Regular Paper

An Inter-Space Avatar Programming Language in a Virtual Space on the WWW

Yin-Huei Loh,[†] Takefumi Ogawa,[††] Masahiko Tsukamoto[†††] and Shojiro Nishio[†††]

In recent years, research work on virtual space has been extensively carried out. Many virtual spaces are formed by multiple subspaces and the user uses an avatar to represent him moving around in the virtual space. In general, moving around in virtual spaces requires real-time user control. To make the avatar control more efficient and interesting, we propose the Avatar Programming language (AP) to program the avatar's behavior in a virtual space. To maintain the independency of subspaces which form the virtual space, we employ the internal control approach where the program handler is built inside each subspace. We implemented the system on a virtual space constructed with IBNR (Image-Based Non-Rendering) [11], an image-based representation technique to construct virtual spaces which run on a normal web browser on the WWW. Using AP, the avatar's behavior can be flexibly predefined beforehand, and we can save the users much time and trouble in controlling the avatar in a virtual space.

1. Introduction

In recent years, research work on virtual space has been extensively carried out [2),5),6)]. Virtual spaces can be used to simulate many things in real space; visiting remote museums, art galleries, famous tourist resorts virtually, shopping in virtual shopping malls, borrowing books from virtual libraries and so on. Since virtual spaces are usually huge, they often consist of multiple virtual subspaces.

In order for users to move around in a virtual space, some systems employ the method of using an avatar to represent the user in the virtual space. By controlling the avatar, the user can see himself moving or carrying out various kinds of activities in the virtual space. In controlling the avatar, real-time user interaction is often required where the user controls the avatar's behavior at the point of time the behavior is desired. To make the avatar control more efficient and interesting, in this paper, we propose a programming language which can be used to preprogram the behavior of the avatar. We name it the avatar programming language (AP). In AP, a program is executed across different subspaces in a virtual space on the WWW. We designed the language structure based on several policies, propose the program executing mechanism to execute the program across different subspaces, and we implemented the system on IBNR (Image-Based Non-Rendering) [11], a method to construct virtual space systems which can run on the WWW.

By defining a program using AP, users can give instructions to the avatar in advance as of how it should act. Given the instruction, the avatar is able to act autonomously and more intelligently in the virtual space without the need to be controlled by the user at real-time. In such a way, the avatar can be used to perform various services to the user. To help the user, the virtual space constructor can also predefine some programs in the virtual space. The program can be executed under certain circumstances when a user visits the virtual space.

The remainder of the paper is organized as follows. In section 2, we explain the approach taken in realizing the avatar programming language. We describe the programming language structure in section 3 and explain the program execution mechanism in section 4. Discussion on sample usages, observation and related work are included in section 5. Finally, we conclude our paper in section 6 with some discussion about future work.

2. The Approach

2.1 Language Design Policy

AP is designed as an inter-space avatar pro-

† Department of Information Systems Engineering, Graduate School of Engineering, Osaka University
†† Infomedia Education Division, Cybermedia Center, Osaka University
††† Department of Multimedia Engineering, Graduate School of Information Science and Technology, Osaka University

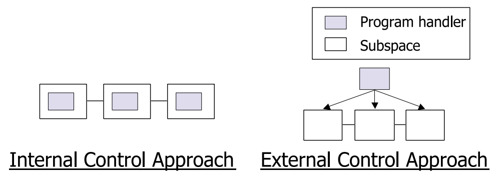


Fig. 1 The internal control approach versus the external control approach.

gramming language in a virtual space on the WWW. To achieve this, we have employed the design policies as listed below.

- Ability to use the program on the WWW.
- Ability to maintain the independency of subspaces on the WWW.
- Comprehensibility and readability of the program by users and virtual space constructors.

In order to maximize the number of users of the programming language, we have decided to make it usable on the WWW as this is the most common platform and can be accessed by anyone using any operating systems from anywhere around the world. We have chosen IBNR as the platform to implement the programming language, because an IBNR virtual space runs on Internet Explorer. A brief introduction of IBNR is included in the next subsection. The avatar programming language is implemented with JavaScript program which runs on the web client.

In a virtual space consisting of a number of independent subspaces, the avatar moves across these subspaces, and the control is passed along from a subspace to another. There are two methods to handle the program in the virtual space, the internal control approach and the external control approach. **Figure 1** illustrates the concept of these two approaches. A program handler is in charge of handling the program, from program input, processing to execution. In the internal control approach, the program handler is built as a component inside the subspace. When the avatar moves from a subspace to another, the control of the program is also passed along from the program handler in the previous subspace to the program handler in the current subspace. In such a way, the independency of the subspaces can be maintained as different program handlers can be used for different subspaces.

As opposed to the internal control approach, the external control approach can be used where the program handler is built as an external component to the subspaces. In the external control approach, since the program handler runs as a resident process to control and execute the program, it is relatively easy to execute the program across different subspaces.

However, since the internal control approach allows different subspaces to have different program handlers, they can carry different definitions for the same commands, and it is possible for a subspace to have some special commands which do not exist in the other subspace. When using the external control approach for the same purposes, everything has to be included in the same program handler which makes it complicated and large. In other words, using the internal control approach makes it easy to maintain the independency of subspaces. Consequently, we have taken the internal control approach.

