Software Platform for Symbiotic Operations of Human and Networked Robots

人間とネットワークロボットの共生のためのソフトウェアプラットフォーム

Vuthichai AMPORNARAMVETH
National Institute of Informatics
ウッティチャイ アムポンアラムウェーツ
国立情報学研究所

Haruki UENO
National Institute of Informatics
上野 晴樹
国立情報学研究所

ABSTRACT

Symbiotic information system is a concept aiming for a user-friendly information system by establishing a symbiosis relationship of human and the information systems. Applying this concept to human operators and networked robots, this paper describes an on-going design and implementation processes of a software platform to support collaborative operations of human operators and networked robots. The software platform is based on the concept of autonomous agents providing abstractions of sophisticated robot hardware interfaces and extensibility. The implemented system allows multiple users to collaboratively operate on a robot, or a group of robots at the same time. Mobile agents are employed to coordinate the accesses to the shared robots through vote-casting, or shared control brokering mechanism.

要旨

共生情報システムは、人間および情報システムの共生関係を築くことにより、ユーザー・フレンドリーな情報システムを目指して進む概念である。人間オペレーターやネットワークに繋がるロボットにこの概念を適用し、本論文は、人間・ロボットの協調作業を支援するためのソフトウェアプラットフォームの進行中の設計および実現について述べる。このソフトウェアプラットフォームは、自律的なエージェントの概念に基づき、複雑なロボット・ハードウェア・インターフェースを隠しながら、拡張性を持つ共生環境を提供する。実現されたシステムは、多数のユーザーにより協力的に多数のロボットの同時作動を許可する。共有されるロボットへのアクセスを調整するために、モバイルエージェントを雇用する。

[Keywords]
Telerobotics, Symbiotic Information System, Software Agents

[キーワード]
遠隔ロボット操作, 共生情報システム, エージェント技術

1 Introduction

Currently, access to the Internet still requires significant amount of training. The Internet and recent information technologies are not yet ready for everybody. Believing that all citizens have “the right to use” of any information by means of advanced information technologies is the motivation of a recently launched research program on the topic of “Symbiotic Information Systems (SIS) – Concepts formation and technology development” at the National Institute of Informatics, a member of the Center of Excellence (COE) research institutes of Japan [1].
Symbiotic Information System (SIS) can be characterized as the natural symbiosis of human and the information systems. The meaning of the word “symbiosis”, as defined in the American Heritage Dictionary, is “A close, prolonged association between two or more different organisms of different species that may, but does not necessarily, benefit each member.” Recently, this term has been used to describe similar relationships among broader range of entities. For example, the article “Knowledge Creation – The Symbiotic partnership of University, Government, and Industry” \[2\] introduces the relationship among institutes and organizations. Similarly, the term is also used to describe the relationship between human and robots in works relating to robots that assist human \[3\].

For our researches, the term “Symbiosis” is defined as “The community in which a number of autonomous entities pay respect and co-operate for benefit of each other”. Thus, a symbiotic information system is an information system whose elements include human beings and are designed based on the concept of co-operative operations or symbiosis. \[4\]

Our system consists of a group of robots operating under a networking environment. All the robots can communicate with each other, and with any network equipment using same standard TCP/IP protocol as used on the Internet. In the system, there are three types of robots:- Melfa robot arm, Scout mobile robot, and Robovie, a human-like mobile robot. The diagram of this robot environment is shown in figure 1.

At the current state, we are focusing on design and development of a software platform to support man-machine collaborative operations within this distributed working environment. The main objective is to come up with a system of intelligent autonomous robots capable of interacting with human using natural communication forms. Any users should be able to access the robots from anywhere with minimal operation training. At the same time, robots should understand high-level commands and be able to adapt themselves to difference in each human user. Autonomous agent, voice recognition, and WWW technologies are widely adopted in the system.

Initially, we chose the topic of human-friendly telerobotics as a case study of SIS research because such a system consists of various autonomous elements, which must collaborate in a realistic environment, and is attractive in showing the concepts and technologies to ordinary persons outside the field.

![System Diagram](image_url)
on the other hand there is a desire to further abstract the
details of the human-to-program interface by delegating to
agents the details of specifying and carrying out complex
tasks [5]. This is the basic idea behind the design of our
system architecture. Agents can help simplifying the com-
plexities of distributed computing and overcoming the
limitations of some ad hoc user interface approaches.

