Regular Paper

Content-independent EPSS with Automatic Context Sensing on the Web

Yuko Ikehata,† Toshio Souya†† and Yoshinori Hijikata†††

An EPSS is an Electronic Performance Support System that provides users with integrated, on-demand access to information for the purpose of job performance enhancement. We propose a framework, called WebAttendant, upon which a "content-independent EPSS" can be developed. WebAttendant automatically tracks and analyzes a user's operational behavior ("automatic context sensing") on the Web, and provides individual instructions in the form of helpful interventions, according to a set of rules, such as guidance windows and balloon help messages. As a result, WebAttendant can be managed independently from the Web sites, and the EPSS can be reused or added without changing the Web content itself. In order to verify the usefulness and efficacy of WebAttendant, we carried out several experiments. The experiments show that WebAttendant is effective as a platform for a content-independent EPSS on the Web.

1. Introduction

Use of Internet for things like online shopping at "amazon.com[1)]", online reservations or online selling of items at "Ebay[2)]", has drastically increased in recent years and continues to grow everyday. People carrying out these online activities need to complete some tasks on the websites. For example, they need to login or register on the websites, search the web sites, click on right links and submit or complete their task by clicking on the appropriate buttons. Web sites of these types are considered "task-oriented Web sites".

We believe that there are two problems with existing "task-oriented Web sites":

- If the Web content is not organized well structurally, the website users may not easily understand how to use the website,
- If the users are not skilled enough to use Web, they will have significant difficulty using the task-oriented Web sites.

If users face any of the above problems they may repeatedly scroll up and down in order to find their way to what they are planning to accomplish on a Web page. If they think they accessed the wrong page, unrelated to their task, they may go back to the previous page trying to get to the link that may lead them to the target page. Unnecessary scrolling up and down and moving into and out of task-unrelated pages may result in a longer time for a given task completion.

The above problems exist mainly because of the gap between the knowledge level of the users and the knowledge level a Web site requires users to have in order to carry out tasks on that specific Web site. For example, if a large number of first-time users, more than anticipated, visit the Web site at the same time, they may find the Web site too difficult to use. It is important to bridge the gaps between the knowledge level that a Web site requires of its users and the actual knowledge levels of the users who visit the site.

One solution to the problems that task-oriented Web sites are facing is attempting to add an Electronic Performance Support System (EPSS[3)]) to the Web site.

Web-based EPSS is designed using methods to modify the composition of Web content. For example, by embedding help functions in the content and adding detailed explanations for the content on the Web page.

Almost every existing Web-based EPSS is a "built-in EPSS", i.e., they have been developed for specific Web content and the service is part of the content. Here are some problems with built-in EPSS:

- It is difficult to reuse a specific website's built-in EPSS for another Web content because the composition of any specific Web content is different from another Web content, and the skills of different Web site users are also different.

† IBM Tokyo Research Laboratory
†† IBM e-business Wireless Consulting
††† Department of Systems and Human Sciences, Graduate School of Engineering Sciences, Osaka University

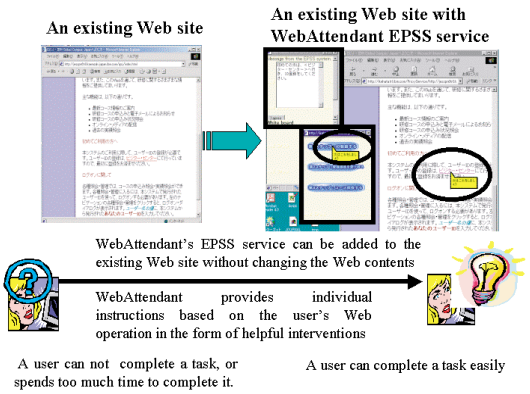


Fig. 1 Content-independent EPSS by WebAttendant.

- Various versions of a built-in EPSS are required for a specific Web site to support users with various skill levels. Development and maintenance of various versions of a built-in EPSS is costly and time consuming.

To resolve the shortcomings of the built-in EPSS, we propose the framework of a content-independent EPSS on the Web called WebAttendant. As shown in **Fig. 1**, WebAttendant is built independent of the Web content. WebAttendant tracks user's DOM (Document Object Model) event on the Web page, automatically records and analyzes a user's operation logs ("automatic context sensing"), and provides users with specific instructions. Instructions are in the form of helpful interventions by guidance window(s) beside the Web page and balloon help message pointing to the target object on a Web page. Processes such as DOM event tracking, users' operation logs recording and analysis, and execution of the rules can be managed individually and independently of the Web sites. The features of WebAttendant are as follows:

- WebAttendant's EPSS can be reused for Web sites with different contents just by changing the rules. This will reduce the cost of development and the maintenance of the whole system
- EPSS developers can easily provide EPSS on the Web content by creating rules using WebAttendant's authoring function.
- Since each process of WebAttendant is independent of the Web server, EPSS creator can add EPSS to the existing Web content without changing the Web content itself.

In order to verify the usefulness and efficacy of WebAttendant, we carried out several experiments. Results of the experiments show that WebAttendant is a highly effective platform for Web-based EPSS.

2. The Concept of EPSS

Here we summarize the purpose of EPSS described by Stevens, et al.[3]. An EPSS is a system that can provide on-demand, task-specific skills training, task- and situation-specific information access, customized tools for task automation, and embedded coaching, help, and validation tools. The aim of EPSS is to address the various levels of skill among the users of applications, in other words, to develop an EPSS that does not require its users to be highly skilled in Web site operation.

In the Web-based EPSS framework, there are two types of EPSS framework: Built-in EPSS framework and content-independent EPSS framework for a Web site.

- Built-in EPSS framework for a Web site: Web-based EPSS framework, which can provide an EPSS dependent on the Web content. To provide an EPSS to a Web site, the developer needs to modify the modules for each required EPSS functions using programming languages, and need to embed these modules for each HTML Web page.
- Content-independent EPSS framework for a Web site: Web-based EPSS framework, which can provide an EPSS independent of the Web content. The developer can add any HTML Web page without changing the Web page itself. This framework already has most of the functions for an EPSS service. The developer can use these EPSS functions to provide any HTML Web page by only creating some new rules.

3. Comparison of Existing EPSS Software and WebAttendant

Figure 2 shows WebAttendant and different categories of existing EPSS. Existing EPSS and WebAttendant are categorized based on their target applications. There are EPSSs which are used for stand-alone applications and for Web sites. We first explain EPSS for stand-alone applications.

3.1 EPSS for Stand-alone Applications

EPSS for stand-alone applications are categorized based on the techniques they use to figure out what their users are trying to do. There are

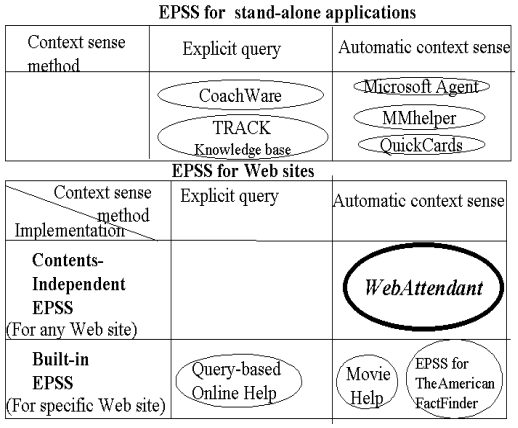


Fig. 2 WebAttendant and existing EPSSs.

two categories of EPSS for stand-alone applications:

- EPSS using explicit query approach.
- EPSS using automatic context sensing approach.

