ELSEVIER

Interacting with Computers 13 (2000) 229–251

Interacting with Computers

www.elsevier.com/locate/intcom

RemUSINE: a bridge between empirical and model-based evaluation when evaluators and users are distant

F. Paternò*, G. Ballardin

CNUCE-C.N.R, Via V.Alfieri, 1-56010 Ghezzano, Pisa, Italy

Received 29 July 1999; revised 3 June 2000; accepted 7 June 2000

Abstract

There are few computer-aided approaches that provide a model-based usability evaluation using empirical data. This paper proposes a solution that allows designers to remotely evaluate the usability of interactive software applications with the support of automatic tools, empirical data, and the task model of the application.

Keywords: Usability engineering; Automatic tools for usability evaluation; Model-based design and evaluation

1. Introduction

While the importance of usability engineering (i.e. the usability evaluation of interactive applications based on the use of structured methods) is generally recognised, there is still a lack of what Nielsen called computer-aided usability engineering (Nielsen, 1993). Indeed, despite the many tasks in the usability-engineering life cycle that could be performed more efficiently with computerised tools, a limited number of such automatic tools are available for this purpose and these usually provide rather limited support. More precisely, while work on providing some tool support for measuring the interactive use of applications has been ongoing for years (see for example Olsen and Halversen, 1988), less attention has been paid on how to use such an automatic support to identify more precisely errors in user interactions and, consequently, the problematic parts of a user interface.

To decide what automatic support should be provided, in our method, we started from the assumption that both empirical data and models relevant to designing interactive systems (such as task models) can give useful information in usability evaluations. In

* Corresponding author. Tel.: +39-50-3153066; fax: +39-50-3138091.
E-mail address: f.paterno@cnuce.cnr.it (F. Paternò).

0953-5438/00/$ - see front matter
PII: S0953-5438(00)00039-4

the former case it is possible to gather information concerning the real use of an application, whereas in the latter case task or user models provide meaningful support to evaluate both the specification of a user interface design and the concrete interactive software artefact. However, there are few methods able to integrate the relevant information gathered by these two types of approaches.

In addition, while the importance of analysing the actual performance of real users is generally recognised, usability testing is not widely used because it is highly time-consuming and expensive to bring users and evaluators together for a usability experiment and to perform analysis of empirical data, which is mainly manual.

A solution might be enabled by the rapidly growing number of Internet connections in most working (and non-working) environments. By remotely gathering user performance data and analysing it with the support of task models, we aim to furnish usability evaluators with a powerful tool and thereby improve the efficiency of usability testing.

In this paper we present our solution, a method called RemUSINE, that is supported by an associated automatic tool and requires the development of the task model corresponding to the application considered. This work is based on previous experiences in this area (Lecerof and Paternò, 1998) and adds a substantial contribution with respect to it because:

- it now supports the possibility of remote usability evaluation;
- it provides a new method, and a new related automatic tool, able to algorithmically analyse large amounts of usability data and then give a richer set of information, including an analysis of several sessions (an introduction to these new possibilities was given in Paternò and Ballardin (1999);
- it reports on experiences developed in the application of RemUSINE to a real case study in an industrial software development environment.

In the paper, we first discuss the previous work in the usability evaluation area to indicate where our contribution fits in and how it provides original solutions. Next, we introduce the architecture of the environment associated with RemUSINE. We describe its main phases and then provide examples of results taken from a real case study, and finally discuss the advantages of our method and give some concluding remarks and indications for further work.

2. Related works

Usability engineering (Nielsen, 1993) concerns the development of systematic methods to support usability evaluation. Various types of approaches have been proposed for this purpose.

