Roboid Studio: A Design Framework for Thin-Client Network Robots

Kyoung Jin Kim  
Robomation, Co., Ltd.  
Seoul, Republic of Korea  
URL: http://www.robomation.kr  
Email: jin_khim@hanmail.net

Il Hong Suh  
College of Information and Communications  
Hanyang University  
Seoul, Republic of Korea  
Email: ihsuh@hanyang.ac.kr

Kwang-Hyun Park  
Department of Information and Control Engineering  
Kwangwoon University  
Seoul, Republic of Korea  
Email: akaii@kw.ac.kr

Abstract—The Roboid Studio is an all-in-one framework based on the Eclipse to develop robotics applications for thin-client network robots. The framework includes support programs, graphical interfaces, code libraries, a modeling language, a script language, execution semantics and a communication protocol. The Roboid Studio enables developers and even the people unskilled in the robotics to create software modules and a wide-variety of robot contents very easily. We first construct a robot model which defines the configuration of hardware and software devices and describes their functions and control methods. Unlike the existing model-based methods, the robot model in our approach is utilized not only to generate the APIs (Application Programming Interfaces) for access to software modules but also to create robot contents and communication packets. The Device Map Protocol (DMP) we propose in this framework is insensitive to transmission delay and packet loss, and ensures the robot can operate with all its parts and play multimedia data in sync.

I. INTRODUCTION

A personal service robot is a robot purchased by individual consumers which entertains, educates, assists and protects us in our living environment, while an industrial robot is exclusively employed in manufacturing and factory automation and a public service robot serves in public sectors[1]. The market size of the personal service robots is believed to be grown dramatically in many literatures even though it is yet immature or restricted to several types such as cleaning robots and robotic toys[2], [3]. The effect of the personal service robots on their users greatly depends on the list of available services and the functionality of the devices that build the robots. Thus, to seize the commercial opportunity and have a social impact, the diversity of services as well as the low price of products is the key to success in the market.

It is instructive to note that the user’s repetitive exposure to the same service contents can lead to the decrease of interest and results in a demand on a new service and benefit of ever-progressing scientific products. Early attempts to comply with this request for a standalone robot have tried to change functional modules and correct procedures to perform a new service content. These activities typically involve laborious, time-consuming and tedious efforts even though they can be carried out by specially trained experts. A network robot can be an alternative to overcome this drawback by continuously supplying new services through a network. In particular, a thin-client robot, a type of network robots, is attractive and competitive in terms of cost, accessibility and convenience. It can impose a computational burden on a remote server with high computing power for intelligent functions such as speech synthesis, face recognition and collective intelligence. It can also distribute and share sensory devices via a ubiquitous sensor network, and service contents can be developed and evolved in an independent machine replacing nothing inside a client robot.

In spite of these advantages of network robots, our attempt to create a competitive and useful application is faced with actual problems or imperative issues:

1) Opposed to the conventional standalone robots, one of the primary weakness of the network robots is to encounter the difficulty in control due to transmission delay, limitation of bandwidth and packet loss[3], [4], [5], [6], [7].
2) The consumers of the Web services have been familiar with multimedia contents showing the use of multimedia has increased by more than 100% each year[8]. This requests multimedia data as well as the motion data of a robot are incorporated into a robot content to conform to the user’s need.
3) The motion of each part often synchronizes with another in a robot. To cope with this requirement, the communication protocol has to ensure multiple sensor and effector data can be delivered at the same time.
4) A common software framework is critical to share the robot contents and reduce the development cost by avoiding repetitive work from scratch and running different robots on the same framework.

Given these problems and issues, we introduce a new framework for thin-client network robots as a solution. The framework includes a communication protocol and several tools for representing a robot model and composing robot contents.

This paper is organized as follows. In Section II, we introduce our framework and present its overall configuration to achieve the goal. In Section III, a communication protocol
II. FRAMEWORK FOR THIN-CLIENT NETWORK ROBOTS

A roboid is a compound word of “robot” and “-oid” which means “robot-like (device)” semantically. From an engineering point of view, a roboid is a particular form of thin-client network robots, composed of a minimal set of devices with very little computing power to cut down the cost. Therefore, in this form of robots, a communication protocol is crucial so as to reduce computational complexity.

The Roboid Studio is a framework based on the Eclipse to develop robotics applications for thin-client network robots. It includes support programs, graphical interfaces, code libraries, a modeling language, a script language, execution semantics and a communication protocol. The configuration of the system and the interface, as shown in Fig. 1, have been developed through HCI work involving design principles and iterative design practices.

