

Extensible Web-based Troubleshooter Generation for Computer Administration

Takao Shimomura

Abstract—Conventional troubleshooters do not provide detailed verification to find the final cause of a failure to fix it. This paper presents a method for generating Web-based troubleshooters that deal with troubles about computer system administration and networks. In the automated diagnosis mode, it automatically detects the cause of a trouble based on the values users entered and shows a solution to recover from the trouble. In the manual diagnosis mode, users can learn the diagnosis process such as what could cause the trouble, how to identify the failures corresponding to the trouble, and how to solve each of those failures. The diagnosis procedure can be easily defined by using several kinds of command tags, which makes troubleshooters extensible.

Keywords—Automatic diagnosis, Command tags, Computer system troubles, Web applications.

I. INTRODUCTION

Troubleshooters have been widely used in the industrial world [3]. Monitors observe the behavior of a manufacturing system. Any monitor detecting abnormal behavior switches the control immediately to a troubleshooter which conducts fault diagnosis. The operator is informed to eliminate the fault source, such as to repair or replace a device. By nature, troubleshooting tasks are complex and require processing of numerous information elements. For example, as described in [14], in troubleshooting a simple electrical DC circuit, specific knowledge of the function of its components such as voltage sources and resistors, and general knowledge about the relation between voltage, current and resistance and about the conservation of energy and charge are needed to be able to determine how the circuit should function. Troubleshooting, that is, diagnosing and repairing faults in a technical system, constitutes an important part of most technical jobs. It depends strongly on human expertise and intuition as well as time-consuming trial-and-error work [10], [15].

DAPFA [9] is a distributed diagnostic algorithm for fault analysis in power networks to improve security, reliability and quality of the power supply. Upon an occurrence of a fault, protective devices such as relays and circuit breakers operate quickly in order to isolate the faulty portion of a network. This causes a surge of alarms to be registered at the control center within a few seconds. If an alarm can be processed at a substation or area operation level, it can be eliminated without transferring it to the central station. Performance Booster (PB) [17] derives a performance model from a specification in UML to automate the performance analysis, and to explore design changes using diagnostic and design-change rules. It identifies critical performance features such as bottlenecks and

long paths, and generates an improved performance model and system configuration such as the allocation of processors. Liu [6] proposes a firewall verification and troubleshooting tool. It takes as input a firewall policy and a given property, then outputs whether the policy satisfies the property. If a firewall policy does not satisfy the property, it outputs which rules cause the verification failure.

In troubleshooting, rule-based expert systems have been used, where systems typically suggest possible actions after an operator enters the observed symptoms [11], [4]. KBTS[5] is a Web-based troubleshooting system for automotive refrigeration system. It uses IF-THEN rule-based knowledge. RCBR [13] is a hybrid solution retrieval system using rule-based reasoning and case-based reasoning. The rule-based reasoning uses If-Then forms. After the error type of a problem is diagnosed, it retrieves the solution from corresponding case base with case-based reasoning approach by recalling a previous similar situation and reusing information and knowledge of that situation. In a diagnostic process for building explanations of patient's observed temporal evolution [7], [8], fuzzy logic has been used to make the representation of quantitative and qualitative imprecise temporal information possible. Desbazeille et al. [2] monitored large diesel engines by analyzing the crankshaft angular speed variations. Neural networks are used for pattern recognition of the angular speed waveforms in normal and faulty conditions. Wu et al. [16] proposed a fuzzy-logic inference to develop the diagnostic rules of the data base in a fault diagnosis system. Q-KE-CLD [1] uses a fuzzy Bayesian inference to diagnose print defects. Users review the results in descending order of probability and decide what action they will take.

This paper presents a method for generating Web-page based troubleshooters, which diagnose troubles about computer system administration and networks. To implement the method this paper proposes, the following requirements are taken into account.

- 1) Users can find the causes and solutions to the troubles of computer networks without technical knowledge or experience.
- 2) Users can gradually obtain technical knowledge and experience while using this system.
- 3) Users can share and add their obtained knowledge to the system so that the system will be enhanced and refined.

This troubleshooter digs out all possible causes (hypotheses) from the current status of a trouble, and then users can examine those hypotheses in turn. To illustrate the status of a trouble, the detailed symptom of a failure and how to check whether the failure really occurs or not are presented. After

T. Shimomura is with the Department of Information Science and Intelligent Systems, University of Tokushima, Tokushima, 770-8506 JAPAN (e-mail: shimomura@is.tokushima-u.ac.jp).