Model-based approaches to usability evaluation use models, usually task or user models, to support this evaluation. They often aim to produce *quantitative predictions* of how well users will be able to perform tasks with a proposed design. Usually, the designer starts with an initial task analysis and a proposed interface design. The designer then uses an engineering model (like GOMS; Card et al., 1983) to find the usability problems of the interface. While model-based evaluation is useful to highlight relevant

aspects in the evaluation, it does not include empirical information, and predictions in some cases can be disproved by the real user behaviour. Thus, it is important to find methods that allow designers to combine meaningful models and empirical information. An attempt in this direction is USAGE (Byrne et al., 1994) that provides a tool supporting a method where the user actions, required to execute an application action in UIDE, are analysed by the NGOMSL approach. However, this information is still limited in comparison with that contained in the logs of the user actions performed during work sessions. Recently, some work has been carried out to derive GOMS models from user log analyses (Hudson et al., 1999) whereas our purpose is to use the logs to analyse possible mismatches between the user behaviour and the application task model.

In *empirical testing*, the behaviour of real users is considered. It can be very expensive and can have some limitation. It requires long observations of users' behaviour. Often these observations are supported by video that can be annotated by some tool. Even observing videos of user behaviour, either in work places or in a usability laboratory, can take designers a lot of time (a complete analysis can take more than five times the duration of the video) and some relevant aspects can still be missed.

In *inspection-based techniques* to usability evaluation, designers analyse a user interface or its description. Several of these techniques, such as heuristic evaluation, cognitive walkthrough, and software guidelines, have been found useful but limited because they are dependent on the ability of the evaluator, require multiple evaluators, or miss some relevant problems (Jeffries et al., 1991).

In recent years there has been increased interest in remote usability evaluation (Hartson et al., 1996), where evaluators are separated in time and/or space from users. This approach has been introduced for many reasons:

- *The increasing availability and improvement of network connections*, all kind of work environments are going to be connected to Internet; this implies the use of software applications and the possibility of exchanging information on their use by remote connections.
- *The cost and the rigidity of laboratory-based usability evaluation*. A well-equipped usability laboratory costs a considerable amount of money. In addition, it requires users to move to the laboratory which can be time-consuming for the users, especially when extensive testing has to be performed, and in some cases users do not like to move or their time is particularly costly.
- *A need for decreasing costs of usability evaluation to make it more affordable*. Usability evaluation should be a fundamental issue in measuring the software quality because if users have problems, they are not likely to use it even if it is functionally correct. However, it is not yet in the current practise of most software companies and so such an introduction is likely to occur only if the use of cheap methods, not requiring expensive tools is proposed.

There are various approaches in remote usability evaluation. One interesting approach is instrumented or automated data collection for remote evaluation, where tools are used to collect and return a journal or log of the interactions performed by the user. These data are

analysed later on, for example using pattern recognition techniques, however the results obtained are often rather limited for the evaluation of an interactive application.

We think that task models can provide additional support for analysing such data. However, to this end it is necessary that task models are powerful, non-prescriptive, and flexible. This means they should be able to describe dynamic and concurrent activities with the possibility of interruptions or choices among them. To support this analysis, task models should be refined to indicate precisely how tasks should be performed in the application considered. In particular, for our analysis we need to develop the task model of the application, i.e. how the design of the application requires that tasks be performed, considering also the possibility that, in some cases, a goal can be reached in different ways.

3. The RemUSINE method

If we consider current approaches, briefly summarised in Section 1, we can notice a lack of methods that are able at the same time:

- To support the evaluation of many users without requiring a heavy involvement of designers;
- To gather information on the users' behaviour at their work place without expensive equipment;
- To apply expressive and flexible task models to the evaluation of logs of user events, thus linking model-based and empirical evaluations. Current automatic tools, such as ErgoLight (Harel, 1999), that support usability evaluation by task models, use simple notations to specify such models, thus still depending on the evaluator to identify problematic parts.