As the internal control approach is chosen, it is necessary to propagate the program from a subspace to another subspace. We have taken the approach to save the program in the cookie, a function provided by web browsers where values stored in a cookie can be shared by different web pages as long as they are on the same server. In such a way, the program can be referred to by different subspaces.

To minimize the data volume propagated, it is necessary to make the program small in size. On top of that, considering the fact that the program is stored in cookie which has limitation in size, it is also logical to make the program simple and short. As a consequence, we have employed the **one-alphabet command/one-line programming** policy where each command consists of only one alphabet with the necessary parameters so that the program can be written in one line.

Finally, considering the fact that the program is used by virtual space users and constructors, the program must be made understandable and readable by human beings. Therefore, the commands are all designed in the Polish form (prefix style) where the operator is first written followed by the operand (parameter) enclosed in parenthesis. This is more readable to human beings compared to the reverse Polish form (suffix style) where the operator is placed after the operand (parameter). As an example, the expression $(a+b)*c$ is written as $*(c+(ab))$ in the Polish form and $((ab)+c)*$ in the reverse Polish form.

With the above, we have designed the programming language and implemented it on IBNR on the web. The system can be executed on a normal web browser without any extra plug-ins or external software. No special resident process on the web server or clients are executed to handle the program. Instead, a JavaScript program is executed on the web browser of the client to handle the program. We will explain IBNR in the next subsection.

2.2 IBNR

In order to construct a virtual space, the geometry-based rendering (GBR) technique[6] and the image-based rendering (IBR) technique[2),5)] are often employed. Since the real space is complex and has many objects, it is very costly to construct a virtual space with high reality using GBR. With IBR, this task becomes easier, but rendering is still necessary. These conventional approaches in building virtual 3D space require much calculation and consume a great deal of computational power, sometimes affecting the processing speed of the applications when a lot of changes or motions are involved in the scene. Thus, rendering speed is a main issue and very often, special application programs are also required.

In order to solve these problems, we have proposed IBNR (image-based non-rendering)[11], an imaged-based representation method. In this method, a virtual space is constructed using many independent subspaces we call scenes, each representing a part of the virtual space. Each scene is constructed using a static picture of the real space as the background of the scene. The avatar which represents a user in the virtual space is formed by pasting a picture of a person with a suitable size on the background of the scene.

Users can control the movement of the avatar using some predefined keys, and the avatar's size changes accordingly when it is moved. Many scenes can be linked together to form a huge virtual space. When the avatar reaches the edge of a scene, the background scene is changed to another scene to reflect the movement. **Figure 2** (a) to (c) show how the avatar's size changes when it moves. As the avatar reaches the edge of the scene, the background scene changes to another scene as shown in Fig. 2 (d).

The virtual spaces constructed by IBNR can be used on the WWW because in IBNR, every scene is defined with an HTML file as shown in **Fig. 3**. All parameters of the scenes and

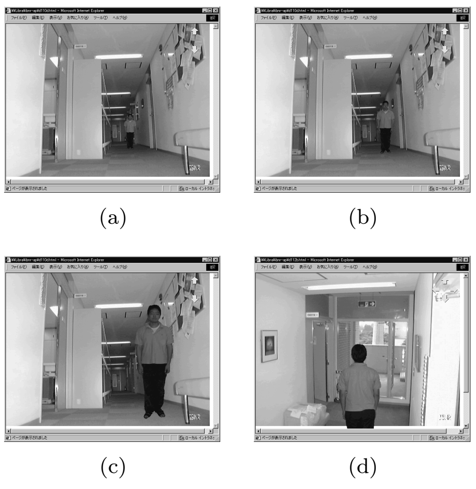


Fig. 2 Sample IBNR scenes showing scene changes when the avatar moves.

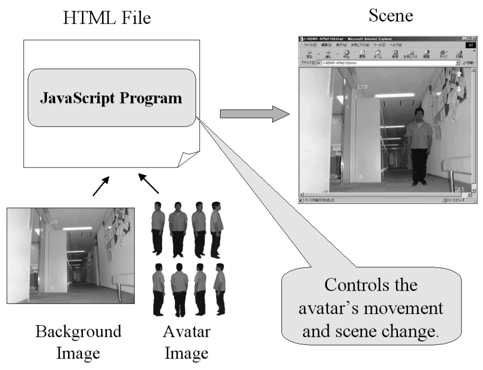


Fig. 3 IBNR system.

a JavaScript program are included in the corresponding HTML files. This JavaScript program controls the avatar's movement and scene change. When the avatar moves, the JavaScript program reflects its movement on the scene; when the avatar moves out from a scene, the JavaScript program loads the corresponding adjacent scene to the web browser. We have proposed some editing tools[7] to construct scenes from the pictures of the real world and link them together to form a virtual space. Some sample applications that we have built using IBNR can be found on the IBNR homepage[3)☆].

We have chosen IBNR because an IBNR vir-

☆ JavaScript for Netscape Navigator (NN) and Microsoft Internet Explorer (IE) are not compatible with each other. Due to this constraint, our system only works well on IE browser. However, it can be transformed into Java versions to work on NN browser.