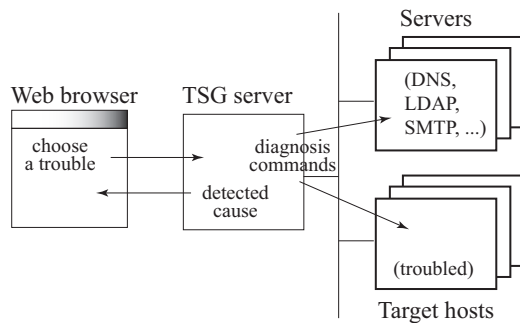


Fig. 1. System configuration of the generated troubleshooter

the cause is identified, it presents a solution to fix the trouble. In addition, it demonstrates the users' diagnosis processes by using a diagnosis stack. If a hypothesis is not verified, another hypothesis will be investigated by popping up a node from the stack to finally lead to a leaf node that provides a solution to the trouble. In an automated diagnosis mode, the system automatically detects the cause of the failure based on the values the users entered.

II. WEB-BASED TROUBLESHOOTER GENERATION

A. System configuration

When a system administrator checks computer systems or networks, he issues several administrative commands to a target machine and related server machines to monitor the system configurations and statuses of those machines. This system automates the same process as the system administrator does. When a trouble occurs in a computer system, it is not always a failure of the system, but it might be caused by a user's mistake such as entering a wrong password, specifying a wrong IMAP server, and so on. This system covers not only system failures but also user-related mistakes. Troubles could be caused by system failures or users' mistakes. In this system, a user first accesses the TSG (Troubleshooter Generator) server using a Web browser, and then chooses a trouble about computer systems and networks among a table of classified troubles shown by the server. TSG server issues diagnosis commands to check the target machine, which has a trouble. If necessary, TSG server also issues diagnosis commands to other servers such as LDAP, NIS, DNS, SMTP, IMAP, etc. to check their statuses and get some information. Finally, TSG server shows the cause and solution to the trouble the user chose. Figure 1 shows the system configuration of the troubleshooter generated from trouble analysis pages, which will be described in detail in Section IV-A.

B. Deployment of trouble analysis pages

To describe troubles and their causes, we define trouble analysis pages. Trouble analysis pages describe troubles (that is, failures), detailed symptoms, how to check the symptoms, and the solutions to the troubles. They are defined in JSP pages [12] by using several types of TS (Troubleshooting) tags. Troubleshooting page directory (tsp/) is a TSG root directory for all trouble analysis pages. Under this root directory, we

have three types of resources, trouble classification subdirectories (e.g. user/), trouble analysis subdirectories (e.g. CannotLogin/), and trouble analysis pages (e.g. CannotLogin.jsp). In general, troubleshooting requires hierarchical knowledge, which forms a complicated network as shown in Fig. 2. To simplify the hierarchical knowledge, we have introduced directory hierarchy and symbolic links. First, we classify all the troubles and arrange them in appropriate layered subdirectories. Second, in each subdirectory, we describe only one trouble and its possible sub-troubles, which could cause the trouble. Other derived sub-troubles and causes are arranged in other subdirectories, and if necessary, they can be referred to by symbolic links as shown in Fig. 3.

C. Trouble classification subdirectories

Trouble classification subdirectories (e.g. user/) are used to classify troubles. If we create a user/ subdirectory under the TSG root directory, this subdirectory's name will be shown in the classification table of the root page of the system. If we would like to display another title such as "User interface" for this trouble classification subdirectory, we need to create a trouble analysis page (user.jsp) whose name is the same as this subdirectory, and define the title as follows:

```
<page type="title" title="User interface"/>
```

D. Trouble analysis subdirectories

A trouble analysis subdirectory (e.g. CannotLogin/) is a leaf directory; that is, it has no subdirectories. This directory describes only one trouble to make it simple to create and allocate trouble analysis pages. A trouble analysis page (which is called a parent node) and its possible sub-troubles (which are called child nodes) that could cause the trouble, are defined here. Each sub-trouble's causes are defined in separate trouble analysis subdirectories. To differentiate a trouble analysis subdirectory from a trouble classification subdirectory (e.g. user/), the trouble analysis subdirectory's name (e.g. CannotLogin/) starts with a capital letter. This trouble analysis subdirectory contains only one parent trouble analysis page (e.g. CannotLogin.jsp) whose name is the same as itself.

E. Trouble analysis pages

A trouble analysis page (e.g. CannotLogin.jsp) is a JSP page in which a <page> tag is written. The <page> tag is only used to specify the type of the page and define the title of a trouble. When the page is of type "admin" or "trouble", its child nodes' titles will be shown in an information Web page that corresponds to this page. The JSP name of a child node (e.g. CannotLogin.useNIS.jsp) is created by appending a simple name to the JSP name of its parent node (e.g. CannotLogin.jsp). For example, if CannotLogin.jsp is of type "admin", a user will choose an appropriate system configuration in this information Web page. Its child nodes such as CannotLogin.useEtcPasswd.jsp, CannotLogin.useNIS.jsp and CannotLogin.useLDAP.jsp describe various kinds of configurations. They will be automatically collected by the system, and their titles will be shown in this information Web page.