RemUSINE addresses all three of these issues. We aim to evaluate applications that can be used by many users located in different places. The users' behaviour can be detected using automatic logging tools that store in files all the user-generated events during the work sessions. Logging tools are able to get this information without disturbing users' work. This information is analysed with the support of the task model to identify deviations in users' behaviours and to understand whether they are motivated by usability problems. In order to perform the automatic analysis, RemUSINE requires the following input (Fig. 1):

- *The log files of the user interactions*. With a logging tool it is possible to automatically generate a file containing all the events performed by a user during a work session. One or more of these files can be used to create the log-task table.
- *The log-task association table*. This table creates an association between the physical events that are generated by the user while interacting with the application and the basic interaction tasks (the tasks that cannot be further decomposed in the task model and require only one action to be performed). This association is a key point in our method because through it we can use the task model to analyse the user behaviour.
- *The task model*, it is specified using the ConcurTaskTrees notation (Paternò, 1999) for specifying task models.

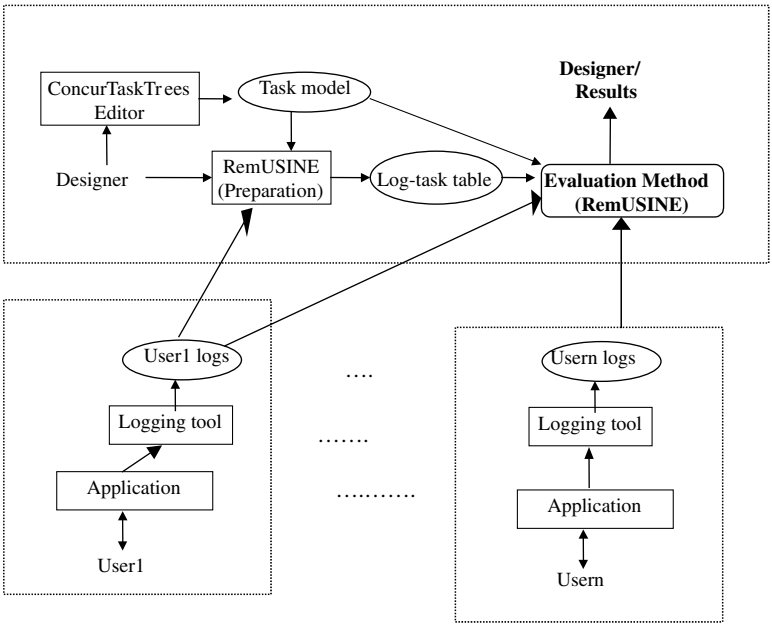

Fig. 1. The environment allowing evaluators to use RemUSINE.

In our method we can distinguish three phases:

- *Preparation phase*, that is mainly the development of the task model (there is an editor publicly available at http://giove.cnuce.cnr.it/ctte.html for this purpose) and the association between physical user-generated events, extracted by log files, and the basic interaction tasks of the task model;
- *The execution of the evaluation tool*, during which the tool first elaborates for each task the related precondition (if any) from the temporal relationships defined in the task model, and next uses this information to elaborate its results: the errors performed, task patterns, duration of the performance of the tasks and so on.
- *The analysis of the results of the evaluation tool*. In this phase, the designer can provide suggestions to improve the user interface by using the information generated by the RemUSINE tool.

By analysing the logs generated during users' sessions, RemUSINE is able to identify user errors. For example, it can detect attempts to activate an interaction that was not allowed because some precondition was not satisfied, or the user selected elements of the user interface that were not selectable.

A user action is considered an error if it is not useful to support the current task. One problem is how to automatically identify the tasks that the user intends to perform. To this end the knowledge of the actions performed by the user can be useful because, for example, if the user tries to submit an electronic form and s/he does not fill all the