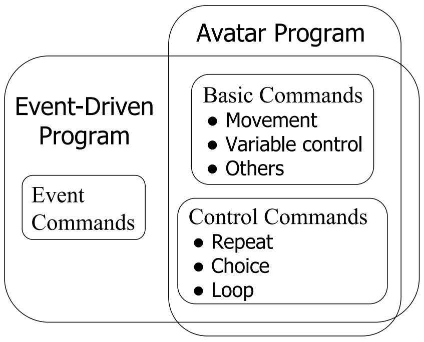


Fig. 4 Classification of programs and commands.

tual space consists of many subspaces; thus it reflects the idea of the internal control approach best. In this paper, we use IBNR as the base of our system to implement the programming language.

3. The Language Structure

In the avatar programming language we propose, we provide two types of program as follows:

- Avatar program
- Event-driven program

The avatar program is a sequence of actions formed by the basic commands and the control commands (see **Fig. 4**). When the avatar program is executed, the avatar performs the corresponding actions specified in the program sequentially. The avatar program is a one-time program. It is executed from the beginning to the end once, and then deleted from the memory.

The event-driven program defines an avatar program to be executed when an event occurs. It is defined in pairs of event-actions where the "event" is one of the event commands while the "action" is an instance of the avatar program (a string of combination of the basic and control commands). When the specified events occur, the corresponding actions will be inserted into the avatar program and then executed. Unlike the avatar program, after the event-driven program is defined, it stays in memory and waits for the event to happen to activate the action. It remains in memory after it is being executed unless being explicitly deleted. In other words, it may be executed multiple times since the action is activated every time the specified event happens.

Figure 4 illustrates the classification of programs, commands and their relation.

Since the programming language is used to instruct the avatar to perform various actions, the language should be made able to define a variety of different instructions flexibly. To accomplish this, we prepare a lot of primitive commands in the language which can form many different actions when they are combined in different manners. We explain them in the following.

3.1 Basic Commands

Basic commands are commands which specify a unit of action of an avatar. They are stand alone commands which can be used independently without being combined with any other commands. They can be further divided into the avatar's movement, variable control commands and other actions.

Avatar's movement are the most basic commands in a virtual space to move the avatar around. Commands to track the avatar's movement are also included in which by recording the movement of the avatar, the program to move from a place to another can be saved into a variable and be used later. The commands related to avatar's movement are listed in **Table 1**.

Variables are used to keep various information in a virtual space. In **Table 2**, *program* is used to store the avatar program, while *eprogram* is used to store the event-driven program. There are 3 system variables, *reg* (register), *cnt* (counter) and *pace* (avatar's moving pace). *reg* is used to store information temporarily; information stored can be manipulated with some operations. *cnt* is used to control a loop command as of when the loop should end (will be explained in the next subsection). *pace* represents the avatar's moving pace where a smaller value makes the avatar move faster and vice versa. A user can also create his own variables we call user-defined variables. Using user-defined variables, a user can store as much information as he likes and organize them in his own way. To manipulate these variables with some operations, the users can load these variables into *reg* and save them back after manipulation. The virtual space constructor can also define some variables inside each subspace to provide users the information about the virtual space. The variables can be manipulated with the variable control commands as follows.

For commands "+", "−", "z", "Z", "A" and "D", the parameter can be a value, or it can refer to a variable name when the symbol "#" is added. When "#" is used, it refers to the

Table 1 Avatar's movement.

f	Move to the scene in front of the avatar (relative direction).
b	Move to the scene at the back of the avatar (relative direction).
r	Move to the scene at the right of the avatar (relative direction).
l	Move to the scene at the left of the avatar (relative direction).
F	Move to the scene in front from the user's point of view (absolute direction).
B	Move to the scene at the back from the user's point of view (absolute direction).
R	Move to the scene at the right from the user's point of view (absolute direction).
L	Move to the scene at the left from the user's point of view (absolute direction).
K(t)	Move forward one step.
K(b)	Move backward one step.
K(h)	Turn right.
K(f)	Turn left.
H	Start recording the movement of the avatar in a forward manner.
T	Start recording the movement of the avatar in a backward manner.
Q	Stop the recording of movement.
X	Execute the movement recorded.

Table 2 Variable control commands.

s(n)	Set *reg* to n ($reg = n$).
+(n)	Add the number n to *reg* ($reg = reg + n$).
−(n)	Subtract the number n from *reg* ($reg = reg - n$).
z(*str*)	Concatenate the string *str* to the front of *reg*.
Z(*str*)	Concatenate the string *str* to the end of *reg*.
A(n)	Add n as an element to the set stored in *reg*.
D(n)	Remove element n from the set stored in *reg*.
u	Set the value of *reg* to the URL of the current scene.
x	Set *reg* to *program* ($reg = program$).
X	Set *program* to *reg* ($program = reg$).
e	Append *reg* to the back of *eprogram*.
E	Overwrite *eprogram* with the the event-driven program stored in *reg*.
i	Increase *cnt* by 1.
d	Decrease *cnt* by 1.
c	Set *cnt* to *reg* ($cnt = reg$).
C	Set *reg* to *cnt* ($reg = cnt$).
P(n)	Set *pace* to n ($pace = n$).
S(*var*)	Set *var* to *reg* ($var = reg$).
G(*var*)	Set *reg* to *var* ($reg = var$).
k(*var*)	Destroy variable *var*.
v(*var*)	Display the value of variable *var* on screen.