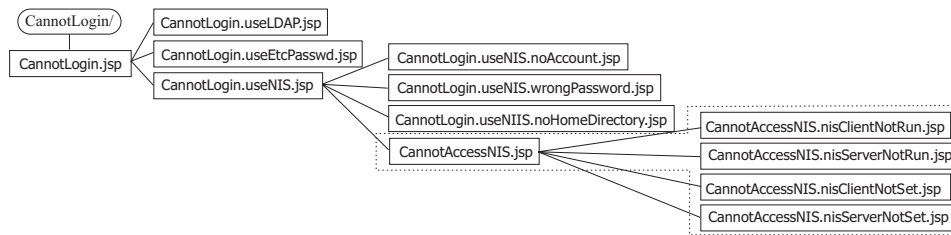


Fig. 2. Network hierarchy for trouble analysis pages

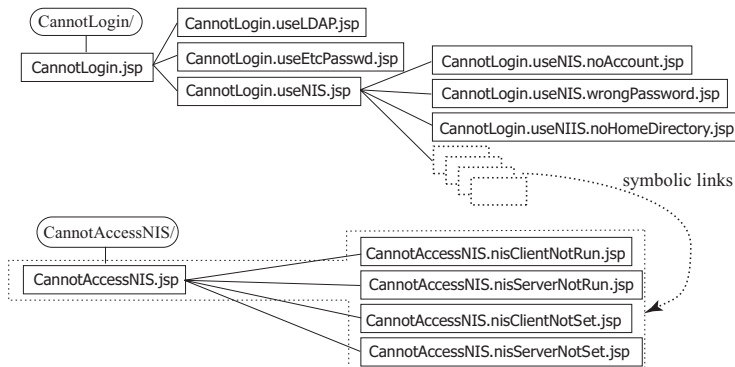


Fig. 3. Plain hierarchy for trouble analysis pages

For example, if `CannotLogin.useNIS.jsp` is of type "trouble", in the manual diagnosis mode, a user will examine all possible sub-troubles (or causes) that could cause this trouble in turn. In this case, its child nodes such as `CannotLogin.useNIS.noAccount.jsp`, `CannotLogin.useNIS.wrongPassword.jsp` and `CannotLogin.useNIS.nisClientNotRun.jsp` will be automatically collected, and their titles will be shown in this information Web page. These child nodes are leaf nodes of type "diag", and the solutions to the detected causes are described there. In the automated diagnosis mode, if a user enters some information about the trouble and clicks on the "Auto Diagnosis" button, the system will execute `<exec>` and `<goto>` tags described in `CannotLogin.useNIS.jsp` to make control transfer to an appropriate one of these child nodes.

As shown in Fig.3, the trouble of `CannotLogin.useNIS.nisClientNotRun.jsp` is defined in another directory (`servers/nis/CannotAccessNIS/`) as `CannotAccessNIS.nisClientNotRun.jsp`. Therefore, `CannotLogin.useNIS.nisClientNotRun.jsp` in `user/login/CannotLogin/` directory is symbolically linked to `CannotAccessNIS.nisClientNotRun.jsp` in `servers/nis/CannotAccessNIS/` directory.

III. TROUBLESHOOTING TAGS

A. Hierarchy of troubleshooting tags

A trouble analysis page is defined by a `<page>` tag, which can include several types of tags such as `<symptom>`, `<check>`, `<auto>` and `<solution>` tags. An `<auto>` tag is used for automatic diagnosis, which further includes `<from>`, `<button>`, `<exec>` and `<goto>` tags. The type attribute of a

`<page>` tag has one of four values such as "title", "admin", "trouble" and "diag". Some inner tags are only used for a page that has a specific type.

B. Functions of `<page>` tags

A `<page>` tag defines a trouble analysis page. It has two mandatory attributes, type and title.

A page whose type is "title" is only used to display the title of the page in an information Web page. For example, `<page type="title" title="Login"/>` defines a page which only shows the title of an analysis classification directory (login/).

A page whose type is "admin" is used for users to choose an appropriate system configuration of the target machine (for example, LDAP, NIS, etc. for user authentication) among its child nodes, which are also defined as trouble analysis pages. For example, `<page type="admin" title="Cannot log in">` defines a page which shows the titles of child nodes which describe several types of system configuration.