For the explicit query approach, users have to request support from the EPSS by sending explicit queries about what they are trying to do or about what help they want. CoachWare[4] and the TRACK Knowledge Base are examples of systems that use explicit queries to find out how to support a user's task. Such systems may use a database of queries that are intended to force users to clarify their situation even when users are not knowledgeable about the application they are trying to use. Alternately, users may be asked to describe their problems in their own words. However, this approach is especially difficult for those users who don't understand the application. Moreover, because some users may be able to explicitly describe their problems, they may need hints, such as marginal annotations, before they can even understand the questions.

For the automatic context sensing approach, the system attempts to infer each user's intentions by automatically tracking their operation histories. If a user makes a mistake, the system can detect his/her mistake and support him/her without any request for assistance being made by the user, even when users do not recognize that they have made mistakes. For this reason, an EPSS with an automatic context sensing approach is more user-friendly and superior to an EPSS with an explicit query approach.

MMHelper[5], Microsoft Agent[6], and QuickCard[7] are examples of EPSS with the same approach. They attempt to automatically sense the context. Examples of useful data that can be used for this detection include the font size and number words input in a form and the amount of time a user spends in an application without doing anything. These existing EPSS detect the operations within the standard GUI components. Since these existing EPSS are designed for a stand-alone application, they cannot be reused for other application.

3.2 EPSS for Web Sites

Categorized from the viewpoint of system design, there are built-in EPSS and content-independent EPSS on the Web. A built-in EPSS is designed for specific Web content by building the EPSS as a part of the content. On the other hand, a content-independent EPSS is designed independently from any Web contents.

Almost all existing Web-based EPSS are built-in EPSS. Since webmasters can design many kinds of interventions and make programs freely, this approach allows a greater flexibility in design. However, there are some disadvantages with built-in EPSS:

- Since the composition of the Web content and user's skill levels vary for each kind of Web content, it is difficult to reuse the modules of one built-in EPSS for other Web contents.
- If a large number of users with different skill levels use the Web sites, webmasters will need to develop several versions of a built-in EPSS service to support its users. This leads to higher cost.

On the other hand, the content-independent EPSS software has some advantages:

- It is easy to add or reuse EPSS for existing Web content without changing the Web content itself.
- It is easy to reuse the separate modules of an EPSS for other Web sites since they are independent of the Web content. And EPSS master can manage each process of EPSS independent of the Web site.

Most existing EPSS on the Web are built-in EPSS with explicit query approach, and these can be used only for specific Web content. The main examples of built-in EPSS with explicit query approach are Query-Based Online Help Services. In a Query-Based Online Help Service, a user has to ask questions by entering keywords or forming a query on the Web page and then get guidance from an EPSS database. A

Query-Based Onlin Help Service does not assist a user unless he/she recognizes the mistakes.

Examples of existing built-in EPSS using automatic context sensing are a Movie Help Service on a Web site and the EPSS for the American FactFinder[8)~9)]. The Movie Help Service demonstrates to the user how to use the Web site using balloon help messages when a user loads the Web site. The movie Help Service only tracks a user's page loading events. It does not grasp a user's intention. The EPSS for the American FactFinder provides performance support based on their unique characteristics and the needs of the key user group to which they belong. However, it mainly tracks the user's page loading events, and it does not track more detailed information, as does WebAttendant. Plus, it is not reusable.

WebAttendant grasps a user's status for offering instructions by automatically tracking the user's more detailed Web operations, such as, the time and the type of the Document Object Model (DOM) event (see Section 5(2)) occurs, the type, number, and value of any target objects where the DOM event occurs, the user name, and the current URL involved. WebAttendant can infer a user's intention more accurately and can provide the user with more individualized instructions by using helpful interventions. Furthermore, since WebAttendant is designed as a content-independent EPSS. These characteristics of WebAttendant makes the development and testing processes much easier, and the development costs lower adding or reusing services without changing the existing Web content itself. Due to the above features, WebAttendant's EPSS can be broadened to any existing Web content. Considering the above-mentioned advantages of WebAttendant, it seems reasonable to consider it is superior to other EPSS.

4. Design of the System

Our objectives for designing WebAttendant are to meet two requirements:

(1) To create a framework for a cost-efficiency development of an EPSS.

(2) To develop an EPSS that does not require its users to be necessarily highly skilled in Web site operation.

To achieve our first objective, we designed the WebAttendant to be a separate EPSS from the existing Web content (Section 4.1). Also, WebAttendant separates the rule module from the WebAttendant execution module to reuse the same rules for other Web content. Furthermore, WebAttendant provides authoring tool functions so that EPSS developers (rule creators) can create rules easily, even if they do not have programming skills (Section 4.2).

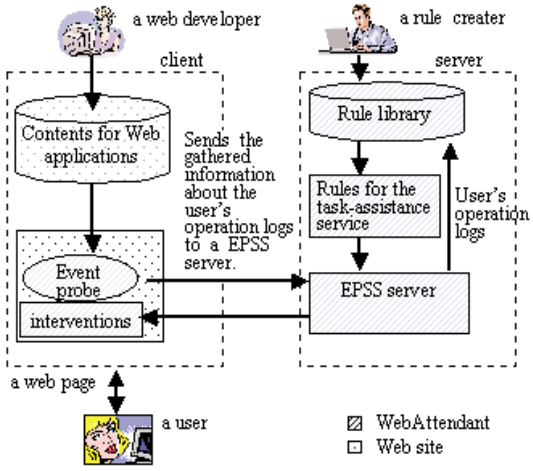


Fig. 3 The structure of WebAttendant and the existing Web site in development.

To achieve our second objective, we designed WebAttendant to provide individual instructions based on the user's Web operation events with an automatic context sensing approach (Section 4.3).

4.1 Separation of the Web Site and WebAttendant

Figure 3 shows the structure of both the existing Web site and WebAttendant in development. WebAttendant is designed to operate separately from a Web site. On the client side, the proxy server embeds several WebAttendant modules to track a user's Web operation and to provide him/her with interventions on the Web page. On the WebAttendant server side, other processes of WebAttendant, such as recording and analysis of user's operation and execution of the rules, operate separately from the Web site. As a result, WebAttendant services can be added, changed, and reused without a Webmaster changing the existing Web content. Also, Web sites and WebAttendant can be maintained individually.

4.2 Separation of the Rules for EPSS from the WebAttendant Execution Module and Authoring Tool

In WebAttendant, the rule module are separated from the WebAttendant execution module, allowing other Web contents to reuse the

same rules for an EPSS and decreasing the number of rules that have to be created. WebAttendant provides authoring tool function to create rules easily.

4.3 Providing Each User with Individual Instructions on a Web Page

This subsection describes the functions required to provide each user individual instructions with automatic context sensing approach. WebAttendant tracks and analyzes a user's operation history, and determines to provide individual instructions according to a set of rules. Then WebAttendant customizes a Web page to provide helpful interventions at the necessary place.

4.3.1 Automatic Tracking of the User's Operation

To provide each user with an individual instruction on a Web site, a method to recognize the user's status or intention by inference from the user's detailed operation history is required. Therefore, WebAttendant is required to provide a history tracking function, which automatically detects the following user's operations, as many as possible on the Web site.