A. Robot Model

A robot model is of particular importance early in the development of thin-client network robots. It is constructed by adding devices and control channels of the robot to the list of a robot profile (Fig. 1b). Unlike the existing model-based approaches, the robot model in our framework is utilized not only to generate the APIs (Application Programming Interfaces) for access to software modules but also to create robot contents and communication packets. We note that the devices and their order in the list define the packet format in a communication protocol, while the control channel describes control methods for corresponding devices and is applied to robot contents. In the robot model, we can select the type of each device in Table I, and specify relevant devices for each control channel in Table II. For example, we can connect the Lip Sync Channel “Lip Sync” to both the Effector “Speaker” and the Effector “Lip” so as to send audio data to the speaker device and control data to the lip device of a robot at the same time.

B. Timeline Editor

The Timeline Editor (Fig. 1c) is a graphical tool to create static contents for thin-client network robots, which aims to synchronize multimedia data and the motion of all devices in a robot. It shows exactly when the motion or multimedia data occur in the control channels. With the Timeline Editor we can sequence the motions of a robot on different tracks and adjust the timing of each motion easily and accurately by simple click and drag operation of a mouse and the change of values in a properties view (Fig. 1e).

In Fig. 1c, the horizontal axis is for the frame number spaced by 20msec and the vertical axis is for the tracks to
TABLE I

DEVICE TYPES IN ROBOT MODEL

<table>
<thead>
<tr>
<th>Device Type</th>
<th>Description</th>
<th>Payload Size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor</td>
<td>Input device (microphone, encoder, etc.); a robot sends data once to a server.</td>
<td>From a server to a robot: 0; from a robot to a server: the size of data defined in a robot model by the user</td>
</tr>
<tr>
<td>Effector</td>
<td>Output device (speaker, motor, LCD, etc.); a server sends data once to a robot.</td>
<td>From a server to a robot: the size of data defined in a robot model by the user; from a robot to a server: 0</td>
</tr>
<tr>
<td>Command</td>
<td>It ensures data are delivered to a robot from a server; a server resends data until it receives ACK from a robot.</td>
<td>Data from a server to a robot: one byte for ID + the size of data defined in a robot model by the user; ACK from a server to a robot: one byte for ID; the ending of a command and its ACK: 0</td>
</tr>
<tr>
<td>Event</td>
<td>It ensures data are delivered to a server from a robot; a server resends data until it receives ACK from a server.</td>
<td>Data from a robot to a server: one byte for ID + the size of data defined in a robot model by the user; ACK from a server to a robot: one byte for ID; the ending of an event and its ACK: 0</td>
</tr>
<tr>
<td>Memory</td>
<td>It ensures a large amount of data are delivered to a robot from a server; a server sends data once and waits for receiving ACK from a robot; if the delivery is failed, the server resends the data.</td>
<td>Data from a server to a robot: one byte for ID + two bytes for the data length (in bytes) + the size of data defined in the data length; request of ACK from a server to a robot: one byte for ID; ACK from a robot to a server: one byte for ID + two bytes for the data length; the ending of a memory command and its ACK: 0</td>
</tr>
<tr>
<td>Memory Event</td>
<td>It ensures a large amount of data are delivered to a server from a robot; a robot sends data once and waits for receiving ACK from a server; if the delivery is failed, the robot resends the data.</td>
<td>Data from a robot to a server: one byte for ID + two bytes for the data length (in bytes) + the size of data defined in the data length; request of ACK from a robot to a server: one byte for ID; ACK from a server to a robot: one byte for ID + two bytes for the data length; the ending of a memory event and its ACK: 0</td>
</tr>
<tr>
<td>Monitor</td>
<td>An effector, command and memory command can have a monitor to watch the status of the device.</td>
<td>The same size as the corresponding device</td>
</tr>
<tr>
<td>Module</td>
<td>A set of devices; if the corresponding bit in a Device Map is 0, all the child devices are invalid.</td>
<td>0</td>
</tr>
</tbody>
</table>

which each control channel of the robot model is automatically assigned. Small rectangles on each track represent either the key frames of motion data for the Linear Channel or the start frame of audio and motion data for the other types of control channels. The audio data are segmented into 960 bytes of data corresponding to 20 msec and the motion data between adjacent key frames are obtained by linear interpolation, so that a set of audio and motion data can be sent to a robot at every 20 msec.

C. Contents Composer

Opposed to the Timeline Editor, the Contents Composer (Fig. 1d) enables the robot to dynamically respond to the user’s behavior such as touch and mouse events by conditional branches, counters, etc. It enriches the robot contents by arranging motion clips created by the Timeline Editor and embedding other motion contents composed by the Contents Composer in sequence or concurrence.