A page whose type is "trouble" is used to display the title of a trouble and all possible failures that could cause this trouble, which are derived from its child nodes. For example, `<page type="trouble" title="NIS is used for account authentication">` defines a page which describes a trouble "Cannot log in" under a NIS system configuration.

A page whose type is "diag" is a leaf node and used to display the solution to the detected cause. For example, `<page type="diag" title="Wrong password">` defines a page which describes a solution to create a new password.

C. Functions of `<exec>` tags

A `<exec>` tag executes commands for automatic diagnosis of troubles. It has one mandatory attribute `cmd` and four

optional attributes host, stdin, stdout and exit.

The cmd attribute defines a command line that the shell can execute for diagnosis. The host attribute defines a remote host where the command will be executed. For example, to get the passwd map of a NIS service, `<exec host="$nisServer" cmd="ypcat passwd"/>` tag is written. Before the command is executed, \$nisServer will be replaced with the host name of a NIS server a user specifies.

The stdin attribute defines the name of the data that will be used as standard input to the command. The stdout attribute defines the name of the data that will be stored as standard output from the command. The standard output can be used as the standard input to the commands of subsequent `<exec>` tags. For example, when `<exec cmd="egrep ^$account:" stdout="accountLine"/>` tag is executed, the standard output of the previous `<exec>` tag will be used as the standard input of the command, and then the data "accountLine" will be obtained as the standard output of the command which contains a line in the passwd file that corresponds to a user (\$account).

The command specified in a `<exec>` tag cannot be always executed. There are some cases such that the command is missing or the command abnormally exits. To deal with these cases, we have introduced an exit attribute. The exit attribute defines a trouble analysis page to which control will transfer when the command has no standard output. To check whether a NIS client is properly running or not, for example, `<exec host="$nisClient" cmd="/sbin/service ypbind status" exit="CannotLogin.useNIS.nisClientNotSet">` tag is used. If a NIS client is not properly set up, this command will have no standard output. In this case, as a cause of the trouble "Cannot log in", trouble analysis page "CannotLogin.useNIS.nisClientNotSet.jsp" will be shown, which illustrates how to set up a NIS client.

To facilitate the analysis of troubles, we have prepared some predefined TSG commands, which can be used as a command for `<exec>` tags. For example, "extract regExp replacement" command reads lines from the standard input, replaces a part of each line that matches regular expression "regExp" with "replacement", and then outputs them to the standard output. A command "crypt passwd salt" generates an encrypted password based on a password and a salt. This command is used to compare the password a user enters with an encrypted password in a passwd file.

D. Functions of `<goto>` tags

When we examine a system failure, we execute several commands to monitor the system behavior and finally detect the cause of the failure by judging from the standard outputs of these commands. Therefore, we have introduced the following `<goto>` tags which check the standard outputs of commands executed in `<exec>` tags.

A `<goto>` tag makes control transfer to an appropriate trouble analysis page after it checks the standard output of the previous `<exec>` tag. It has one mandatory attribute page and five optional attributes stdin, eq, ne, ma and um. Attributes ma and um specify a regular expression for text matching.

The stdin attribute defines the name of the data that has been stored as the standard output of the previous `<exec>` tag. The page attribute defines a trouble analysis page to which control will transfer when a condition is satisfied.

If at least one line of attribute stdin equals the value of attribute eq, control will transfer to a trouble analysis page specified by the page attribute. If all lines of attribute stdin do not equal the value of attribute ne, control will transfer to a trouble analysis page specified by the page attribute. If at least one line of attribute stdin has a part that matches attribute ma, control will transfer to a trouble analysis page specified by the page attribute. If all lines of attribute stdin do not have a part that matches attribute um, control will transfer to a trouble analysis page specified by the page attribute.

For example, let's assume that `<goto um="$account:" page="CannotLogin.useNIS.noAccount.jsp"/>` is written after `<exec host="$nisServer" cmd="ypcat passwd"/>` tag. If a user (\$account) does not have an account in the passwd file of the NIS server, all lines of the standard output of "ypcat passwd" command do not have a part that matches "\$account:". Then, control will transfer to "CannotLogin.useNIS.noAccount.jsp".

IV. TROUBLESHOOTING WITH TROUBLE ANALYSIS PAGES

A. Example of a trouble analysis page

We here consider an example of a trouble analysis page "CannotLogin.useNIS.jsp". Because its parent node "CannotLogin.jsp" is of type "admin", this page describes one of system configurations for account authentication. Its child nodes include some trouble analysis pages such as CannotLogin.useNIS.noAccount.jsp, CannotLogin.useNIS.wrongPassword.jsp and CannotLogin.useNIS.nisClientNotRun.jsp that describe the failures or causes that could cause the original trouble "Cannot log in" under the system configuration of a NIS service.