(1) Sensing the context by tracking a user's operations on any HTML Web pages that he/she visits.

Generally, a Web site consists of many pages. If the tasks in the Web site are complicated, users will need to carry out various operations over many pages. Therefore, a function, which tracks the users operations for several pages is required.

(2) Sensing a context by tracking users operations at each object on the same page.

If the users' tasks in the Web site are complicated, they will need to carry out various operations on the same page. WebAttendant detects users operations at each object (for example, each inputting form) on the same page in order to examine if he/she carries out the operation in a correct way.

4.3.2 Analysis of User's Web Operation and Determination of the Rules Execution

WebAttendant is capable of analyzing a user's operation behavior and activity. To analyze a user's operation, WebAttendant records and manages the user's operation as several contexts with names and attributes on the server side. Then, WebAttendant analyzes each user's contexts by examining some attributes to find out whether or not it has satisfied certain conditions. For example, it will examine to find out if the number of scrolling operations is less or more than 5. Then, it will examine the rules that need to be executed.

4.3.3 Customization of the Web Page for Individual Instructions

After WebAttendant determines to execute a rule to provide a user with individual instructions, it customizes the Web page dynamically by displaying several interventions at suitable places, or it modifies the Web page automatically in order to complete a task efficiently.

5. The Composition of WebAttendant

Figure 4 shows the structure of WebAttendant. It consists of following modules:

(1) Proxy server module, which embeds the tags for event probe module and guidance module into each Web page.

The tags refer to modules on the Script Server (a standard HTTP server). The Web page where the tags are embedded is then sent to a client browser[10].

(2) Event probe module, which tracks user's operations in detail.

When a user operates on the Web page, the event probe module automatically tracks the user's operation by handling a DOM event in the Web browser. Examples of a DOM events are CLICK, MOUSEMOVE, MOUSEOVER, SCROLL, KEYDOWN, KEYUP, LOAD, SELECT, and so on.

The event probe module collects a complete set of information, including

- The time and the type of the DOM event that occurs,
- The type, number, and value of any target objects where the DOM event occurs,
- The IP address of the client or the user name,
- The current URL involved.

The collected information about the users operations is an "operation event". The event probe module sends operation event to a log server[11]. Examples of different types of target objects are TEXT, RADIOBUTTON, CHECKBOX, and so on.

(3) Log server module, which receives the operation events from the event probe module and sends them to the context module.

Log server module also transmits DOM actions from a rule module to the guidance module.

(4) Context module, which has general contexts and represents what a user has done.

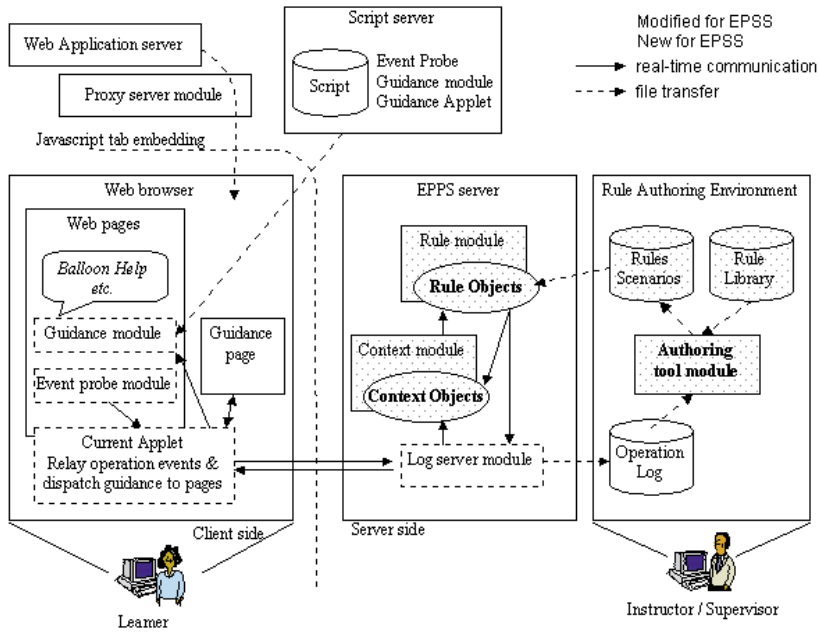


Fig. 4 The structure of WebAttendant.

The context consists of a set of more than one context items with a name and attribute. There are two kinds of context items:

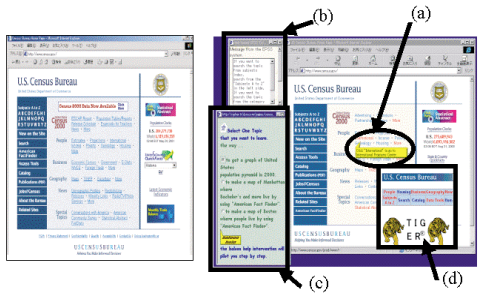
- A parameter from the history of a user's operation event: such as "user's name", "URL user loads", "DOM event name that a user operates", "the time that DOM event occurs", "the target object ID that user focuses on a Web page", and so on
- A parameter to analyze operation event: such as, "a standard time parameter to consider that a user stops inputting a specific form for long", "a standard number parameter to consider that a user focuses on a specific object too much", and so on.

When the context module accepts the operation event from the log server it records and updates the context. After this, the context module makes a new context item, which is appropriate to the DOM event, or updates the value of a context item. Also, the context module examines a value of a context item and if the value is satisfactory, according to the set standard parameter, it makes the context event to notify the rule module that the user's context is updated and sends the context event to the rule module. There are three kinds of context events:

- DOM event itself,
- Context events that notifies the rule module that user's context updated by analyzing the history of user's operation: such as context events that notifies that a user spends inputting a form too much time, that a user moves a mouse without inputting anything in a form, and so on.
- Context events that notifies the rule module that user's context updated by analyzing the value of input form: such as, context event that notifies the context item's changes when a user inputs a specific key word, when a user inputs a number larger than a specific number, and so on.

(5) Rule module, which decides which rule should be selected to provide individual instructions based on the user's context / DOM events. The rule module receives the context/DOM event from the context module. Then, the rule module selects a rule that is appropriate to the context/DOM event and decides the execution of a guidance to the user as a DOM action or an update to the value of a context item as a context action. The rule module sends a DOM action to the log server, which is a command for guidance module to execute guidance on a DOM object. Or, the rule module sends the context action to the context module, which is a command for the context module to update the value of a context item.

(6) Guidance module, which runs as a client-side applet and give an individual instruction to a user.



(1) An existing Web site without WebAttendant's EPSS service

(2)An existing Web site with WebAttendant's EPSS service

Fig. 5 WebAttendant experimental service to US Census Bureau Web site (http://www.census.gov/). An example of several interventions by DOM actions: (a) Balloon help on: adding a babble balloon help beside a specific link with a message, (b) Auto input: inputting a message automatically in a form, (c) Window on: displaying a window with a specific URL, (d) Web page on: displaying a specific URL Web page beside a link in a part of Web page.