Table 3 Other actions.

t(x)	Display the message x on screen.
@(x)($p1$):($p2$)	Display a dialog box with the question x on screen; if the user chooses the "ok" button, execute the program $p1$, otherwise if he chooses the "cancel" button, execute the program $p2$.
w(n)	Wait for n milliseconds before executing the next command.
j(x)	Jump to the URL x.
J	Jump to the URL stored in *reg*.

value stored in the variable. For example, "+(#*anum*)" will add the value stored in variable *anum* to *reg*. In such a way, values stored in different variables can be added up or combined together easily.

var is used to represent any user-defined variables. By using the "S" and "G" commands, user-defined variables can be loaded into *reg*, some operations can be performed on the value in the *reg* and then the value can be loaded back to the user-defined variables.

Commands provided for other actions are listed in **Table 3**.

In "@" command, both $p1$ and $p2$ are avatar programs formed by one or more commands. In order to save the processing cost, the syntax of "@" command is made similar to those in the choice commands, where the command is divided into three portions: the "if" portion "@(x)", the "then" portion "($p1$)" followed by a colon ":", and the "else" portion "($p2$)". Either of the "then" or the "else" portion can be omitted.

Table 4 Repeat commands.

$n(p1)$	Execute program $p1$ n times, where n is a positive integer.
$(p1)$	Execute program $p1$ indefinitely.

Table 5 Choice and loop commands.

?$m(p1)$:$(p2)$	If movement m is possible, execute program $p1$, otherwise execute program $p2$. m can be either one of "f", "b", "r", "l", "F", "B", "R", "L" which indicate the corresponding movement to the adjacent scenes, or "s", which indicates *move one step to the front without exiting the current scene.*
?$(val)(p1)$:$(p2)$	If the value of *reg* is equal to *val*, execute program $p1$, otherwise execute program $p2$.
p$(p1)$:$(p2)$	If the value of *reg* is more than 0, execute program $p1$, otherwise execute program $p2$.
!$(p1)$	While *cnt* is not 0, execute program $p1$.

3.2 Control commands

Control commands are used for controlling the basic commands but they can also be used recursively on themselves. These commands cannot be used independently; they must include the basic commands as parameters in order to make sense. Control commands can be further divided into two groups, the repeat commands and the choice and loop commands.

Repeat commands (**Table 4**) are used to specify the number of times a command (or a group of commands) should be repeated. They can shorten the program when there are many similar actions to be performed.

Repeat commands' structure has some similarity with the Logo programming language[4], where in Logo, the cursor is moved around the screen by using simple commands like "right 40" and "forward 20".

Choice and loop commands (**Table 5**) refer to commands which adaptively change according to the situation. In a choice command, a condition is stated and different commands can be carried out when the condition is and is not fulfilled. This is equivalent to the "if-then-else" command in common programming languages. Loop command is an improvement over the repeat commands where the number of repeats does not need to be fixed but can be changed dynamically according to the situation. It is equivalent to the "do-while" command in common programming languages. The commands provided are as follows.

Since choice and loop commands are adaptive and they control the flow of the program, they make the avatar program more flexible.

3.3 Event Commands

The last category of commands is the event commands as shown in **Table 6**. These commands are used in the event-driven program to specify the events which will activate an avatar program (formed by a combination of basic and/or control commands) to be executed. A number of events and the corresponding actions can be defined. The possible events are as follows.

Table 6 Event commands.

i	The avatar enters a scene.
o	The avatar leaves a scene.
m	The avatar makes a move.
f	The avatar moves forward.
b	The avatar moves backward.
r	The avatar moves to the right.
l	The avatar moves to the left.

The syntax of the event-driven program is $e_1\hat{}p_1\&e_2\hat{}p_2\&\ldots\&e_n\hat{}p_n$, where e_i represents an event command while p_i represents an avatar program. They are separated with "ˆ", and pairs of event-program are separated with "&". In the above, whenever $e1$ occurs, $p1$ will be inserted into the avatar program and executed. The same thing happens for $e2$ and $p2$ and so on.

3.4 Sample Programs

Combining the above commands, various programs can be formed. Below, we list some possible avatar programs.

2fr Go forward two scenes, then go rightward one scene.

s(2)c!(f?l(d))l Make the avatar move forward and take the second left turn.

P(100)2fP(400)w(200)bf Rush to two scenes in front, wait for a while, and then slowly move back to the current scene.

s(0)3(f?l(+(1)))p(b2f) Check whether there are left turns to any of the three scenes in front. If there exists at least one such scene, go back to the original starting point.

s(1)cs(0)!(?s(+(1)K(t)):(d))v(*reg***)** Calculate the number of steps needed to reach the edge of a scene.

p:(@(Would you like to go back to the beginning?)(j(http://www.test.com/begin.html))) If *reg* is not positive, ask the user whether he wants to go back to the beginning and if he answers yes, load the web page which shows the starting scene.

s(1)cs(0)S(*scnnum*)!(G(*scn*)?(balcony)(d):(G(*scnnum*)+(1)S(*scnnum*)f)) v(*scnnum*) Calculate the number of scenes to go forward before reaching the balcony scene. Here, *scn* is prepared by the virtual space constructor beforehand as a description of the scene.

Avatar programs can be even more powerful when used in event-driven programs. Here, we list some sample event-driven programs.

m^G(*step*)+(1)S(*step*) Record the number of steps the avatar has taken.

i^G(*scn*)?(balcony):(G(*scnnum*)+(1)S(*scnnum*)f) Count the number of scenes to go through to the balcony scene.

i^G(*scnname*)A(#*scn*)S(*scnname*) Collect all the scene names an avatar goes through. Similar to the above, *scn* is a description of the scene prepared by the virtual space constructor beforehand.