An `<auto>` tag is written in this page for automatic diagnosis. If this tag is not written in this page, users need to perform a manual diagnosis. Because its child nodes' titles are automatically collected by the system and shown in an information Web page that corresponds to this page, users click on these child nodes in turn to detect its cause. Each child node describes the symptom of a failure, how to check whether the failure has occurred or not, and a solution to the failure. According to these pieces of information, users can confirm whether this failure has really occurred and whether it caused the original trouble. If this failure does not occur, users can backtrack their diagnosis process to check the other child nodes that have not yet been examined.

B. Automatic diagnosis

An `<auto>` tag contains one `<form>` tag and several `<exec>` and `<goto>` tags. The `<form>` defines some values necessary for the automatic diagnosis. In the "CannotLogin.useNIS.jsp" page, values nisClient, nisServer, account, and password are defined as the query data of a `<form>` tag. When a user clicks on the "Auto Diagnosis" button, these values will be sent to the TSG server, and then this page itself will receive

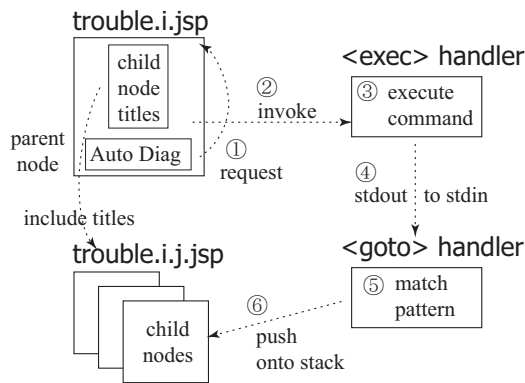


Fig. 4. Implementation of the troubleshooter generator

them. Therefore, inside the following `<exec>` and `<goto>` tags, these values can be referred to by `$nisClient`, `$nisServer`, `$account`, and `$password`. The details will be described in Section V-C.

V. IMPLEMENTATION

A. Program structure

Figure 4 illustrates the program structure of the Web-based troubleshooter generation. The programs are classified as TSG pages and TA pages. TSG pages control the page transfer of the system. TA pages are pre-installed trouble analysis pages and can be enhanced by users.

A user accesses an entry JSP page of the system (`index.jsp`) by using a Web browser. First, `index.jsp` initializes a stack and stores it in a session. The stack contains some nodes that have been already examined by the user. Next, `index.jsp` forwards a request to `guide.jsp`, which defines two frames, `stackFrame` and `infoFrame`. The left frame (`stackFrame`) sends a request to `stack.jsp`, and the right frame (`infoFrame`) sends a request to `info.jsp`. Then, `stack.jsp` displays the contents of the stack, and `info.jsp` displays an information Web page that shows the contents of the top node of the stack. In `stackFrame`, the tiles of the nodes contained in the stack are displayed. To display such a title, `stack.jsp` sends a request to a trouble analysis page (`trouble.i.jsp`) and includes its output. The trouble analysis page returns the title that is defined in the title attribute of its `<page>` tag. In `infoFrame`, `info.jsp` forwards a request to the top node of the stack (`trouble.i.jsp`), and then the contents of the trouble analysis page are displayed.

For the root node (`tsp/`), `info.jsp` displays a table of classified troubles, which contains trouble classification subdirectories (e.g. `user/`, `servers/`, and `user/login/`). When a user clicks on a node of the stack shown in `stackFrame`, a request will be sent to `pop.jsp`, and it will pop up some nodes from the stack so that the clicked node will be the top of the stack. When the user clicks on one of child nodes shown in `infoFrame`, a request will be sent to `push.jsp`, and it will push down the clicked node onto the stack.

B. Trouble analysis pages and their tag handlers

When a trouble analysis page (`trouble.i.jsp`) is displayed in `infoFrame`, its child nodes are also shown in an information

Web page that corresponds to this page. To display the child nodes' titles, it sends a request to these nodes (`trouble.i.jsp`) and includes their output.

In an automatic diagnosis mode, the following processes are conducted. (1) When a user clicks on the "Auto Diagnosis" button, a request will be sent to the same page to receive the values the user entered. (2) The `<exec>` tag written in this page will invoke its tag handler. (3) The `<exec>` tag handler will execute a specified command. (4) The subsequent `<goto>` tag handler will get the standard output from the previous `<exec>` tag handler. (5) It will then match the standard output with a specified value of its attribute. (6) According to the result of matching, it will determine an appropriate child node (`trouble.i.jsp`) to which control should transfer, and then generates a temporary JSP page to inform the user of this child node in a JavaScript alert window. Finally, a request will be sent from this generated temporary JSP page to `push.jsp` so that the child node will be pushed down onto the stack.

