Three-Layered Architecture for Teleoperators and Its Module Arrangement

K. Hoshino and Y. Kunii
Chuo University, Tokyo, Japan
{kosshi_11, kunii}@hmsl.elect.chuo-u.ac.jp

Abstract—In this paper, we propose an intelligent system architecture for teleoperators (e.g., planetary exploration rovers). Our proposed architecture offers advanced flexibility (variability), efficiency, scalability, and transparency. The architecture is composed of one hardware layer (bottom) and two software layers (middle and top). Because the software is divided into two layers, users are not required to actually connect the software modules. With this architecture, we aim to achieve both efficient task construction by the users and easy management of modules by the system. Moreover, to achieve high-speed data communication between software modules we use a shared memory. Using our proposed architecture, we can efficiently perform repairs and consequently enhance the functionality of teleoperator systems. Therefore, our proposed architecture can provide significant contributions to the development and operation of teleoperators.

I. INTRODUCTION

In this paper, we propose a system architecture for teleoperators (i.e., remote mobile robots). In general, teleoperators are required to achieve stable performance while performing advanced missions in various environments. Several system architectures for robots have been proposed that achieve this capability [1]-[6]. Teleoperators are composed of several functions (e.g., action planning, recognition, and motion control) and various subsystems (e.g., moving mechanisms, communication system, and various sensors such as cameras). Because teleoperator systems are multifunctional, they tend to be bulky and complex. Conversely, the systems are required to be scalable and efficiently adapt to any situation. To overcome this problem, these systems should enable users to freely combine various elements of the robot via a network, thus allowing us to easily modify parts of the system. Thereby, these systems can efficiently perform advanced tasks.

A problem with most existing software architectures of teleoperator systems is that it is difficult to operate a robot when system failures occur in remote locations [7]-[9]. This is because it is difficult to dynamically modify functions of the robot. From this viewpoint, it is highly desirable to develop an architecture that has advanced scalability and variability. Moreover, teleoperator systems should address the problem of info-communication, i.e., to pass sensory information from a remote site to human operators [10][11]. In particular, to operate the robot safely, it is important to know the state of the system. However, this is difficult because such systems are complex.

For these reasons, we designed a system architecture that emphasizes on the flexibility (variability) of the structure of functions and transparency of data. In this study, we introduce, implement, and evaluate an intelligent system architecture for teleoperators. Moreover, we show that our architecture provides flexibility (variability), scalability, and transparency. We realize advanced variability by defining real and virtual connections in different layers. Moreover, because the operator is not required to actually connect software modules, our architecture also provides usability. Our implementation is based on RT-Middleware [12]-[14]. Finally, we evaluate our architecture by comparing its performance against a system composed by genuine RT-Middleware components.

II. PROPOSED SOFTWARE ARCHITECTURE FOR TELEOPERATED SYSTEMS

In general, teleoperators operate at locations where humans cannot easily perform activities. Moreover, the environment in these locations is not necessarily well known. Therefore, system failures may occur because of nonconformity of parameters or algorithms. In addition, harsh environment conditions often cause hardware problems. In this case, because teleoperators operate in locations where humans cannot easily perform activities, the system should be altered without any physical restrictions using only software. For these reasons, the system must be flexible and should be able to change the structure of its functions accordingly. Moreover, to operate the robot safely, it is important to know the state of the system during its operation. Therefore, we designed a system that emphasizes on the flexibility (variability) of the structure of functions and transparency (accessibility) of data.

In our proposed architecture (shown in Fig. 1.), each function is modularized and connected via a network. Furthermore, advanced variability is achieved by defining real and virtual connections in different layers. In the next subsections, we describe in detail each layer of our architecture.