The event-driven program can be activated when the avatar moves either by an avatar program or by user's real-time control.

As shown in the above examples, various programs can be formed flexibly with the commands provided. The program execution mechanism is explained next.

4. Program Execution Mechanism

In this section, we explain the program input mechanism, the details of the program handler and the program propagation mechanism.

4.1 Program Input

Program input can be done by the user using the user interface we provided. In a scene of an IBNR virtual space, a user simply presses the key "s" (for script), and a dialog box asking the user to key in the avatar program will appear on the scene to let the user key in the program. To key in an event-driven program, a user can press the key "e" (for event-driven script). Similarly, a dialog box asking the user to key in the event-driven program will appear on the scene.

The virtual space constructor can also predefine the programs in the HTML files which form the scenes. This can be done with the "saveProgram" and "saveEvent" functions AP provides. The details will be explained in section 4.4.

(event)	$\dfrac{A\Delta E}{B_i A\Delta E}$	e_i occurred ($E = e_1\hat{}B_1 \& e_2\hat{}B_2 \& \ldots \& e_n\hat{}B_n$)
(basic)	$\dfrac{bA\Delta E}{A\Delta E}$	execute b
(replace)	$\dfrac{\mathbf{e}A\Delta E}{A\Delta E^*}$	$E^* = reg$
(append)	$\dfrac{\mathbf{E}A\Delta E}{A\Delta E \& E^*}$	$E^* = reg$
(program)	$\dfrac{\mathbf{X}A\Delta E}{A^*\Delta E}$	$A^* = reg$
(repeat)	$\dfrac{S_n aA\Delta E}{aS_{n-1}aA\Delta E}$	if $n > 1$
(repeat1)	$\dfrac{1aA\Delta E}{aA\Delta E}$	
(indefinite)	$\dfrac{*aA\Delta E}{a * aA\Delta E}$	
(if-true)	$\dfrac{?CZ_1 : Z_2 A\Delta E}{Z_1 A\Delta E}$	if C (Z_1 or Z_2 must not be null)
(if-false)	$\dfrac{?CZ_1 : Z_2 A\Delta E}{Z_2 A\Delta E}$	if not C (Z_1 or Z_2 must not be null)
(reg-pos)	$\dfrac{pZ_1 : Z_2 A\Delta E}{Z_1 A\Delta E}$	if $reg > 0$ (Z_1 or Z_2 must not be null)
(reg-npos)	$\dfrac{pZ_1 : Z_2 A\Delta E}{Z_2 A\Delta E}$	if $reg \leq 0$ (Z_1 or Z_2 must not be null)
(loop)	$\dfrac{!(A)B\Delta E}{A!(A)B\Delta E}$	if $cnt \neq 0$
(loop-end)	$\dfrac{!(A)B\Delta E}{B\Delta E}$	if $cnt = 0$

Fig. 5 Rules of program execution.

4.2 Program Handler

This section explains the core part of the program execution mechanism by showing the details of the program handler to process, namely, to parse and to execute, the avatar program.

The rules of program execution are shown in **Fig. 5**. In this figure, $A\Delta E$ represents the state that the avatar program equals to A and the event-driven program equals to E. Each rule represents the possible change of state from the

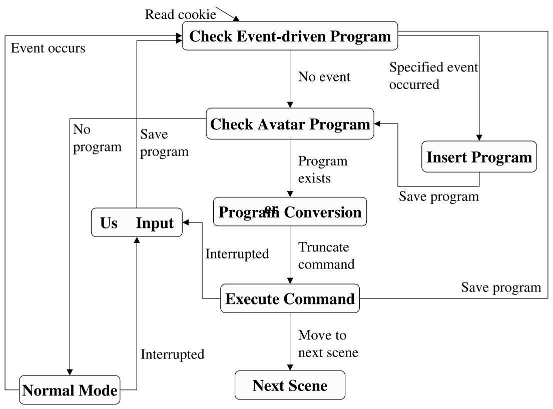


Fig. 6 The procedure of AP program handler.

above to the below when program is executed. We explain the other notations as follows.

- A, B and B_i each refers to an avatar program or null (i.e., an empty string).
- E refers to an event-driven program.
- b represents a basic command except "e", "E" and "X".
- S_i represents the string of integer i.
- a represents an atomic program which may be a single command or an avatar program enclosed in brackets.
- C represents the condition of the "?" command.
- Z_1 and Z_2 each refers to an avatar program enclosed in brackets or null.

These rules are used in executing the avatar program and the event-driven program. In the following explanation, we will refer to the rules in this figure.

As we have mentioned before, there is no special process to handle the avatar program execution. Every scene acts as an HTML file independent of other scenes, and the control is lost when scene changes. Thus, program execution is handled separately from each scene with a JavaScript program.

Figure 6 shows the detail actions of AP program handler. It is done by every single scene on the client side to execute the avatar program. The process is first started when a scene is loaded or reloaded into the web browser. The program handler first reads the cookie to load the event-driven program and the avatar program, and checks whether any events specified in the event-driven program has occurred. If a specified event has occurred, the corresponding program specified in the event-driven program will be inserted into the avatar program as stated in rule "event" in Fig. 5. For example, when the event-driven program "iˆ+(1)" and the avatar program "2f" are currently stored in the system and event "i" (the avatar enters the scene) has occurred, the system inserts "+(1)" into the avatar program to make it become "+(1)2f". The modified program is then saved into the cookie again.