3.1 Software Agents

Our system consists of four types of agents correspond-
ing to each physical objects and functional units: 1) Robot
Agent, representing the robot under control, 2) User Agent,
relaying commands from remote user, 3) Server Agent,
providing communication among all agents, resolving
command conflicts from multiple users, and 4) World
Agent, monitoring the states of objects inside the work-
space environment. These agents are called “host agent” as
they can host some task-specific Mobile Agents, represent-
ing mobile control unit such as camera controller, robot
controller, video display, vote-casting dialog, etc. The mo-
bile agent may be transferred from one host agent to an-
other (figure 3). It maintains its internal state while moving
itself among these host agents. Within this community,
some types of agent may exist in multiple as in the case of
multiple robots, or multiple users. A task is usually per-
formed via communications among these agents.

All host and mobile agents are implemented as Java
classes derived from their respective base classes. In-
stances of these agent classes are running autonomously on
separate execution environments. They may be running on
different workstations, or as different threads on the same
workstation, depending on its functions and location to the
real physical robots/objects. Their behaviors are defined by
two sets of forward and backward chaining rules as ex-
plained in section 3.3. Each host agent maintains a list of
methods for other agents to access, and events to which
other agents can subscribe by calling “subscribe()”
method, provided by the host agent base class.
Class AgentBase

Methods: subscribe(EventName)

Messages are sent to the event subscribers whenever that particular event occurs. The communication among agents in our system is shown in figure 2. Some of the host agent classes and their methods, events implemented in our system are:

Class ServerAgent

The server agent serves as a message-switching hub, a center for relaying messages among the robot and user agents. The agent maintains a list of active agents and dispatches separate threads for each of the agents in order to optimize the response time. Some broadcast messages can be originated from the server agent, to notify the community of newly participated or retired members.

Server agent is also in charge of relaying video streams. Transmission of real-time video from environment camera, robot camera, and user camera to the agents was implemented by sending a sequence of still JPEG image files to the targets.

When there is a conflict in user commands, the server agent becomes an arbitrator trying to resolve that conflict by means of vote-casting or limiting exclusive control of a particular robot to one user at a time.

Class WorldAgent

The WorldAgent monitors the workspace and laboratory environment by video cameras. The camera movement can be controlled directly from other agents using the provided pan(), tilt(), zoom() methods. Kanade-Lucas feature tracking algorithm was integrated with camera controller into a closed-loop control to add arbitrary pattern tracking ability to the camera. The command that instructs camera to track the movement of the pattern at the specified image coordinate is also provided. Robot agents can query camera agent for the position of object with the given color. Images from the camera can be delivered to NewVideoFrame event subscribers.

Class RobotArm extends AgentBase

Properties: Hand position, grip status

Methods: Move(dx, dy, dZ), Go(x, y, z),
Pick(Object)

Events: Error

However, the agent needs to communicate with external agent in order to achieve the high level tasks. For example, the agent needs to send a query of “Where is object A” to the world agent before it can process the request “move object A to <x, y>”.

Class MelfaManipulatorAgent

For the robot arm in figure 4 (left), methods are provided to accept both low level commands (move, go), and high level one (pick object). Notification can be made when an error occurs.

Class UserAgent

A User Agent represents each human user on the system, relays commands from the user to other agents, queries states of the robot agents, and provides the user with

Figure 4: Melfa Robot Arm (left) and Mobile Robot (right) in the System

Figure 5: Browser-based User Interface: Java Applet and Agent Character
enough feedback information. Because it needs to interact
with user operating remotely, the user agent has two parts:
a server-side Java code communicating with other agents
on the host computer, and a client-side Java applet running
on the user’s browser screen as shown in figure 5. Interac-
tion with users is done using standard web browser, in
combination with an animated character. The character can
communicate to user with synthesized voice and under-
stand simple voice commands. Behavior of the character
can be controlled by the methods provided.

Class UserAgent extends AgentBase
Properties: username/userID
Methods: speak(), ask(),
characterAction(bye|hello|…)
Events: Logout, Login

Information relating to current user is queried from the
user camera, which monitors the user who is currently
operating the system. This is done by first using a simple
optical-flow algorithm to identify the existence and loca-
tion of a moving object in front of the user camera. Then a
number of image processing algorithms [6], are used to
locate and track the location of a human face on the scene,
and identify who the user is by comparing it against the
database of known user faces. Example of the output of
these algorithms is shown in figure 6. This user informa-
tion is then made available to other agents.

Class MobileRobots extends AgentBase
Properties: States(Run/Stop),
Current Location
Methods: wander(), search(Color),
speak("Word")
Events: LowPower, FoundObstacle,
FoundObject, NewVideoFrame

class MobileAgentBase

The mobile agents are derived from MobileAgentBase
class.