When the guidance module receives a DOM action from the log server, it executes the DOM action, such as an action to provide interventions to the target object[12] or change pages automatically. **Figure 5** shows an example of WebAttendant providing an experiment EPSS to an existing Web site. The examples of DOM action are following:

- Balloon help on: adding a balloon help beside a HTML element with a message (Fig. 5 (a)).
- Auto input: inputting a message automatically in an input form (Fig. 5 (b)).
- Window on: displaying a window with a specific URL (Fig. 5 (c)).
- Web page on: displaying a specific URL Web page besides a HTML object in a Web page (Fig. 5 (d)).

(7) Rule, which defines the functions of EPSS service, i.e., the way to instruct each user depending on the user's situation.
Rules describe what kind of intervention should happen to the user and how that should happen, depending on the user's situation. Rules are described as a XML rule format. A rule format consists of a "condition part" and an "execution part".
In condition part, the user's situation is described by using context event and the attribute of a specific context item. In execution part, the way to instruct each user or the update of his/her contexts are described by using more than one DOM action and context action.
(8) Authoring tool module, which provides a simple point-and-click user interface to create rule descriptions.
Authoring tool module allows users to create rules by minimum operations, linking them directly to the target object on the Web page. The rule creator can describe which URL and object, when, what kinds of messages, and how he/she want to show by the interventions on the Web site by selecting an item in the window beside the target Web page.

6. Experiments to Verify the Superiority of WebAttendant over Built-in EPSS

As mentioned in Section 4, our first objective for developing WebAttendant was to create a framework for a cost-efficient development of an EPSS. Our second objective was to develop an EPSS that does not require its users to be necessarily highly skilled on the Web. In other words, we wanted to reduce the gap between the knowledge level of the users and the knowledge level a Web site requires its users to have in order to carry out tasks on that specific Web site. We conducted two types of experiments to examine whether or not we achieved our objectives. We describe one of the experiments in this section and the other experiment in Section 7. In order to verify the superiority of WebAttendant framework over a built-in EPSS its cost-efficiency of development, we ran several sets of experiments to evaluate two aspects of EPSS development:
(1) An amount of work involved to develop an EPSS.
First we examined the total operational steps involved in developing an EPSS when we used WebAttendant. Then, we examined the total size of the programs involved to develop the same EPSS when we used a built-in EPSS framework. Finally, We compared the total operational steps involved in EPSS development to the total size of program involved in EPSS development.
(2) Reusability To examine a cost-efficiency of development of an EPSS when EPSS modules are reused, we compared the total number of operational steps involved when we used WebAttendant framework to the size of a program involved when we used a built-in EPSS framework to reuse the same EPSS modules. We did

Table 1 The scenarios of EPSS services for each Web site and for each skill level user, and examples of a rule.

case	scenario
EPSS -A	Assisting users how to register using their ID, check the availability of the courses, and fill out course application
EPSS -B	Assisting users how to make a map using data in the US Census Bureau
EPSS -C	Assisting users how to transfer a money to other's bank account
EPSS -L1	The target is a user who does not have a Web operation skill, and also a first time user of this site. Assisting a user by explaining the flow of a task for his purpose in detail. And also explaining the summary of each page.
EPSS -L2	The target is a user who have an ordinary Web operation skill. Or a user who used this site before. Assisting a user by explaining the summary of each task simply.
EPSS -L3	The target is a user who is a Web expert. Or a user who used this site many times. Assisting a user only he makes a mistake, for example, when inputting a world in a target object with mistake.

this comparison for two possibilities:

(a) `Developing EPSSs for Different Web sites:`
We developed several different EPSSs for different Web sites. We then estimated the amount of work when modules for EPSS functions to provide EPSS to some specific Web sites were reused for other Web sites as well.

(b) `Developing EPSSs for users with different skill levels on the same Web site:`
Assume an EPSS module is used to provide EPSS service to "highly skilled users" on a specific Web site called site A. We call this EPSS-L1. Similarly, we call EPSS service provided to "intermediate-skilled users" and "low-skilled users" in site A as EPSS-L2 and EPSS-L3 respectively. We estimated the amount of work when these different EPSSs modules for EPSS functions were reused for other users with different skill levels even though we initially designated it to be EPSS for highly skilled users.

6.1 The Method and the Results of EPSS about an Amount of Work Experiments:

To compare an amount of work using WebAttendant framework versus using a Built-in EPSS framework, we developed EPSS-A, which is an EPSS for a Web-training site. As shown

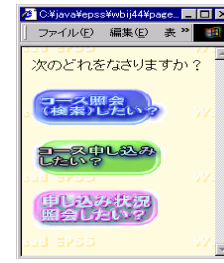


Fig. 6 A wizard asking a user the purpose of his/her task.

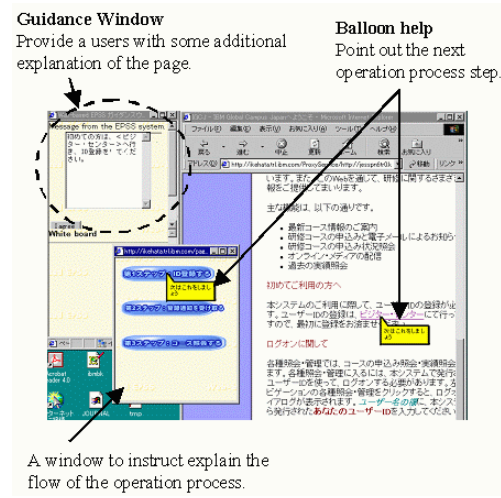


Fig. 7 WebAttendant providing task-assistance to a training business Web site.

in **Table 1** (EPSS-A), the purpose of EPSS-A service was to help users register using their ID, check the availability of courses and fill out course applications. **Figures 6** and **7** shows an example of EPSS-A services. To carry out the required functions, EPSS-A had two types of modules:(1) EVENT module, for tracking and analyzing a user's operational behavior. (2) GUIDANCE module, for providing instructions in the form of helpful interventions. **Table 2** shows the details about these modules. **Table 3** (EPSS-A) shows the EPSS-A's different modules and the number of times these modules are used.

6.1.1 Results of EPSS Development Experiments Using a WebAttendant Framework

As WebAttendant already had all modules shown in Table 2 and could use any HTML Web page without changing each Web page, the only work required to develop EPSS-A was to create rules for defining the functions of EPSS-A. An

Table 2 An example of modules for EPSS functions.

Event module name	
load	• load a page
unload	• unload a page
mouseover	• put the mouse over the target object
mouseout	• put the mouse out of the target object
click	• click the mouse at the target object
submit	• submit the target form
no-change	• lose focus without inputting
repeated-focus	• focus the target object repeatedly
specified-focus-order	• focus some objects with specified-order
specified-focus-unorder	• do not focus some objects with specified-order
long-focus	• foucus an object for a long time
long-no-keypress	• focus an object for a long time without inputting
many-mouse-operations	• move a mouse many times
many-scroll-operations	• scroll many times
many-mouseover	• put a mouse over some objects many times
many-back-button-return	• back the page to previous page many times
click-hesitation	• hesitate a click operation
input-include-prohibited-char	• input characters with prohibited char
Guidance module name	
page change	• change a different URL automatically from the current URL in a same window
window on	• display a window which a specific URL
balloon help on	• add a balloon help beside a HTML element with a message
wizard	• display a window as a wizard for a user
Web page on	• display a specific URL Web page besides a HTML object in a Web page
autoinput	• input a message automatically in an input form
alert	• display an alert window

authoring tool function was used for Extensible Markup Language (XML) format rule description. Thus, the EPSS creator did not need to have any programming skill to implement an XML format rules, and he or she created rules by simple operations. For example, to create the rule "when a user puts a mouse over a specific link many times, the window which instructs how to use this Web page is provided"

Table 3 The number of times using a module for each Web site.