After checking the event-driven program, the avatar program is examined. If no program exists, the scene works as a normal web page with IBNR virtual space functions where the avatar can be moved around freely by the users. When a program exists, the system retrieves the saved program and starts processing the program.

Processing starts with program conversion where the first command of the avatar program is extracted and analyzed. If it is a basic command except command "e", "E" and "X", the avatar program is executed as stated in rule "basic" in Fig. 5. If the basic command is "e" (or "E"), the avatar program is executed as stated in rule "replace" (or "append") in Fig. 5 where the event-driven program is modified. If the basic command is "X", the avatar program is executed as stated in rule "program" in Fig. 5.

If the first command of the avatar program is not a basic command, program conversion is done to convert this command. The first command of the avatar program (which changes every time the conversion is done) is repeatedly analyzed and converted until the first command becomes a basic command (as shown in the rest of the rules in Fig. 5). The details of program conversion are discussed in the next subsection. After program conversion, the program handler extracts the first command from the avatar program to execute. This command is then truncated from the avatar program and the new program is saved into the cookie. As the command is being executed, some events specified in the event-driven program may happen. Thus the system goes back to check the event-driven program and the same process repeats. These four steps, check event, check program, convert program and execute program are repeated until the execution is interrupted, or no more commands exist, i.e. the program is finished executed, or the scene changes.

The program is interrupted when a user interrupts or the next command is an error. In both cases, the program execution is paused to allow the user to modify the current avatar program or event-driven program. When the user is done with program modification, the system continues from the "check event" process. In the case where the program is finished executed, the scene returns to the normal mode, i.e., it works as a normal web page with IBNR virtual space functions. In the case where the scene changes, i.e., the avatar moves to another scene, the program execution control is lost from the old scene and passed to the new scene. In the new scene, the whole thing repeats again.

In the normal mode, users can always key in a program (avatar program or event-driven program). When this happens, the system starts the "check event" process. As we had explained in the previous chapter, an avatar program is a one-time program where the commands are executed once and then deleted, but an event-driven program remains in memory unless being explicitly deleted. Therefore, in normal mode, whenever an event occurs (an event can happen when the user moves the avatar), "check event" process is started. Similarly, as explained above, when an event specified in the event-driven program has occurred, the corresponding program specified in the event-driven program will be inserted into the avatar program and the processing will be carried out.

4.2.1 Program Conversion

Program conversion is done only when the first command is a control command (repeat command or choice and loop commands). This preprocessing is done to filter out the control commands so that in the next process, command execution, only simple and straightforward basic commands need to be handled. In program conversion, a command is either expanded or simplified into zero or more simpler commands. Program conversion is done recursively until the first command in the avatar program becomes a basic command. Then the first command is executed.

If the command is a repeat command, the parameter is duplicated to the front, and the number of repeats is decremented (rules "repeat", "repeat1" and "indefinite" in Fig. 5).

If the command is a choice command, the condition specified in the "if" portion is evaluated and only either the "then" or "else" portion will remain after conversion (rules "if-true", "if-false", "reg-pos", "reg-npos" in Fig. 5). This result may be null as either the "then" or "else" portion can be ommited.

As for a loop command, the system checks the value of *cnt*. If *cnt* is not zero, the parameter is duplicated to the front as in the repeat commands, otherwise the command is converted into null (rules "loop" and "loop-end" in Fig. 5).

4.2.2 Command Execution

Command execution basically performs the basic action specified by the first command in the avatar program. However, when the first action is one of the movement "f", "b", "r" or "l", it needs to be converted to commands of absolute direction "F", "B", "R", "L" which is executed without depending on the direction the avatar is facing. This step is necessary because during program execution, the avatar's direction may change and if a user interrupts the program and then continues execution, the commands "f", "b", "r", "l" will indicate different directions from the initial ones. Converting the movement commands to the commands of absolute direction ensures that the program is executed correctly after user interruption.

4.3 Program Propagation

In the existing IBNR, an avatar's position in the scene is controlled by a JavaScript program. When the avatar reaches the edge of a scene (which is an HTML file), the current scene automatically changes to another scene which is predefined during the virtual space construction stage, and the control is passed to the new scene. As a result, all information in the old scene is lost.

In AP, the program and data must be propagated to the new scene. To achieve this, the program must be stored before the old scene changes to the new scene so that it can be accessed from the new scene. By utilizing the cookie function of a web browser, the programs and the values of variables are saved into a cookie. Since cookies can be accessed by different HTML files, the information stored can be shared by all scenes. In such a way, even after scene change, the programs can be retrieved back and thus can continue to be executed.

4.4 Implementation

In order to use AP in an IBNR virtual space, it is necessary to modify the HTML file of an IBNR subspace. The AP program handler that we have built with a JavaScript program, "ap01.js", must be included into the HTML file, in addition to the "ibbnr08.js" and "avatar.js" used for constructing the IBNR virtual space. We show a sample HTML file in **Fig. 7**.