A. Physical layer

In this bottom layer, all hardware is connected via a network as shown in Fig. 2. Thus, it is possible to directly access any function, and connections can be changed without any physical restrictions using only software. Thus, our system is accessible and has an advanced variable structure. In addition, it increases fault tolerance by minimizing the units that are lost when system failures occur.
B. Connection layer

For a robot to operate in remote locations, it must be capable of switching multiple tasks (composed of a module’s behavior logic) in a flexible manner depending on the situation. However, due to the cost associated with its connecting tasks, it is not realistic to actually switch connections. For this reason, this middle layer manages the actual modules of the system and virtually realizes task dependencies defined in the top layer. This action is performed by the database node module (DNM), which relays information between the functions of the modules. In particular, all modules are connected to DNM, as shown in Fig. 3, and data is exchanged between them at high-speeds via shared memory. DNM realizes a network list by transmitting the destination addresses for each module that contains task dependencies defined by the user in the logical layer. Hence module connections can be switched by changing reference pointers. DNM manages the timing of the switches. Moreover, because DNM contains the data of all modules, system transparency is improved.

To achieve load balancing and reduce traffic, DNM can be arranged in a hierarchical structure (see Fig. 4). In particular, the Newman algorithm [15][16] can be used to cluster modules, which can then be placed at each layer of the hierarchical DNM structure. Moreover, because the hierarchical structure limits the range of failures, using this structure we can more easily identify the causes of failures.

C. Logical layer

In this top layer, we introduce a method that enables users to intuitively compose tasks, thereby improving the efficiency of composing tasks. Users collect the necessary modules and connect them according to the intended task flow, as shown in Fig. 5. Our method allows free swapping, adding, replacing, and deleting modules. Thus, the system can effectively rebuild its functions and respond quickly when situations change or problems arise.

III. REALIZATION OF VARIABILITY FOR TASK FLOW

To realize the proposed architecture, the system is required to dynamically switch tasks and share data between multiple DNM’s. In this section, we describe in detail these functionalities.

A. Dynamically switching tasks

To switch tasks, we must dynamically change the connections between modules that comprise a task. Using DMM to manage the execution of all modules is not preferred because of increase in overhead. Thus, DNM only provides information about the connections, and each module manages its own execution. In particular,:

1. Each module is connected to the DNM and assigned a dedicated shared memory space. The capacity of the memory is provided on the basis of the number of each output port.
2. Using the netlist (i.e., task flow) information of each module, DNM sets references between input and output ports of the appropriate module. Input ports continuously monitor the connections to the output ports.

3. When the execution of a module is complete, the value of each output port is updated. When an input port receives the updated information, it reads the value of the output port.

B. Data sharing between multiple DNMs

In our architecture, we use shared memory to achieve high-speed data communication between modules. However, when modules are present between different DNMs, communication between modules cannot be achieved. This is because the shared memory space of each DNM is independent. To overcome this problem, we use the netlist to send and receive data between DNMs. Therefore, each DNM is equipped with a function that automatically finds a route to a destination using the information of the netlist, and synchronizes the data of all shared memory spaces. Moreover, because this architecture implementation is based on RT-Middleware (CORBA), DNMs communicate via CORBA.

IV. OPTIMIZED PLACEMENT OF MODULES

A. Optimized placement method

In hierarchical DNMs, we achieve load distribution by optimizing the placement of modules on each DNM. Therefore, we cluster the network of modules on the basis of the task dependencies generated in the logic layer. Then, groups of modules that are classified as a cluster are mapped automatically (Fig. 6). Thus, compared with the conventional manual placement of modules, we improve operational efficiency because our modules are highly cohesive with low coupling.