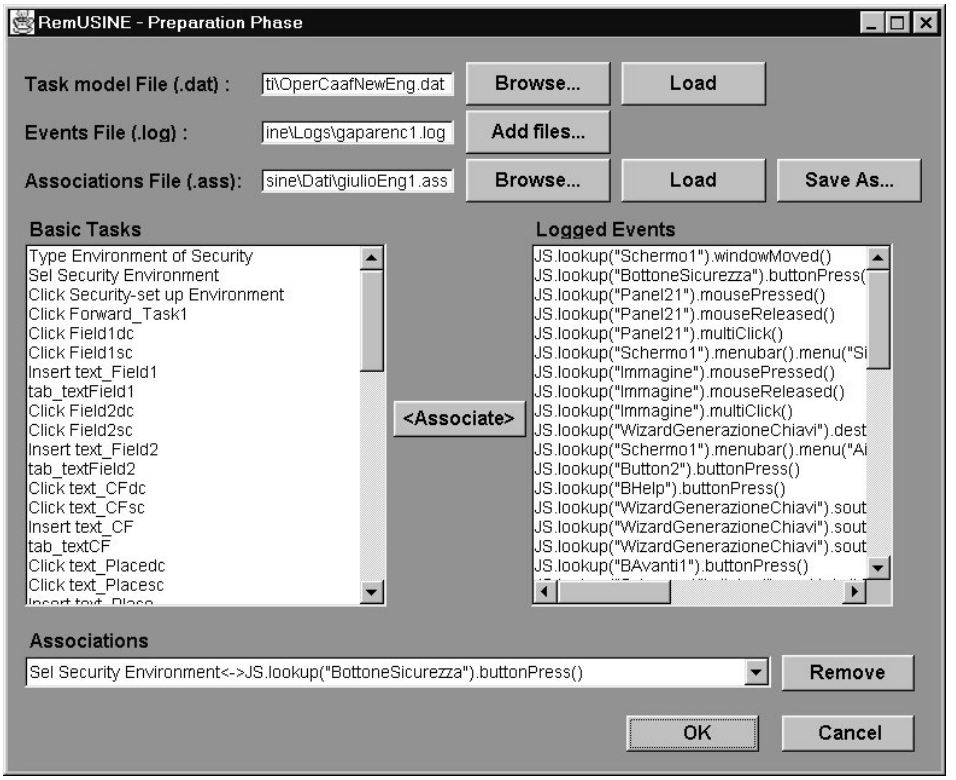

Fig. 2. The tool support for the preparation phase.

mandatory fields, it is possible to understand what the current user intention is (submitting a form) by detecting the related action, for example the selection of a Send button. Besides, a similar precondition error highlights that there is a problem with the user interface, as it probably does not highlight sufficiently what the mandatory fields of the form are.

4. The preparation phase

There are various tools available to automatically collect data on the user-generated events during an interactive session. For example, JavaStar (http://www.sun.com/ forte/ testingtools/javastar.html) is able to log interactions with Java applications or applets, and QCReplay (http://www.centerline.com/productline/qcreplay/qcreplay.html) can log interactions with applications written in various programming languages.

They provide files that indicate what events occurred and when they occurred. Typical events considered are mouse click, text input, and mouse movements. When user interface elements such as menu, pull-down menu are selected, they are able also to indicate what menu element was selected. The resulting files are editable text files. Similarly, using the

ConcurTaskTrees editor it is possible to save the task model specification in a file for further modifications and analysis.

In RemUSINE, there is a part of the tool that is dedicated to the preparation phase whose main purpose is to create the association between logs of user events and the basic interaction tasks of the task model. This association is then used to analyse the user interactions with the support of the task model.

In the preparation phase (as you can see in Fig. 2), the evaluator can load one log file and one file with the task model. The lists of elements contained in the two files appear in two different, parallel columns. In the task-related part only the basic interaction tasks appear, as they are the only elements that can be associated with logged events. While each basic task is indicated only once in the list, the number of times that one specific event appears in the related list depends on the session considered and the number of times the user performed it during such a session.

One basic task can be associated with one or more events. For example, selecting the next field of a graphical form in some application could be done by mouse selection, PageDown key or the down arrow key. Contrarywise, any given event is always associated with only one basic task. An event is identified by the type of action (such as mouse press, mouse release, multi-click) and the associated widget where it occurs.

The designer then has to select one physical event, and the corresponding basic task, and add this association to the table containing all of them by the related button. The list of basic tasks indicates only those tasks that still need to be associated with an event. The associations performed can be displayed by the *Associations* pull-down menu. In case of mistakes, the designer can remove elements from this list by the *Remove* button. The associations can be saved in a file and loaded later on for further expansions or for the evaluation phase.

