Service Robotics Technologies, Chengdu, China, 16–19 March 2018; pp. 62–66.
3. Wang, T.M.; Tao, Y.; Liu, H. Current researches and future development trend of intelligent robot: A review. *Int. J. Autom. Comput.* **2018**, *15*, 525–546. [[CrossRef](#)]
4. Lee, I. Service Robots: A Systematic Literature Review. *Electronics* **2021**, *10*, 2658. [[CrossRef](#)]
5. Chiang, A.H.; Trimi, S. Impacts of service robots on service quality. *Serv. Bus.* **2020**, *14*, 439–459. [[CrossRef](#)]
6. Ismail, Z.H.; Sariff, N.; Hurtado, E.G. A survey and analysis of cooperative multi-agent robot systems: Challenges and directions. In *Applications of Mobile Robots*; IntechOpen: London, UK, 2018; pp. 8–14.
7. Johannsmeier, L.; Haddadin, S. A hierarchical human-robot interaction-planning framework for task allocation in collaborative industrial assembly processes. *IEEE Robot. Autom. Lett.* **2016**, *2*, 41–48. [[CrossRef](#)]
8. Pupa, A.; Van Dijk, W.; Secchi, C. A human-centered dynamic scheduling architecture for collaborative application. *IEEE Robot. Autom. Lett.* **2021**, *6*, 4736–4743. [[CrossRef](#)]
9. Maderna, R.; Pozzi, M.; Zanchettin, A.M.; Rocco, P.; Prattichizzo, D. Flexible scheduling and tactile communication for human–robot collaboration. *Robot. Comput. Integr. Manuf.* **2022**, *73*, 102233. [[CrossRef](#)]
10. García, S.; Strüber, D.; Brugali, D.; Berger, T.; Pelliccione, P. Robotics software engineering: A perspective from the service robotics domain. In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, 8–13 November 2020; pp. 593–604.
11. Xu, K.; Fei, R.; He, D. A tabu-search algorithm for scheduling jobs with precedence constraints on parallel machines. In Proceedings of the 2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA), Wuhan, China, 31 May–2 June 2018; pp. 2774–2781.
12. Casalino, A.; Zanchettin, A.M.; Piroddi, L.; Rocco, P. Optimal scheduling of human–robot collaborative assembly operations with time petri nets. *IEEE Trans. Autom. Sci. Eng.* **2019**, *18*, 70–84. [[CrossRef](#)]
13. el Makrini, I.; Merckaert, K.; de Winter, J.; Lefeber, D.; Vanderborght, B. Task allocation for improved ergonomics in Human-Robot Collaborative Assembly. *Interact. Stud.* **2019**, *20*, 102–133. [[CrossRef](#)]
14. Malik, A.A.; Bilberg, A. Complexity-based task allocation in human-robot collaborative assembly. *Industrial Robot* **2019**, *46*, 471–480. [[CrossRef](#)]
15. Riedelbauch, D. Dynamic Task Sharing for Flexible Human-Robot Teaming under Partial Workspace Observability. Ph.D. Thesis, University of Bayreuth, Bayreuth, Germany, 2020.
16. Ranz, F.; Hummel, V.; Sihm, W. Capability-based task allocation in human-robot collaboration. *Procedia Manuf.* **2017**, *9*, 182–189. [[CrossRef](#)]

17. Dalle Mura, M.; Dini, G. Designing assembly lines with humans and collaborative robots: A genetic approach. *CIRP Ann.* **2019**, *68*, 1–4. [[CrossRef](#)]
18. Pearce, M.; Mutlu, B.; Shah, J.; Radwin, R. Optimizing makespan and ergonomics in integrating collaborative robots into manufacturing processes. *IEEE Trans. Autom. Sci. Eng.* **2018**, *15*, 1772–1784. [[CrossRef](#)]
19. Levine, S.J.; Williams, B.C. Watching and Acting Together: Concurrent Plan Recognition and Adaptation for Human-Robot Teams. *J. Artif. Intell. Res.* **2018**, *63*, 281–359. [[CrossRef](#)]
20. Darvish, K.; Bruno, B.; Simetti, E.; Mastrogiovanni, F.; Casalino, G. Interleaved online task planning, simulation, task allocation and motion control for flexible human-robot cooperation. In Proceedings of the 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), Nanjing, China, 27–31 August 2018; pp. 58–65.
21. Bänziger, T.; Kunz, A.; Wegener, K. Optimizing human-robot task allocation using a simulation tool based on standardized work descriptions. *J. Intell. Manuf.* **2020**, *31*, 1635–1648. [[CrossRef](#)]
22. Raatz, A.; Blankemeyer, S.; Recker, T.; Pischke, D.; Nyhuis, P. Task scheduling method for HRC workplaces based on capabilities and execution time assumptions for robots. *CIRP Ann.* **2020**, *69*, 13–16. [[CrossRef](#)]
23. Zhu, J.; Shi, J.; Yang, Z.; Li, B. A real-time decentralized algorithm for task scheduling in multi-agent system with continuous damage. *Appl. Soft Comput.* **2019**, *83*, 105628. [[CrossRef](#)]
24. Liu, S.; Tian, G.; Zhang, Y.; Zhang, M.; Liu, S. Service planning oriented efficient object search: A knowledge-based framework for home service robot. *Expert Syst. Appl.* **2022**, *187*, 115853. [[CrossRef](#)]
25. Zhang, Y.; Tian, G.; Zhang, S.; Li, C. A knowledge-based approach for multiagent collaboration in smart home: From activity recognition to guidance service. *IEEE Trans. Instrum. Meas.* **2019**, *69*, 317–329. [[CrossRef](#)]
26. Zhang, Y.; Tian, G.; Chen, H. Exploring the cognitive process for service task in smart home: A robot service mechanism. *Future Gener. Comput. Syst.* **2020**, *102*, 588–602. [[CrossRef](#)]
27. Wang, Z.; Tian, G. Hybrid offline and online task planning for service robot using object-level semantic map and probabilistic inference. *Inf. Sci.* **2022**, *593*, 78–98. [[CrossRef](#)]
28. Kinova Jaco Assistive Robotic Arm. Available online: <https://assistive.kinovarobotics.com/product/jaco-robotic-arm> (accessed on 18 September 2022).