Motion clips and motion contents can be connected with various logical, control, functional or conditional elements shown in the left vertical bar of Fig. 1d and in Table III to control the process flow in a content. When we play the content, the flow is initiated from the start element shaped like a movie film, and an arrow coming from one element and ending at another element represents that the process passes to the element the arrow points to. For example, in Fig. 1d, a motion clip “intro” is first executed, and then a motion content “clock_auto” follows after it. While this motion content is running, if we place the mouse cursor at (0,0) position of a screen and press a left mouse button, the motion content is stopped and the process passes to the next object since all the conditions in three consecutive triggers are met. Then, a motion content “talking_time” and a motion clip “talk_action” are executed together. When either is finished, the other is also stopped and the flow goes back to the motion content “clock_auto” again. To create graphical interfaces and to augment motion contents, we can also use JavaScript codes in the Contents Composer as shown in Fig. 2. For example, with JavaScript we can open a web browser or a flash movie and link the motion content with them to control web pages and flash movies and to respond to the events from them.
III. DEVICE MAP PROTOCOL (DMP)

A communication protocol is essential in the framework to exchange information between a server and a client robot. In traditional teleoperation and remote control, there have been numerous studies to overcome the problems due to transmission delay, limitation of bandwidth and packet loss[4], [5], [6], [7]. However, the methods are ineffective to thin-client network robots for lack of computing power, and some researches improve only the performance of a transport protocol having no consideration for the characteristics of a robot[4], [9], [10].

The Transmission Control Protocol (TCP) is the most popular and reliable protocol in the Internet with narrow bandwidth and high error rate, and a retransmission mechanism is used to avoid data loss[4]. However, the reliability is obtained at the cost of transmission delay. The User Datagram Protocol (UDP) has some advantages in terms of efficiency and bandwidth sensitivity[10], and is thus more efficient to transmit multimedia data in Internet-based applications.

The Device Map Protocol (DMP) we propose is basically running over the UDP, since timely delivery of data can be more important rather than accurate delivery to control a network robot. For example, sensor data for collision avoidance should be delivered in time. To handle the cases packet loss causes a serious problem, the DMP also supports a selective retransmission mechanism using acknowledgement (ACK)[3]. A server resends packets or the ending packet until it receives an ACK packet from a robot as shown in Fig. 3.

### TABLE III

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait Clip</td>
<td>Waits for given seconds</td>
</tr>
<tr>
<td>Join</td>
<td>Combines the paths of process flow</td>
</tr>
<tr>
<td>Fork</td>
<td>Starts the following motion clips and motion contents at the same time</td>
</tr>
<tr>
<td>Merge</td>
<td>Stops all the preceding motion clips and motion contents together if either is finished</td>
</tr>
<tr>
<td>Stop</td>
<td>Finishes a motion content</td>
</tr>
<tr>
<td>Counter</td>
<td>Counts the number of times the process flow passes</td>
</tr>
<tr>
<td>Dice</td>
<td>Generates a random number</td>
</tr>
<tr>
<td>Trigger</td>
<td>If a given condition is met, stops a connected motion clip or motion content and moves to the next element</td>
</tr>
<tr>
<td>Branch</td>
<td>Branches the process flow according to a given condition</td>
</tr>
</tbody>
</table>

![Fig. 3. Retransmission Mechanism in the DMP](image)

A. Packet Format of the DMP

The packet format of the DMP is very simple as shown in Fig. 4, and is composed of three parts: DMP header, Device Map and Payload. The DMP header represents the meta-data of the protocol and consists of nine elements as shown in Fig. 4 and Table IV. The size of the Device Map and the order of bits are determined by the list of devices in the robot model (Fig. 1b). For example, when the user includes sixteen devices into the model, the length of the Device Map becomes 16 bits.

The fixed-length packet format is less effective and sometimes can lead to problems in controlling thin-client network robots due to the small size of communication packets and short transmission rate (20msec). The DMP adopts the variable-length packet format so as to minimize overheads. When the device bit in Device Map is 0, the corresponding
payload data are omitted in runtime so that only the minimal data can be delivered through a network. For example, in Fig. 5, the first bit of the Device Map represents the Effector “RightWing” of a robot. The value “1” indicates that the corresponding device must be activated and its payload contains data. When the bit is 0, the corresponding payload can be omitted. For example, in Fig. 5 b, the third bit is 0 and the data of the Effector “Lip” can be omitted in the payload.

B. Correction of Delayed Packets

According to [6], the load of internet nodes is critical to transmission delay and the delay time is unpredictable since we cannot estimate the load of a node. If the transmission delay becomes larger, multiple packets can be received almost simultaneously as shown in Fig. 6. In this case, the robot needs to correct the cumulative data[6], while extensively delayed data is discarded and interpolated using the previous data. In this way, a client robot can maintain its posture until a new data is received.

IV. CONCLUSION

In this paper, we proposed a new framework and a communication protocol to develop robotics applications for thin-client network robots. In the framework, we can create two types of robot contents, a motion clip and a motion content,
by using the *Timeline Editor* and the *Contents Composer* respectively. The DMP protocol is insensitive to transmission delay and packet loss thanks to the variable-length packet format and a selective retransmission mechanism. In addition, the DMP ensures the robot can operate with all its parts and play multimedia data in sync. The proposed framework and communication protocol can be applied not only to thin-client network robots but also to rich-client network robots reducing the cost of robots.

**REFERENCES**