This association needs to be made only once to evaluate as many user sessions with the considered application as desired. Indeed, the log/task table contains the information required by the evaluation tool by mapping all the possible interaction basic tasks with the corresponding user-generated events. Each session is associated with one sequence of events belonging to the set of input events that can be generated interacting with the interactive application considered. Thus, to evaluate a new session it is sufficient to provide the relative log file and, exploiting the log/task association table previously created, the tool can analyse such a session with the support of the task model. This is possible because, for each event in the log, the tool, by analysing the association table, can immediately indicate whether there is a task associated with it and, in the positive case, what task it is.

5. The execution of the evaluation tool

Our method can generate a wide variety of results that can be useful for the evaluator. There are two main types of information that RemUSINE provides:

- *Interactive analysis of a log of events*, it is possible to interactively execute the log of events with RemUSINE. For each event, the tool indicates what task is associated with

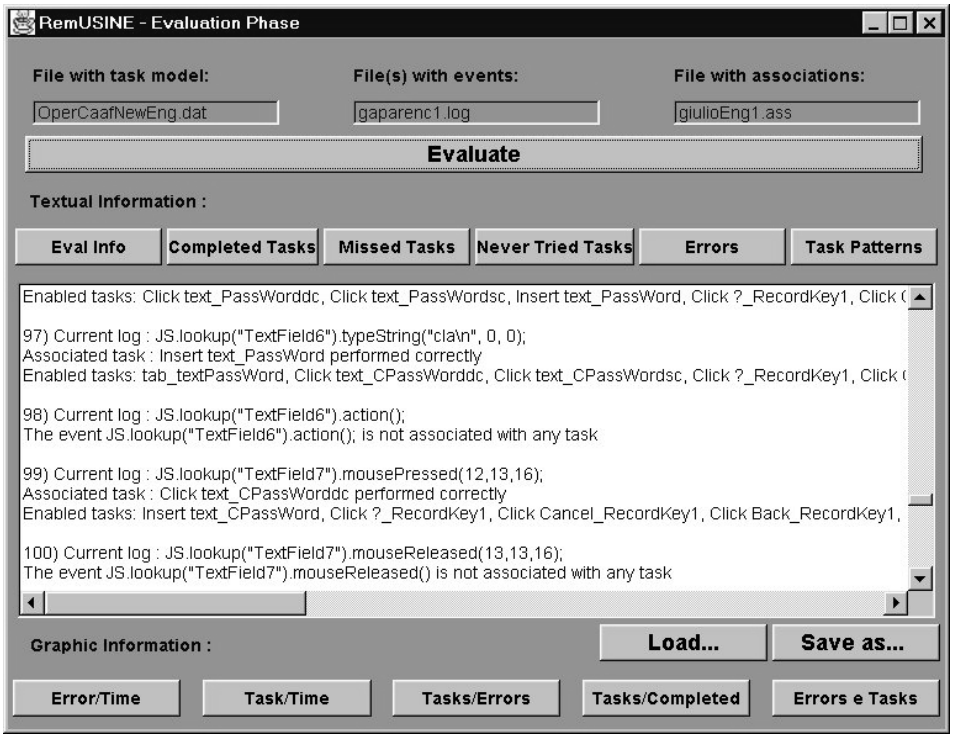

Fig. 3. An example of interactive analysis of a log.

it, what other tasks were available when they occurred, if the task associated had preconditions verified when the event occurred, and, in case such preconditions were not satisfied, what tasks had to be performed before in order to satisfy them (Fig. 3).

- *summary and statistical information on the user sessions*, such as duration, number of tasks failed and completed, number of errors, number of scrollbar or windows moved (see Fig. 4), more detailed information for the tasks considered, and some graphical representations of such results. When tasks are counted we consider all the tasks in the ConcurTaskTrees specification thus including both basic and high levels tasks.