Module name	number*1			total*2
	a*3	b*3	c*3	(E)
The total number of rules	43	35	30	-
EVENT				
load	4	2	6	11
unload	1	-	-	0
mouseover	8	3	2	12
mouseout	8	3	2	12
click	2	3	-	4
submit	-	1	-	0
no-change	-	-	1	0
repeated-focus	-	-	3	2
specified-focus-order	-	1	2	2
specified-focus-unorder	-	1	3	3
long-focus	1	-	-	0
long-no-keypress	1	-	1	1
many-mouse-operations	1	10	-	10
many-scroll-operations	3	-	2	4
many-mouseover	8	5	-	12
many-back-button-return	3	1	3	6
click-hesitation	1	5	4	9
input-include-prohibited-char	2	-	1	2
GUIDANCE				
pagechange	3	1	2	5
window on	8	1	-	8
balloon help on	19	14	17	49
wizard	3	10	7	19
Web page on	8	7	-	14
autoinput	-	-	1	0
alert	2	2	3	6

∗1: The number of times using modules.
∗2: The total number of times reusing a module (E).
∗3: a: EPSS-A, b: EPSS-B, c: EPSS-C

the EPSS creator only needed to follow the following 4 steps using an authoring tool function:

`(Step1)` select the condition part of the rule from the drop-down lists,

`(Step2)` click a target object (link) on the target Web page,

`(Step3)` select the type of intervention that you want to provide from a drop-down list, and

`(Step4)` fill in a message that you want to show with an intervention.

We estimated the amount of work that needed to be done to develop EPSS-A using the number of 43 rules for each Web site and total number of operational steps for creating Web sites from **Table 4**. A rule format consists of a "condition part" and an "execution part". Condition part is defined as the function for the EVENT module. Execution part is defined as the function for the GUIDANCE module.

Table 5 shows the total number of steps us-

Table 4 The size of programs in built-in EPSS framework and the number of steps using WebAttendant.

module name	Built-in		Web-attendant	
	(A)	(B)	(C)	(D)
Event				
load	12	1	1	0
unload	32	1	1	0
mouseover	34	1	2	1
mouseout	45	1	2	1
click	109	1	2	1
submit	38	1	2	1
no-change	223	4	2	1
repeated-focus	227	6	2	1
specified-focus-order	277	16	2	1
specified-focus-unorder	277	16	2	1
long-focus	188	4	2	1
long-no-keypress	217	6	2	1
many-mouse-operations	193	6	2	1
many-scroll-operations	170	2	1	0
many-mouseover	159	4	2	1
many-back-button-return	216	4	1	0
click-hesitation	228	4	2	1
input-include-prohibited -char	160	4	2	1
Guidance				
pagechange	18	1	1	1
window on	26	1	1	1
balloon help on	132	1	1	1
wizard	26	1	1	1
Web page on	258	1	1	1
autoinput	15	1	1	1
alert	4	1	1	1

(A): The total number of programming-lines the creator modified at the first time by **Table 9**. (line)
(B): The total number of programming-lines the creator modified when reusing a module. (line)
(C): The number of steps to create rules at the first time.
(D): The number of steps to change a rule when reusing a rule.

ing WebAttendant. As shown in Table 5 the amount of work to develop EPSS-A was 103 operational steps using WebAttendant.

6.1.2 Results of EPSS Development Experiments Using a Built-in EPSS framework

To develop EPSS-A, the creator had to modify several modules for required EPSS functions, using JavaScript and Java languages. Also, the creator had to modify a module to define these EPSS functions and embed it in each HTML Web page. Therefore, the EPSS creator had to have programming skills.

Using the size of the programming by Java and JavaScript languages, we estimated the amount of work that was done for EPSS-A development. Table 5 shows the results of EPSS development experiments. Table 4 shows the

Table 5 The results of EPSS about an amount of work experiments.

module name	Built-in	Web-attendant
Event		
load	15	1
unload	32	1
mouseover	41	10
mouseout	52	10
click	110	4
long-focus	188	3
long-no-keypress	217	3
many-mouse-operations	193	3
many-scroll-operations	174	1
many-mouseover	187	10
many-back-button-return	224	1
click-hesitation	228	3
input-include-prohibited-char	164	4
Guidance		
pagechange	20	4
window on	33	9
balloon help on	204	20
wizard	28	4
Web page on	272	9
alert	8	3
total	2,390	103

Built-in: The total number of programming lines the creator had to modify. =(A)+(B)×(E)
WebAttendant: The total number of operational steps to create rules. =(C)+(D)×(E)
(A),(B),(C),(D) in Table 4, (E) in Table 3

size of the programming lines using built-in EPSS. For example, when modifying a "many-mouse-operations" module to handle a user's operation that of moving a mouse many times, the required size of programs is 193 lines, as shown in Table 4. Even the developer could reuse this module at the second time, he still needed to modify a module to define this module for required EPSS functions and embed several modules for each target object in each HTML Web page. When reusing this module several times, the required size of programs was 6 lines from the second time as shown in Table 4. When modifying a "window-on" module to provide a user with a guidance window with a message, the size of programs were 26 lines as shown in Table 4. When reusing this module, the size of programs was 1 line but needed to embed for each target object in each HTML Web page. As shown in Table 5, a total size of programming for development of EPSS-A became 2,390 lines by summing all the required size of programs for all modules.

6.1.3 Comparison

To develop EPSS-A using WebAttendant, the creator's only work was making rules by using the WebAttendant authoring tool function.

The total number of operational steps involved to make 43 rules was 103, and no special programming skill was required. In contrast, using a built-in EPSS framework, the creator had to modify several EPSS modules using a programming language. Even if the creator could reuse EPSS modules that he/she had modified before, there was still the need to modify a module to define the EVENT and the GUINDANCE modules and embed these modules for each target object in each HTML Web page. From the results of the experiments, we found that the total size of programs the creator had to modify were 2,390 programming lines. Though we could not compare the total number of operational steps using WebAttendant and total size of programs in built-in EPSS framework, we are certain that the cost of EPSS-A development using a built-in EPSS was higher compared to the cost of EPSS-A development using WebAttendant framework.

6.2 The Method and the Results of Reusability Experiment:

6.2.1 Developing EPSSs for Different Web Sites

We developed three different EPSS for three different Web sites, using the WebAttendant framework once and another time using a built-in EPSS framework. The three different EPSS were the EPSS for Web training site (EPSS-A), which we already described in Section 6.1, the EPSS for US Census Bureau Web site (EPSS-B) as shown in Fig. 5 and the EPSS for Web banking site (EPSS-C). We then compared the amount of work needed to complete for development of these EPSSs when we used the WebAttendant framework compared to when we used a built-in EPSS framework. Table 1 shows the scenarios of EPSS services for each Web site. From the scenarios, we determined the functions each EPSS required to perform. Table 3 shows a list of modules for each function of the EPSS-A, EPSS-B and EPSS-C.

(1) Results of EPSS development experiments using a WebAttendant framework.

The only work needed to be completed to make the EPSS modules reusable was to change several parts of each rule that WebAttendant was going to reuse.