Using the Newman method, nodes are clustered according to the descending order of coupling, which depends on the network configuration. However, in our architecture, we also need to consider the communication between modules. Therefore, our clustering method uses the improved Newman algorithm with weighting. Weights are determined on the basis of data capacity, communication times between modules, and strength of the connection. In particular, the weights \( w_{nn} \) are computed using (1), where \( K_{char} \) is a constant that depends on network characteristics, \( D_n \) is the amount of data exchanged between modules, and \( E_m \) is the number of communication exchanges between modules. Furthermore, the normalized weight \( w_{nn} \) is obtained by (2), where \( l \) is the total number of connections between modules.

\[
w_{nn} = K_{char} \cdot D_n \cdot E_m
\]

\[
w_{nn} = \left( K_{char} \cdot D_n \cdot E_m \right) \left( \frac{l}{\sum_{w} w_{nn}} \right)
\]

B. Measure of optimality

We evaluated the weighted Newman algorithm against the conventional one [16] by comparing their modularity quality (MQ), which indicates the goodness of a cluster. The results of the comparison are shown in Fig. 7. The maximum MQ value obtained by the weighted Newman method (0.53915) was larger than that of the conventional method (0.35893). Therefore, we conclude that our algorithm achieves a more efficient load distribution.

Figure 6. Clustering and mapping processes

Figure 7. Evaluation with MQ
V. SIMULATION RESULTS AND EVALUATION

A. Data communication time

We evaluated the performance of our system by comparing the data communication time of our virtual inter-module connections with that of conventional RT-Middleware. The results of this comparison are shown in Fig. 8. The results indicate that the communication time of the virtual connections in our architecture is considerably lower than that of conventional RT-Middleware. Therefore, we conclude that data communication in this system is efficient.

B. Variability

To compare variability, we measured the speed of switching tasks of actual connections using RT-Middleware and that of the virtual connections used in our proposed architecture. The results of the comparison are shown in Fig. 9. The results show that the speed of switching in our architecture is faster than that using RT-Middleware. Therefore, we confirm that our architecture offers efficient operation and task execution.

C. Load distribution and traffic reduction

To compare load distribution and traffic reduction, for each placement method shown Fig. 10, we measured the number of communications between DNMs and the CPU usage of DNM processes. The results of the comparison are shown in Table I. The results show that the weighted Newman method is more efficient than the regular Newman method (without weights).

VI. IMPLEMENTATION

We implemented our architecture on Beetle-One (Fig. 11), an experimental mobile robot aimed for planetary exploration, which is similar to the Micro6 rover (Fig. 12). The drive system of Beetle-One is based on the system used in electric wheelchairs. Its wheels are controlled via differential steering, and can be controlled by both joystick and computer controls. We implemented the monitoring and driving systems of Beetle-One using our proposed architecture.

In addition, we constructed a GUI (shown in Fig. 13), through which we could generate the tasks required by the logic layer and obtain the connection status in the connection layer. In particular, we provided a task generation interface through which users could intuitively generate a task. In addition, we provided a module connection interface, which is a visualization tool allowing users to monitor the connection status in the connection layer.

The results obtained using this implementation are shown in Fig. 14. Hence, we confirmed that it is possible to intuitively generate tasks and efficiently operate a robot.

<p>| TABLE I. CPU USAGE AND COMMUNICATION COUNT BY DNMs |
|---------------------------------|-------------|----------------|-----------------|</p>
<table>
<thead>
<tr>
<th>DNM(s)</th>
<th>CPU usage [%]</th>
<th>Number of communications between DNMs per minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>all DNMs</td>
<td>Per DNM</td>
<td>per minute</td>
</tr>
<tr>
<td>Single DNM</td>
<td>1</td>
<td>40.0</td>
</tr>
<tr>
<td>Newman method</td>
<td>4</td>
<td>89.5</td>
</tr>
<tr>
<td>Weighted Newman method</td>
<td>6</td>
<td>70.6</td>
</tr>
</tbody>
</table>
In this paper, we proposed a system architecture for teleoperators that offers advanced flexibility (variability), efficiency, scalability, and transparency. We realized advanced variability by defining real and virtual connections in different layers. Software modules are managed by the DNM. The system transparency is improved because the DNM contains the data of all modules. The architecture characteristics are validated by simulation results. Thus, our proposed architecture provides significant contributions to the development and operation of teleoperators. In future work, we plan to further improve the efficiency of our proposed architecture by incorporating a task scheduler in the logic layer.

REFERENCES