Class MobileAgentBase
Properties: States(Active/Inactive),
Methods: Run(), Suspend(),
SubmitCommand(Command)

This MobileAgentBase class provides basic
communication with its host agent, and other mobile
agents via the SubmitCommand method. Operation of the
agent can be paused (suspend()) and resumed (Run())
before and after the mobile agent is moved from one host
agent to another respectively. Because this class is a
subclass of Java awt’s Frame class, all mobile agents
appear on user screen as an individual window. An exa-
mple of the robot arm controller mobile agent is shown in
figure 7. Depending on the control mode, the number of
this mobile agent on the system can be one (for exclusive
control mode), or equal to the number of remote users
(shared mode). In the latter case, the mobile agents are
distributed to users requesting for control of the robot.

The host agents communicate with each other by passing
messages. Messages are implemented as instances of Java
class representing the message. There are a number of sup-
ported message classes. All of them are subclasses of a

![Figure 6: User Face Recognition](image)

![Figure 7: Mobile Agent for Robot Arm Controller](image)
Software Platform for Symbiotic Operations of Human and Networked Robots

common MessageBase base class as shown in the class hierarchy in figure 8.

Information regarding the message sender and intended recipient can be obtained by calling the corresponding methods provided by the base class. The instances of mobile agent are encapsulated in their own classes derived from Maction class, whose run() method is invoked when the mobile agent lands on a new host. Currently, the event notification and method invocation are sent using the Mcommand message class.

3.3 Agent Reasoning

Autonomy is achieved by incorporating rule-based reasoning engine into the agent [7][8]. The agent maintains its knowledge of the World as a hierarchy of Java classes implementing the forward and backward chaining rules. Agent’s knowledge is stored as state variables in the “working memory” area, which is updated by event-processing rules upon receiving messages from other agents or being triggered by local hardware or software notifications. These rules are declared as a list of forward-chaining if-then statements. For example, basic rules for processing message queue, common to all agents, can be declared as:-

\[
\begin{align*}
\text{If (notempty (received_buffer))}: & \quad \text{queue\_command (received\_buffer)}. \\
\text{If (notempty (command\_queue))}: & \quad \text{process\_command (command\_queue)}. \\
\text{If (notempty (Error))}: & \quad \text{notify\_user (Error), empty (Error)}. 
\end{align*}
\]

The first rule puts the received command message in processing queue when it arrives. The second rule processes the command at the head of the queue. The third rule monitors the variable “Error” in local working memory and reports it to the user who submitted the command. These rules are generally evaluated when there are changes in working memory, and processed by separated execution threads. Thus, multiple rules can be fired simultaneously.

These forward-chaining rules are used to process all the agent events. For example, the World agent and Scout agent with video camera contains the following rules for broadcasting the video frames.

\[
\begin{align*}
\text{If (newframe)}: & \quad \text{broadcast\_frame (newframe), rec\_face (newframe), rec\_object (newframe)}. \\
\text{If (known\_face)}: & \quad \text{notify\_subscriber (known\_face\_subscriber, known\_face)}. \\
\text{If (target == known\_object)}: & \quad \text{notify\_subscriber (caller, target)}. 
\end{align*}
\]

The first rule fires when the hardware frame grabber puts a new frame in “newframe” working memory. The rule then tries to recognize the image by calling rec_face() and rec_object() functions. If a known face is found or the object matches the search target, the corresponding subscriber list is notified.

In processing the message commands and agent methods, another set of backward-chaining command-processing rules is provided. By borrowing the syntax of Prolog programming language, these rules are shown in figure 9. Each rule in this figure consists of the method name on the LHS, a separator (:-), and the RHS. The expressions on the RHS must all be satisfied for the LHS to be processed. Similar to Prolog, there can be multiple rule entries for the same method. The first rule entry shown up in the rule set is evaluated first. The method is fulfilled if RHS expressions are all satisfied. The successive rule entries will be tried if any of the RHS expressions fails. Lines beginning with “%” are comments.

For example, some rules from Scout mobile robot and the Melfa robot arm are shown in this figure. Scout robot can be instructed to wander and search for an object using “search()” method. The definition of this method shows that the task is done by setting the working memory variable “motormode” to “wander” and “target to the given object. These changes in working memory variables will be detected by the event-processing rules, which consequently put the robot in the specified operation mode. As explained
earlier, the event-processing rules will also automatically notify the user when the robot finds the specified object. set_wm() and get_wm() are provided for accessing the working memory variables.