We have mentioned that the functions "saveProgram" and "saveEvent" can be used in the

```
<html>
<head>
<script language="JavaScript1.2">
var Background="../part1/dsc00157.jpg";
var FloorWidth=410, FloorDepth=400;
... (variable declaration of IBNR virtual
  space)
</script>
<script language="JavaScript1.2"
  src="avatar.js">
</script>
<script language="JavaScript1.2"
  src="ibnr08.js">
</script>
<script language="JavaScript1.2"
  src="ap01.js">
</script>
</head>
<body>...</body>
</html>
```

Fig. 7 A sample HTML file with AP function.

HTML files to save the avatar program and event-driven program into the system. The following shows an example of lines that should be included in the file.

```
<script language="JavaScript1.2">
function start() {
 saveProgram("?f(f):(r)");
 saveEvent("i^G(scnname)A(#scn)S(scnname)");
}
</script>
```

Using the above, a virtual space constructor can predefine the avatar program and the event-driven program in the subspace and execute them when a user comes in to the subspace or when he performs some action like pressing a button.

5. Discussion

5.1 Sample Usages

In this section, we discuss how the avatar programming language can be applied in various virtual spaces.

Information searching and information collecting in a virtual space can be realized. For example, finding a book in a virtual library, finding the shop which sells the CD with the lowest price in a virtual mall by comparing the prices in different shops, or collecting the titles of the CDs produced in year 2000. The avatar program can be used to specify the condition and the action to be taken when the condition is/is not fulfilled using various commands that we have prepared. Nevertheless, it is necessary that the virtual space constructor defines the information about the virtual space inside the subspaces and the user must know how the information is defined to retrieve them. Assume that in a virtual shop formed by a number of subspaces, each subspace contains information about a product which can be specified in variables such as “type”, “price” and “brand”. These information definition can be made known to the user who comes to this virtual space.

In such a way, using avatar programming, users can instruct the avatar to search information. This can save them time and effort to do the searching on their own at real-time, especially when the work is monotonous.

As for movement tracking, users can record down the movement of the avatar when it moves in a forward or backward manner. By doing so, the movement recorded down can be reused when the user comes to the same virtual space the next time. Moreover, by tracking the movement in a backward manner, the avatar can be instructed to move back to the starting point at any time, thus preventing the user from getting lost in a virtual space.

Since the avatar program can be predefined by the virtual space constructors as we have mentioned in the previous chapter, it can be used by the virtual space constructors to provide guidance in a virtual space to the users. By defining the movement from the current scene to the destination scene in a program, the avatar can be instructed to guide the users to a place. Indirectly, it acts as a tour guide in the virtual space.

The virtual space constructors can also make use of the avatar program to provide some useful information for the users. For example, giving the promotion information or product information in a virtual shopping mall, or explanation about an exhibit in a virtual museum or art gallery. By predefining the program, the information about sales, special discounts or new arrivals can be displayed to the user with a message box when the users come to the subspace. Some explanation about product or exhibits can be done in the same way. By utilizing the information stored in cookie, it is possible to display different information for different users with different interest, or only display the information when some criteria is satisfied.

5.2 Observation

Here, we would like to discuss some observation in this research work.

In this work, since we have chosen the internal control approach, independency of subspaces can be maintained. Accordingly, it is possible to have subspaces with different program handlers to handle the program propagating from a subspace to another. Even though the subspaces may contain different program handlers, a program can still be propagated across these subspaces being executed by different program handlers in these subspaces.

In a broader sense, even with other kinds of virtual space (e.g., those constructed by VRML[12]), even though the data format and definition of the virtual space may be different, the program execution mechanism works exactly in the same way. As long as the program handlers for these virtual spaces are implemented, AP can be used in the same manner. Furthermore, it is possible to use AP across different virtual spaces where the program propagates from a virtual space to another kind of virtual space, being executed by the program handlers in these virtual spaces. This makes the integration easier.

5.3 Related Work

Our work has some similar characteristics with mobile agents[1),8)] in that it uses an agent, the avatar, to provide some service for the user and that it aims to achieve autonomous behavior in the agent. However, our idea is to use the avatar as an agent in a virtual space even though currently we are only at the very first step of achieving this.

In Ref. 10), a WAVE language to program mobile processes in distributed environments is proposed while in Ref. 9), a language for specifying dynamically evolving networks of distributed processes is proposed. In these works, the program hops from node to node in a network, which is analogous to the avatar program which hops from a subspace to another subspace. These works also use the same approach with our work where the language is defined with short commands such that short programs are possible. However, the purpose of these languages is to handle mobile processes in distributed environment which differs from AP.

The language structure in AP has some similarity with the Logo programming language[4)] in the part where the movement is repeated – “2f” in avatar programming and “forward 20” in Logo programming. Apart from that, in some computer games especially MUD (multiuser dungeon, a type of multi-player interactive game) where the player, represented by an avatar, explores the virtual space in the game, the movement of the avatar can be predefined. However, the avatar programming is a one-alphabet program, it is executed across different subspaces (thus an inter-space programming language) and it runs on the web.

6. Conclusions

In this paper, we have proposed AP, an avatar programming language for controlling the avatar in a virtual space which can be used on the WWW. We have shown the language structure, explained our implementation and shown some sample usages on how the concept can be used in a virtual space.

We have designed the avatar programming language handler with the internal control approach. We mainly focused on the technical issues like the mechanism to propagate and execute a program across different independent subspaces.