We estimated the amount of work involved in modifying the EPSS-A to be reusable as EPSS-B or EPSS-C. We also estimated the amount of work involved in modifying the EPSS-B to be reusable as EPSS-C. The estimation was based on the total number of operational steps involved in modifying the reusable rules. As shown in Table 3 the number of rules created for each Web site were 43 for EPSS-A, 35 for EPSS-B, and 30 for EPSS-C. We consider each part of reusability individually about EVENT module as a "condition part" and GUIDANCE module as an "execution part". Reusable rules that require no modifications:

These rules define the functions of an EVENT module, which are independent of the Web site contents. The following are examples of reusable rules:

- A rule that defines the function of "many-mouse operations (handling a mouse many times)".
- A rule that defines the function of "many-scroll operations (handling user scrolling many times)".

There are some reusable rules that require some changes, these are dependent on the Web site contents. For example, "click-hesitation" module in Table 4, 2 operational steps which define a target DOM ID for an instruction in a rule had to be changed. As show in the **Table 6**, the total number of operational steps involved

Table 6 The results of reusability experiment developing EPSSs for Different Web sites.

module name	Built-in	Web-Attendant
Event		
load	11	0
mouseover	12	0
mouseout	12	12
click	4	4
repeated-focus	12	2
specified-focus-order	32	2
specified-focus-unorder	48	3
long-no-keypress	6	1
many-mouse-operations	60	10
many-scroll-operations	8	0
many-mouseover	48	12
many-back-button-return	24	0
click-hesitation	36	9
input-include-prohibited-char	36	9
Guidance		
pagechange	5	5
window on	8	8
balloon help on	49	49
wizard	19	19
Web page on	14	14
alert	6	6
total	422	158

Built-in: The total number of programming lines the creator had to modify to reuse =(B)×(E)
WebAttendant: The total number of operational steps to change rules when reusing modules =(D)×(E)
(B),(D) in Table 4, (E) in Table 3

Table 7 The number of times using a module for a user with a different skill.

module name	L1	L2	L3	total(F)
The total number of rules	10	20	30	
Event				
load	1	2	4	6
unload		1	1	1
mouseover			3	2
mouseout			3	2
long-focus		1	1	1
long-no-keypress		1	1	1
many-mouse-operations		1	1	1
many-scroll-operations	2	2	2	5
many-mouseover		4	8	11
many-back-button-return	2	3	3	7
click-hesitation	3	3	1	6
input-include-prohibited-char	2	2	2	5
Guidance				
pagechange			2	1
window on	4	5	7	15
balloon help on		4	10	13
wizard		3	3	5
Web page on	4	6	6	15
alert	2	2	2	5

L1: EPSS-L1, L2: EPSS-L2, L3: EPSS-L3
total: the total number of times reusing a module = (F)

in modifying the reusable rules was 158 steps.
(2) Results of EPSS development experiments using a built-in EPSS framework
We developed EPSS-A, EPSS-B and EPSS-C using a built-in EPSS framework. To reuse these EPSSs, the creator had to modify the modules for each function using JavaScript and Java languages, and had to embed them in each module in each HTML Web page. Therefore, the EPSS creator had to have programming skills. We estimated the amount of work involved in EPSS development based on the total number of programming lines the creator had to modify to reuse these EPSS modules. As show in Table 6, the total programming lines for development of EPSS-A, EPSS-B and EPSS-C were 422 lines.

6.2.2 Developing EPSSs for Users with Different Skill Levels in a Same Web Site

We developed three different EPSSs once using the WebAttendant framework and another time using a built-in EPSS framework. The three different EPSSs were intended for users of three different skill levels in Web site operation. We developed the EPSS for "highly skilled users" as EPSS-L1, EPSS-L2 for "intermediate-skilled users" and EPSS-L3 for "users with little skill" in Web operation. We compared the tasks involved in reusing the modules of these EPSSs when we used a WebAttendant framework compared to when we used a built-in EPSS framework.

Table 1 shows the scenarios for three EPSS services for users with three different skill levels. From the scenarios, we determined the functions each EPSS was required to perform. **Table 7** shows a list of modules for each of the functions the EPSS-L1, EPSS-L2 and EPSS-L3 were required to perform.
(1) Results of experiments using a WebAttendant framework
The only works needed to complete to make EPSS modules to be reusable was to change several parts of each rule that WebAttendant was going to reuse.
We estimated the amount of work involved in modifying EPSS-L1 to be reused as EPSS-L2 or EPSS-L3. We also estimated the amount of work involved in modifying the EPSS-L2 to be reused as EPSS-L3. The estimation was based on the total number of reusable rules that the total number of operational steps involved in modifying rules. Table 7 shows the number of times using a module for a user with a different skill. **Table 8** shows the results of reusability experiment developing EPSSs for users with

Table 8 The results of reusability experiment developing EPSSs for users with different skill level in a same Web site.

module name	Built-in	Web-attendant
Event		
load	6	0
unload	1	0
mouseover	2	2
mouseout	2	2
long-focus	4	1
long-no-keypress	6	1
many-mouse-operations	6	1
many-scroll-operations	10	0
many-mouseover	44	11
many-back-button-return	28	7
click-hesitation	24	6
input-include-prohibited-char	20	5
Guidance		
pagechange	1	1
window on	15	15
balloon help on	13	13
wizard	5	5
Web page on	15	15
alert	5	5
total	207	90

Built-in: The total number of programming lines the creator had to modify to reuse modules =(B)×(F)
WebAttendant: The total number of operational steps to change rules for reusing modules =(D)×(F)
(B),(D) in Table 4, (F) in Table 7

different skill levels on the same Web site. As shown in Table 8, the total number of steps involved in modifying rules for reusing the modules was 90 steps. Reusable rules requiring no modifications are independent of the Web site contents as described in section 6.2.1(1). There are some reusable rules that required some changes which are dependent on the Web site contents as described in section 6.2.1(1). For example, "window on" module in Table 4, 1 step which defines an intervention message in a rule had to be changed.

(2) Results of experiments using a built-in EPSS framework

We developed EPSS-L1, EPSS-L2, and EPSS-L3 using a built-in EPSS framework. To reuse these EPSS functions, the creator had to modify required modules using JavaScript and Java languages, and had to embed each module for each target object in each HTML Web page. The work involved in developing the EPSS was the same as that is described in section 6.2.1(2). We estimated the amount of work based on the total number of programming lines we had to modify to make the EPSS modules reusable. As shown in Table 8, the total number of programming lines that needed to be modified was 207.

6.2.3 Comparison

Based on the results of experiments described in 6.2.1 and 6.2.2, the only work needed to complete to make the EPSS modules reusable was to change several parts of each rule that WebAttendant was going to use. For example, the creator changed the target object ID, and an intervention message to make a rule reusable. Moreover, some rules could be reused without changing any parts of the rule. Therefore, the work involved in making the EPSS reusable for another Web site was 158 operational steps for changing the rules in (a) developing EPSSs for Different Web sites and 90 operational steps in (b) developing EPSSs for users with a different skill level in the same Web site.

On the other hand, in the case of a built-in EPSS, the developer had to modify and embed JavaScript programs for each target object in each HTML Web page to define the EVENT and GUIDANCE module. The work involved in making the EPSS reusable was the task of embedding 422 programming-lines for (a) and 207 programming-lines for (b). From the results of these experiments, we concluded that the WebAttendant framework is superior to a built-in EPSS framework in terms of its reusability for developing EPSSs for different Web sites.