C. Execution of tags

This section describes the automatic diagnosis in more detail. (1) When a user clicks on the "Auto Diagnosis" button, the values defined in the `<form>` tag such as `nisClient`, `nisServer`, `account`, and `password` will be sent to the same page. (2) This page will receive these values that the user entered for automatic diagnosis. First, `<exec host="$nisServer" cmd="/sbin/service ypsserv status" exit="CannotLogin.useNIS.nisServerNotSet.jsp" />` tag will be processed. If the user enters "nisweb" as the machine name of a NIS server, the part of "`$nisServer`" will be replaced with "nisweb". Because a host attribute is specified in this tag, the corresponding tag handler will execute an ssh command "`ssh nisweb /sbin/service ypsserv status`".

In this system, we assume that all machines in the network can be logged in from the TSG machine by the ssh command without entering a password. However, a target machine is not always connected to the TSG machine, or it might not be able to be logged in by the ssh command. Therefore, before the specified ssh command is issued, the system implicitly issues a test command "`ssh nisweb pwd`". If this test command has no standard output, the system will judge that the target machine is not properly connected to the TSG machine, and lead the user to an error page "`administration/network/SshNotConnected/SshNotConnected.jsp`". This error page illustrates how to connect the target machine by ssh; that is, how to set up a network interface, create `id_rsa.pub` without a password, and copy it to `/.ssh/authorized_keys` of the target machine.

(3) In this way, three `<exec>` tags have been executed because both the NIS server and client are running properly. If the user enters "simomura" as the account name, `<exec cmd="egrep '^$account:'" stdout="accountLine" />` tag will execute a command "`egrep '^simomura:'`" by using the standard output from the previous command "`ypcat passwd`" as its standard input. The standard output from this tag will be stored in data "accountLine".

(4) `<exec cmd="extract ':(\\$.?\\$.+?\\$)' '$1'"`

stdin="accountLine" stdout="salt" /> tag obtains a salt from data "accountLine".

(5) <exec cmd="extract ':(\\\$.?\\\$.+?\\\$.+?):' '\$1'" stdin="accountLine" stdout="encryptedPwd" /> tag obtains an encrypted password from data "accountLine".

(6) <exec cmd="crypt '\$password' '\$salt'" stdout="encryptedEnteredPwd" /> tag obtains the encrypted password that corresponds to the password the user entered.

(7) Finally, <goto ne="\$encryptedPwd" stdin="encryptedEnteredPwd" page="CannotLogin.useNIS.wrongPassword.jsp" /> tag judges that the user's entered password is not consistent with the password in the passwd file of the NIS server, and then pushes down a child node "CannotLogin.useNIS.wrongPassword.jsp" onto the stack. The user will be lead to this page, which demonstrates the symptom of this failure, how to check the failure and how to create a new password.

VI. EXAMPLE OF AUTOMATED DIAGNOSIS

A. Cannot log in

This section illustrates some pages that are shown in this system when a user cannot log in a target machine, where a NIS server is used for user authentication. The left frame (stackFrame) of Fig. 5 (b) shows a sequence of processes a user has chosen. The user first selected a trouble analysis subdirectory "Login" from a classified table shown in the root page. Then, he chose a trouble analysis page "Cannot Log in", where he has selected a system configuration for user authentication "NIS is used for account authentication". In the right frame (infoFrame) of Fig. 5 (b), the title of this page, manual diagnosis, a table of child nodes' titles, and automatic diagnosis are shown.

He can click on each of child nodes in turn to know the symptom of the corresponding failure, whether that failure occurs or not, and how to fix this failure if it causes the trouble. When he confirms that one child node is not a real cause, he can backtrack his diagnosis process to click on another child node. The child nodes that have been examined are shown in green (already examined), and the others in pink (not yet examined).

B. Automatic diagnosis status

In the page shown in Fig. 5 (b), if the user enters a target machine, NIS server, his account and password, and then clicks on the "Auto Diagnosis" button, the automated diagnosis will start. He will be informed of the result of the automated diagnosis. There, he can (1) know which page control will transfer to, that is, which cause has been detected, (2) some values that have been used and generated in the automated diagnosis process, and (3) a sequence of executed <exec> and <goto> tags.

C. Automatically detected cause

When the user clicks on "OK" in the dialog, a trouble analysis page will be shown, which describes that the trouble

"Cannot log in" is caused by "Wrong password". This page shows the trouble symptom, how to check this failure, how to deal with this diagnosis result, and the solution to this trouble, that is, how to create a new password. In a manual diagnosis mode, the user can click on the "Analyze another case" button to backtrack his diagnosis process. If he clicks on this button, its parent node will be shown again, and the background color of this child node's title will be changed from pink (not yet examined) to green (already examined).