One problem in this language is that it is rather low level as all commands are one-alphabet commands and thus it is not easy for users to remember and use the commands. A solution is to provide a more user-friendly tool to enable the users to make full use of this language. This can be done by providing some predefined macros to perform some jobs, in which a user only needs to choose from the list of macros and provide some information to carry out the job. During execution, the program handler we have proposed can still be used to process the program. Another possibility is to provide a high-level language which is easier to be used, working as a layer in between the user and the program handler. This remains as part of our future work.

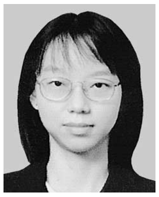
Acknowledgments This research was supported in part by Special Coordination Funds for promoting Science and Technology of the Ministry of Education, Culture, Sports, Science and Technology of Japan, and by Grant-in-Aid for Scientific Research on Priority Areas numbered 13224059 from the Ministry of Education, Culture, Sports, Science and Technology of Japan.

References

1) Harrison, C.G., Chess, D.M. and Kershenbaum, A.: Mobile Agents: Are They a Good Idea? *Mobile Object Systems: Towards the Programmable Internet*, Vitedk, J. and Tschudin,

C. (Eds.), Lecture Notes in Computer Science 1222, pp.25–45, Springer-Verlag (Mar. 1997).
2) Hirose, M.: Image-Based Virtual World Generation, *IEEE Multimedia*, pp.27–33 (Jan.–Mar. 1997).
3) IBNR Project, Nishio Laboratory, Osaka University, http://www-nishio.ise.eng.osaka-u.ac.jp/IBNR/.
4) Logo Foundation — The Turtle, http://el.www.media. mit.edu/groups/logo-foundation/Logo/Turtle.html.
5) Moezzi, S., Tai, L.C. and Gerard, P.: Virtual View Generation for 3D Digital Video, *IEEE Multimedia*, pp.18–26 (Jan.–Mar. 1997).
6) Nakanishi, H., Yoshida, C., Nishimura, T. and Ishida, T.: Free-Walk: Supporting Casual Meetings in a Network, *Proc. ACM 1996 Conference on Computer Supported Cooperative Work*, pp.308–314 (1996).
7) Ogawa, T. and Tsukamoto, M.: Tools for Constructing Pseudo-3D Space on the WWW Using Images, *New Generation Computing*, Vol.18, pp.391–407 (2000).
8) Pham, V.A. and Karmouch, A.: Mobile Software Agents: An Overview, *IEEE Communications*, Vol.36, No.7, pp.26–37 (July 1998).
9) Riely, J. and Hennessy, M.: A Typed Language for Distributed Mobile Processes (Extended Abstract), *Proc. 25th ACM-SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (*POPL 98*), pp.378–390 (1998).
10) Sapaty, P.: *Mobile Processing in Distributed and Open Environments*, John Wiley & Sons, Inc. (1998).
11) Tsukamoto, M.: Image-based Pseudo-3D Visualization of Real Space on WWW, *Digital Cities: Technologies, Experiences, and Future Perspectives*, Ishida, T. and Isbister, K. (Eds.), Lecture Notes in Computer Science, Vol.1765, pp.288–302, Springer-Verlag (2000).
12) VRML, http://www.web3d.org.

(Received April 12, 2001)
(Accepted May 15, 2002)



Yin-Huei Loh received her Bachelor of Computer Sciences in University of Science, Malaysia in 1997. She joined Osaka University in the following year and obtained her M.E. degree in Information Systems Engineering in 2001. She is currently pursuing her Ph.D. degree in the same university.

Takefumi Ogawa received his B.E. and M.E. degrees in Information Systems Engineering from Osaka University, Japan, in 1997 and 1999, respectively. Currently, he is a research associate in the Cybermedia Center, Osaka University. His research interests include groupware systems and augmented reality. He is a member of four learned societies, including IEEE.

Masahiko Tsukamoto received his B.E., M.E., and Dr.E. degrees from Kyoto University, Japan, in 1987, 1989 and 1994, respectively. From 1989 to February 1995, he was a research engineer of Sharp Corporation. From March 1995 to September 1996, he was an assistant professor in the Department of Information Systems Engineering of Osaka University, and since October 1996 to March 2002, he was an associate professor in the same department. Since April 2002, he has been an associate professor in the Department of Multimedia Engineering of Osaka University. His current research interests include mobile computing and augmented reality. He is a member of eight learned societies, including ACM and IEEE.

Shojiro Nishio received his B.E., M.E., and Dr.E. degrees from Kyoto University, Japan, in 1975, 1977 and 1980, respectively. From 1980 to 1988 he was with the Department of Applied Mathematics and Physics of Kyoto University. In October 1988, he joined the faculty of the Department of Information and Computer Sciences, Osaka University, Japan. In August 1992, he became a full professor in the Department of Information Systems Engineering of Osaka University. He has been serving as the director of Cybermedia Center of Osaka University since April 2000. Since April 2002, he has been a full professor in the Department of Multimedia Engineering of Osaka University. His current research interests include database systems, multimedia systems and distributed computing systems. Dr. Nishio has served on the Editorial Board of *IEEE Transaction on Knowledge and Data Engineering*, and is currently involved in the editorial board of *Data and Knowledge Engineering*, *New Generation Computing*, *International Journal of Information Technology*, *Data Mining and Knowledge Discovery*, *The VLDB Journal*, and *ACM Transactions on Internet Technology*. He is a fellow of IPSJ, and he is a member of eight learned societies, including ACM and IEEE.