6.3 Discussion about Experiments to Verify the Superiority of WebAttendant over Built-in EPSS

We ran several sets of experiments to verify the superiority of WebAttendant over a built-in EPSS in terms of its cost-efficiency of development. We examined the cost-efficiency of WebAttendant development, and we concluded that WebAttendant development is cost-efficient because of its small amount of work and its high reusability for other Web sites and its capability to provide EPSS for users with different Web operation skill levels.

7. An Experiment to Verify the Usefulness and Efficacy of EPSS System Using WebAttendant

In order to verify the usefulness and efficacy of WebAttendant, we carried out experiments in which WebAttendant provided task-assistance service (EPSS service) to users of a Web training site. This Web training site was a task-oriented Web site where users had to register using their ID, check the availability of the courses, and fill out course application.

The first-time users tasks were searching to find how to register using their ID, finding the page for ID registration, and submitting their ID. First-time users carried out these tasks with or without task-assistance service by WebAttendant.

7.1 Outline of the Created Rules in the Experiment

We determined what kinds of rules are required for task-assistance. Then we created the following 43 rules. These rules consist of "condition part" and "execution part". The type of a context event and the current status of a context are described in the condition part of the rule. A method of guidance or the next status of a context is described in the execution part of the rule. Some examples of the rules are shown below.

(1) A rule for assessing the user's current purpose:

An example of the rule is as follows:

(Condition part) When the top page is loaded and the user performs MOUSEMOVE events 5 times or more on the page.

(Execution part) a wizard (Fig. 6) asking the user the purpose of his/her task is provided beside the Web page.

(2) A rule to help users track the process flow

of the business task
An example of the rule is as follows.

(Condition part) When the user checks "I have not registered ID yet" in the window asking whether or not the user has a registration ID.

(Execution part) A help balloon with the message "carrying out ID registration for the next step" is provided beside the ID registration item in the current window (Fig. 7).

(3) A rule that directs users until the completion of registration.

This rule helps guide navigation until the user completes the registration task, after Rule (1) has been activated. For example, when a user chooses ID registration task from the wizard, the system will tell the user what to do to complete the task, pointing to correct places until registration is completed. An example of the rule is as follows.

(Condition part) When the user's mouse moves over the input form of the ID registration page.

(Execution part) A help balloon with the message "Please input a six-digit number, and click on the registration button" appears on the input form.

7.2 The Results of the Experiment

The experiment was conducted over a 1-month period. Twenty people, all first-time WebAttendant task-assistance users, participated in the experiment. The participants carried out the experiment either with or without WebAttendant task-assistance service. WebAttendant tracked the participants' operations generating activity histories. The experiment was analyzed from four points of view:

(1) Whether or not a participant used WebAttendant task-assistance service
(2) Time spent to complete ID registration
(3) The number of a user's operation events
(4) The results of the questionnaire

7.2.1 Analysis of Time Spent to Complete Tasks

Figure 8 shows the average total time each participant spent to register. It also shows the average time each participant spent to complete each process of registration task. The average total time using WebAttendant task-assistance service was 2.73 minutes or shorter, while for those without a task-assistance service it was 5.84 minutes.

For ID registration, Users who carried out the experiment without WebAttendant had to visit

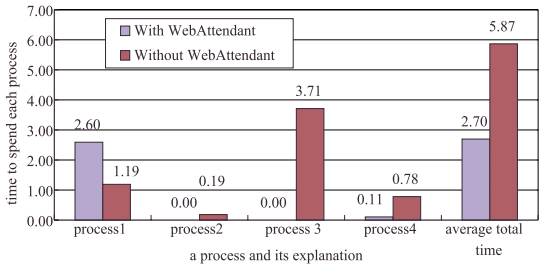


Fig. 8 The average time participants spent to register.

at least 4 specific Web pages. "Time spent on each process" in the Fig. 8 is an average time that users spent from the one go specific page to another page. For example, for registration task,

- The time for process # 1 is the average time a user spent loading the top page and the visitor center Web page.
- The time for process # 2 is the average time a user spent loading the visitor center Web page and the page describing how to register.
- The time for process # 3 is the average time a user spent loading the Web page describing how to register and the registration Web page.

It is important to mention that the average time for each process does include the time a user may have wasted going to and returning from pages unrelated to each specific process.

WebAttendant task-assistance helped users spent less time to complete each process, compared to when there was no assistance.

Users spent an average 2.60 minutes to complete process # 1 when they used WebAttendant task-assistance, compared to an average 1.19 minutes when there was no assistance. The difference in average time is due to the fact that several interventions, such as guidance windows and three different wizards had to be loaded and taught to the users in the case where there was assistance. The 1.41 minutes increased average time in process # 1 is insignificant compared to the 3.11 minutes reduction in total time needed to complete the task.

Users spent longer time to complete process # 1 through process # 3 mainly because they had difficulty finding the right links leading to task completion.

With task-assistance service, users didn't have to do process # 2 and # 3 because WebAttendant automatically changed the top page to

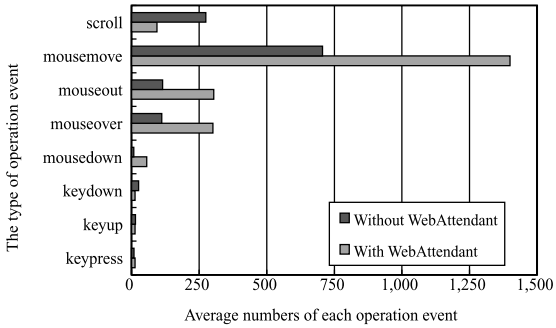


Fig. 9 Average number for each operation event.

the page where users started process # 4. Plus, balloon help messages and windows, which facilitate flow of each process, helped users to complete process # 4 more efficiently. As a result, average total time for registration reduced significantly.

7.2.2 Analysis of Each Experiment Based on the User's Operation Event

Figure 9 showed the results of numbers of user's each operation event during the task to register in the case with a task-assistance service and without.

- The numbers of SCROLL events decreased much more compared to the case without a task-assistance service. This was because users without a task-assistance service could not easily find the link to the next page to lead the next process of the task.
- Users could find the link easily thanks to the interventions such as a balloon help messages. The balloon help messages pointed out the link to a next page to lead the next process.
- The numbers of MOUSEMOVE events increased more compared to the case without a task-assistance service. The reason for this is that users frequently moved the mouse to the several interventions when they appeared.
- The numbers of MOUSEOVER, and MOUSEOUT events increased more compared to the case without a task-assistance service. The reason for this is owing to the balloon help. Many balloon help messages appeared when the users put the mouse over the link and a form object and they disappeared when the user put the mouse out of these objects. As a result, users had to repeat MOUSEOVER and MOUSEOUT operations if they needed to read messages again. This caused the numbers of MOUSEMOVE and MOUSEOUT events to increase.