VII. EVALUATION

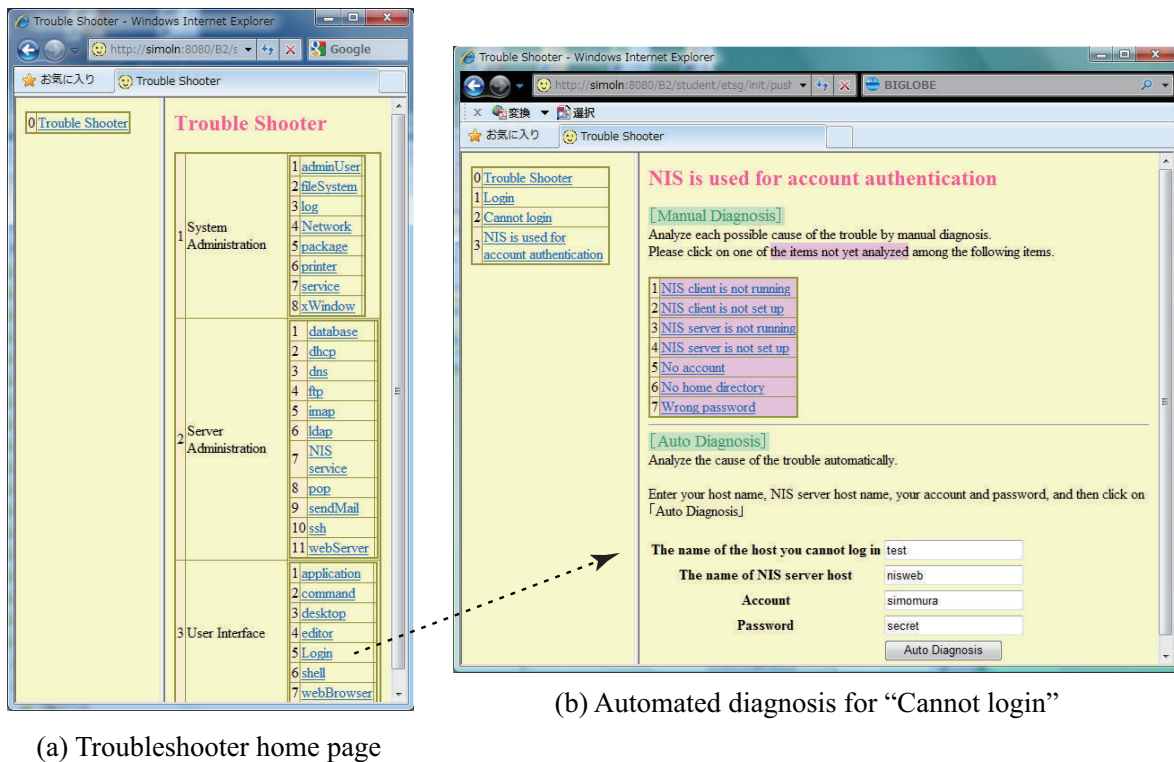
Troubleshooting requires a lot of knowledge and experience. Some people cannot detect the causes of troubles or solve those troubles. Therefore, a lot of work is imposed on system administrators. This system eases administrators' tasks. Even though the people do not know how to fix the troubles, if they conduct a manual diagnosis in this system, they can learn a lot of knowledge and obtain useful experiences.

We prepared some test machines which we can configure freely to test and evaluate the system. If some troubles happen in these machines, both a user and a system administrator can detect their causes by using this system. If they use this system in the manual diagnosis mode, they can know all possible causes of a trouble, and examine each possible cause in turn by issuing some commands according to the description of trouble analysis pages about how to check whether the corresponding failures have occurred or not. Finally they can obtain the knowledge and experience about what the symptom is like for a specific trouble, how to check the failure corresponding to the trouble, and how to solve that failure. Although it takes a lot of time to conduct the manual diagnosis processes, they can learn how to troubleshoot computer systems and networks.

Instead, if they use the automated diagnosis mode, they can detect the cause of a trouble quickly. To detect the cause of a trouble that a user cannot log in under the system configuration of a NIS service, it took only 12 seconds, most time of which was required to enter the names of a NIS server and a NIS client, and a user's account and password. On the other hand, when the manual diagnosis was conducted, it took 215 seconds in total because each of seven pages had to be examined in turn.