Pick(Object) method and related rules for the Melfa robot agent is shown in the lower part of the figure. There are three rule entries for pick() method. The first rule indicates that pick() is already satisfied if the robot is already holding the specified object. Should the first rule fails, the second rule gives list of expressions (methods) to process in sequential order to perform pick(). Should one of the expression fails, the third rule returns error message to the user. Call() is used to invoke low-level hardware interface functions.

Note that this Prolog syntax is used for explanation only. The rules are implemented as Java codes in our experimental system.
The Experimental System

In our experimental system, the robots under control are a six-axis MELFA industrial robot arm with an inverted U-shape hand gripper, a scout mobile robot with a video camera, and a robovie robot as shown in figures 1 and 4. T-Shape LEGO objects in red and green color are put on the table for the robot hand to manipulate. The robot movement is controlled by a Linux PC, on which the agent software is running. Two video cameras are connected to frame grabbers on another PC running image-processing software, which calculates three-dimensional object locations and feeds them to the World agent. The mobile robot agent is running on the robot itself, which carries a PC board also running the Linux operating system. The robot connects to the laboratory network thru a wireless LAN.

Java was chosen as the primary language for our system development because its object-oriented nature can greatly simplify the development of agents and their interactions. Java also poses the benefits of platform-independent, having a rich set of ready-to-use classes, including the RMI for remote invocation, JMF for streaming multimedia, object serialization for packaging and transferring of mobile agents and GUI layouts.

4.1 Scenario

The interaction among these agents and how this system allows sharing access to multiple robots from multiple web users can be best explained by considering the scenario below.

1. During the startup process, the Server agent is launched first so that it can monitor the registration of other agents.

2. World agent, robot agents and the corresponding hardware are initialized. Server agent receives notification from these agents and registers them to the system.

3. When a user login, a new UserAgent is created and attached to the corresponding user.

4. User agent queries for world information from the server agent, and presents to the user, a list of available robots in the system. Information regarding the list of available video camera feeds, or robots to operate can also be obtained manually later by clicking on the “VIEWS” and “OBJECTS” buttons in figure 5.

5. The user selects the robot he/she would like to operate. User agent submits the request to the robot agent. The robot agent sends back to the user, the mobile agent representing the robot controller (figure 7).

6. Similarly, the user may request the World agent to send VideoView mobile agent displaying video streams of any cameras he/she would like to view (figure 10). Multiple-view is also supported, as there is no restriction on the number of concurrent mobile agents on the user side. Optionally, the camera control mobile agent can also be requested (figure 11).

7. The user (user A) submits a low-level command, “moveto (x,y,z)”, to control robot. User agent passes the command to the robot agent.

8. If another user (user B) logins and submits another low-level command, “close gripper”, while the moveto command is in progress. The robot agent also executes this command since there is no conflict between these two commands.

9. UserA may later submit a high-level command, “pick red object”. In this case, the robot agent must first query the World agent for current position of “red object”, and then it can perform the pick operation. If UserB happens to submit a conflict command “pick green object”, the robot agent will need to ask the server agent, as an arbitrator, to rule out the color of object to pick. Server agent may create a vote-casting mobile agent (figure 12) and forward it to each user agents in turn to come up with a public consensus.

Discussions & Conclusion

In this system, the design using autonomous agents allows us to provide the end users with both low-level “moveto(x,y)” and high-level “pickup(red object)” commands which require less interaction between the system and remote user, and hence, lower the network bandwidth requirement. This results in higher robustness against the
network time delay, and increased user satisfaction. The cooperative operations among agents in resolving the command conflicts also provide support for simultaneous operations of multiple concurrent users. User interface is enhanced by the use of character agent with voice synthesizer and voice recognition to communicate with user in real voice.

It is important to note that, at the current state, we can provide only access to basic functionalities of each robot. Robots are controlled independently and there are yet no global objectives for this group of robots to achieve. Most of the communications taken place are between human operators and the robots. It will be more interesting if a global task requiring collaborative operations of the robots is assigned in the future.

As for the system improvement, current support for voice command is limited to basic operations of the on-screen buttons and input controls. Higher level of natural language processing ability is still needed to achieve the system, which can interact with users more naturally. Though preliminary results suggest improvement over the ordinary telerobotic systems in term of user satisfactory and ease of use, further experiments on a broader range of Internet user are still needed to confirm the conclusion.

Acknowledgements

The authors express our sincere thanks to Prof. Shin-yo Muto, Dr. Md. Al-Amin Bhuiyan, and Dr. Tomo’o Inoue for their cooperation in this project.

References