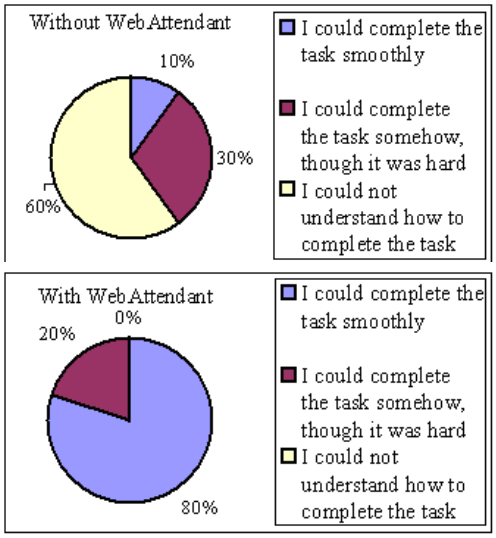


Fig. 10 Results of questionnaires on completion of tasks with or without WebAttendant.

7.2.3 Results of Questionnaires

Results of the questionnaires are shown in **Fig. 10**. With task-assistance service, 80 percent of the users completed their tasks with no difficulties and 20 percent of the users completed their tasks with some degree of difficulties. Without a task-assistance service 10 percent of users completed their tasks with no difficulties, 30 percent of the users completed their tasks with some degree of difficulty and 60 percent of users could not complete their tasks.

The following are the reasons why WebAttendant task-assistance service users completed their tasks more efficiently:

- Help windows and balloon help messages helped users to better understand the flow of each process.
- Balloon help messages helped the users by pointing out the links that users had to follow to proceed to complete a given task.

7.3 Discussion about the Experiment to Verify the Usefulness and Efficacy of EPSS System Using WebAttendant

Results of the experiments illustrate that WebAttendant task-assistance service can reduce the time required for completing a task. Interactive balloon help messages reduce the

time wasted on SCROLL operations mainly because help messages pointed out the link objects leading to the next process and navigated the users to complete a task. On the other hand, MOUSEMOVE, MOUSEOVER, and MOUSEOUT operations increased due to help windows, wizards and balloon help messages intervention. Results of the experiments also show that every intervention was effective in decreasing the completion time of a task.

8. Future Directions

Results of the experiments suggest the need for research in the following areas:

(1) Management of rules

We created 30 rules for the experiment. As the number of rules increases, keeping track of each rule and operation condition becomes very complicated. Thus making it very desirable to have a management tool to show the purpose and operating condition of each rule.

(2) Dealing with dynamically generated pages

If the Web server is linked to a database and generates pages dynamically, management of access to DOM objects in the Web page becomes complicated since the document structure may change. Even an Xpointer[13] will fail unless the original server defines the proper IDs. It is desirable to extend the rule-description technology to deal with some of the structural changes in such dynamic pages.

9. Conclusion

We proposed a framework for development of a content-independent EPSS called WebAttendant. In contrast to built-in EPSS, which is a part of Web content, WebAttendant's EPSS can be built independent of the Web content. We proposed the design, the structure and the functions of WebAttendant. We have also tested and evaluated its effectiveness.

We conducted a series of experiments to test the effectiveness of WebAttendant. Based on the results of these experiments, and we concluded that WebAttendant is highly effective as a platform of content-independent EPSS on the Web site.

Acknowledgments We would like to thank Takehisa Sato, Yukie Masuda, and Takayuki Itoh and other IBM staff members for their cooperation.

References

1) Amazon.com, http://www.amazon.com.
2) Ebay, http://www.ebay.com.
3) Stevens, G.H. and Stevens, E.F.: Designing Electronic Performance Support Tools: Talent Requirements, *Performance and Instruction*, Vol.24, No.2, pp.9–11 (1995).
4) CoachWare, http://sterlingnet.com/sterling/coachware.htm
5) Mmhelper, http://www.esmmi.com/product_more_weel.htm
6) Microsoft Agent, http://msdn.microsoft.com/workshop/imedia/agent/
7) QuickCards, http://www.epssinfosite.com/dd_qcard.html
8) Duke-Moran, C., Swope, G, Morariu, J. and deKam, P.: Performance Support Case Studies from IBM, International Society for Performance Improvement, *Performance Improvement Journal*, Vol.38, No.7 (1999).
9) American FactFinder, http://www.census.gov
10) Furui, Y., Aoki, Y. and Hijikata, Y.: A Web proxy for embedding operation profiling and automatic navigation programs in Web pages, *The 60th IPSJ* 5S-8, March 16, pp.421–422 (2000).
11) Aoki, Y. and Nakajima, A.: User-Side Web Page Customization, *The 8th International Conference on Human Computer Interaction* (*HCI International '99*), Vol.1, pp.580–584 (1999).
12) Aoki, Y., Ando, F. and Nakajima, A.: Web Operation Recorder and Player, *The 7th International Conference on Parallel and Distributed Systems* (*ICPADS2000*), pp.501–508 (2000).
13) XML Pointer Language (Xpointer) Version 1.0 W3C, Candidate Recommendation, 7 June (2000). http://www.w3.org/TR/xptr

Appendix

A.1 Programming Lines

Table 9 Total number of programming lines for each module using built-in EPSS.

module name	(a)	(b)	(c)	(d)	(e)	total
Load	1	0	8	3	0	12
unload	1	20	8	3	0	32
mouseover	1	20	10	3	0	34
mouseout	1	31	10	3	0	45
click	1	95	10	3	0	109
submit	1	30	4	3	0	38
no-change	4	49	35	120	15	223
repeated-focus	6	49	35	122	15	227
specified -focus-order	16	49	35	145	32	277
specified -focus-unorder	16	49	35	145	32	277
Long-focus	4	49	35	90	10	188
Long-no -keypress	6	49	35	105	22	217
many-mouse -operations	6	30	10	123	24	193
many-scroll -operations	2	33	10	110	15	170
many-mouseover	4	20	10	110	15	159
many-back -button-return	4	25	10	140	37	216
click-hesitation	4	45	35	126	18	228
input-include -prohibited-char	4	40	12	93	11	160

(a): Handling Event
(b): Adding information to a event
(c): Making an event as an object
(d): Making an context event
(e): Updating a context
total: Total number of programming lines

(Received June 7, 2001)
(Accepted November 14, 2001)

Yuko Ikehata received the M.E. degrees from Keio University, Japan, in 2000. She also studied computer science in Ecole Centrale de Nante, France, as an exchange student in 1999. She has been working in IBM Research, Tokyo Research Laboratory since 2000. Her current research interests are information visualization, Internet application technology, computer supported collaborative work/learning, and human interface. She received Best Paper Award of IPSJ National Convention and also Fujiwara Memorial Award from Keio University in 2000. She is a member of IPSJ.

Toshio Souya was born 1964 in Tokyo and received the B.E., M.E., and Ph.D. degrees from Tokyo University of Agriculture and Technology, Tokyo, Japan, in 1987, 1989 and 1992, respectively. In 1992, he joined IBM Research, Tokyo Research Laboratory. He moved to IBM Global Services, Business Innovation Services in early 2001 and currently he is working as an associate consultant. His research interests includes human interface, especially pen input, and computer supported collaborative learning. He is a member of IPSJ, IEEE and ACM.

Yoshinori Hijikata received the B.E. and M.E. degrees from Osaka University, Osaka, Japan, in 1996 and 1998, respectively. In 1998, he joined IBM Research, Tokyo Research Laboratory. Currently, he is a Ph.D. candidate in Osaka University. His research interests are on human interface and Internet application technology. He is a member of IPSJ, the Japan Society for Software Science and Technology (JSSST), the Human Interface Society (HIS) and the IEEE.