VIII. CONCLUSION

This paper has presented a Web-based troubleshooter generator, with which we can automatically detect the cause of a trouble and know how to solve the trouble. In its manual diagnosis mode, we can learn all possible causes of the trouble and obtain a lot of knowledge and experiences for troubleshooting. We can easily enhance this system by adding trouble analysis pages to deal with other troubles. However, to automatically detect the cause of a trouble, this system issues some commands to a target machine that are specified in <exec> tags. If some dangerous commands are executed such as rm and mv by mistake, the target system will be destroyed. To avoid this problem, we are going to restrict commands to be executed in <exec> tags in the future. In addition, we will enhance the system so that it will lead a user to an appropriate



(a) Troubleshooter home page

(b) Automated diagnosis for "Cannot login"

Fig. 5. Stack and the information Web page displayed by CannotLogin.useNIS.jsp

trouble analysis page only by entering the description of a trouble.

REFERENCES

- [1] Choe, P., Lehto, M. R., Park, H. J. and Allebach, J.: A query-based cross-language diagnosis tool for distributed decision making support, *Computers & Industrial Engineering*, Vol. 57, No. 1, pp. 37–45 (2009).
- [2] Desbazeille, M., Randall, R., Guillet, F., Badaoui, M. E. and Hoisnard, C.: Model-based diagnosis of large diesel engines based on angular speed variations of the crankshaft, *Mechanical Systems and Signal Processing*, Vol. 24, No. 5, pp. 1529–1541 (2010).
- [3] Jeng, W. H. and Liang, G. R.: Reliable automated manufacturing system design based on SMT framework, *Computers in Industry*, Vol. 35, No. 2, pp. 121–147 (1998).
- [4] Jeon, W.-S.: An expert system for estimation of fatigue properties of metallic materials, *International Journal of Fatigue*, Vol. 24, No. 6, pp. 685–698 (2002).
- [5] Liang, J. S.: A Web-based automotive refrigeration troubleshooting system applying knowledge engineering approach, *Computers in Industry*, Vol. 61, No. 1, pp. 29–43 (2010).
- [6] Liu, A. X.: Firewall policy verification and troubleshooting, *Computer Networks*, Vol. 53, No. 16, pp. 2800–2809 (2009).
- [7] Palma, J., Juarez, J. M., Campos, M. and Marin, R.: Fuzzy theory approach for temporal model-based diagnosis: An application to medical domains, *Artificial Intelligence in Medicine*, Vol. 38, No. 2, pp. 197–218 (2006).
- [8] Rahimi, S., Gandy, L. and Mogharreban, N.: A web-based high-performance multicriteria decision support system for medical diagnosis, *International Journal of Intelligent Systems*, Vol. 22, No. 10, pp. 1083–1099 (2007).
- [9] Rayudu, R. K.: A knowledge-based architecture for distributed fault analysis in power networks, *Engineering Applications of Artificial Intelligence*, Vol. 23, No. 4, pp. 514–525 (2010).
- [10] Schaaftal, A., Schraagen, J. M. and Van Berlo, M.: Cognitive task analysis and innovation of training: The case of structured troubleshooting, *Human Factors*, No. 42, pp. 75–86 (2000).
- [11] Stein, E. W., Pauster, M. C. and May, D.: A knowledge-based system to improve the quality and efficiency of titanium melting, *Expert Systems with Applications*, Vol. 24, No. 2, pp. 239–246 (2003).
- [12] : Sun Microsystems, Inc. : *JavaServer Pages Technology*, <http://java.sun.com/products/jsp> (2006).
- [13] Tung, Y.-H., Tseng, S.-S., Weng, J.-F., Lee, T.-P., Liao, A. Y. and Tsai, W.-N.: A rule-based CBR approach for expert finding and problem diagnosis, *Expert Systems with Applications*, Vol. 37, No. 3, pp. 2427–2438 (2010).
- [14] van Gog, T., Paas, F. and van Merrinboer, J. J.: Effects of process-oriented worked examples on troubleshooting transfer performance, *Learning and Instruction*, Vol. 16, No. 2, pp. 154–164 (2006).
- [15] Van Merriënboer, J. J. G. and Sweller, J.: Cognitive load theory and complex learning: Recent developments and future directions, *Educational Psychology Review*, No. 17, pp. 147–177 (2005).
- [16] Wu, J.-D., Wang, Y.-H. and Bai, M. R.: Development of an expert system for fault diagnosis in scooter engine platform using fuzzy-logic inference, *Expert Systems with Applications*, Vol. 33, No. 4, pp. 1063–1075 (2007).
- [17] Xu, J.: Rule-based automatic software performance diagnosis and improvement, *Performance Evaluation*, Vol. 67, No. 8, pp. 585–611 (2010).

Takao Shimomura obtained his PhD in Computer Science from Tokyo Institute of Technology in Japan. He is currently a professor of Dept. of Information Science and Intelligent Systems at University of Tokushima in Japan. His research interests include software design automation, program visualization, and automated debugging.