The more detailed information about the tasks include:

- A display of the accomplished tasks and how many times they are performed.
- A display of the tasks the user tried to perform but failed because their preconditions were not satisfied, and how many times each task failed.
- A display of the tasks the user never tried to perform. This information can be useful to identify parts of the user interface that are either useless or difficult to achieve for the users; this result is more difficult to obtain with other approaches based on observations.
- A display of all the errors divided into precondition errors and others.
- A display of the task patterns found (specific sequences of tasks) among the

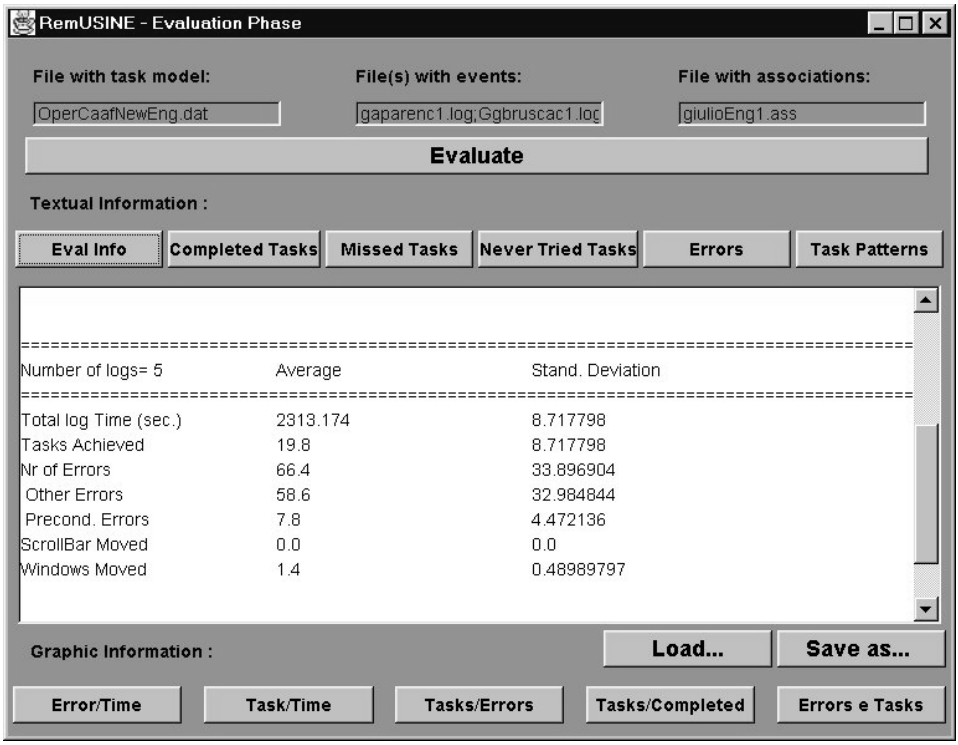

Fig. 4. An example of general information of a set of user sessions.

accomplished tasks (see Fig. 5). The presentation shows first the frequency and next the pattern, and orders them by frequency. Patterns are useful to identify sequence of tasks frequently performed by users. This information can be useful to try to improve the design so as to speed-up the performances of such sequences of tasks.

- A display of the entire result from the evaluation in temporal order. It is also possible to save this result in a file for loading it at a later moment.

The different graphs, showing the data from the evaluation in different manners, are:

- the *Tasks/Time* chart graph with the tasks on the *x*-scale and how long they took to perform on the *y*-scale (see Fig. 6). To make such a representation more readable, we split it into two parts: that related to basic tasks and that related to high level tasks. In case of analysis of a group of sessions, for each task the related bar chart highlights the fastest, the slowest and the average performance in the group of sessions considered.
- the *Errors/Time* graph with the number of errors on the *y*-scale and the time on the *x*-scale.
- the *Tasks/Errors* chart graph containing the number of precondition errors associated with each task.
- the *Tasks/Completed* chart graph containing the number of times the tasks were performed.
- the *Errors and Tasks* pie chart containing the different types of errors and their percentage,

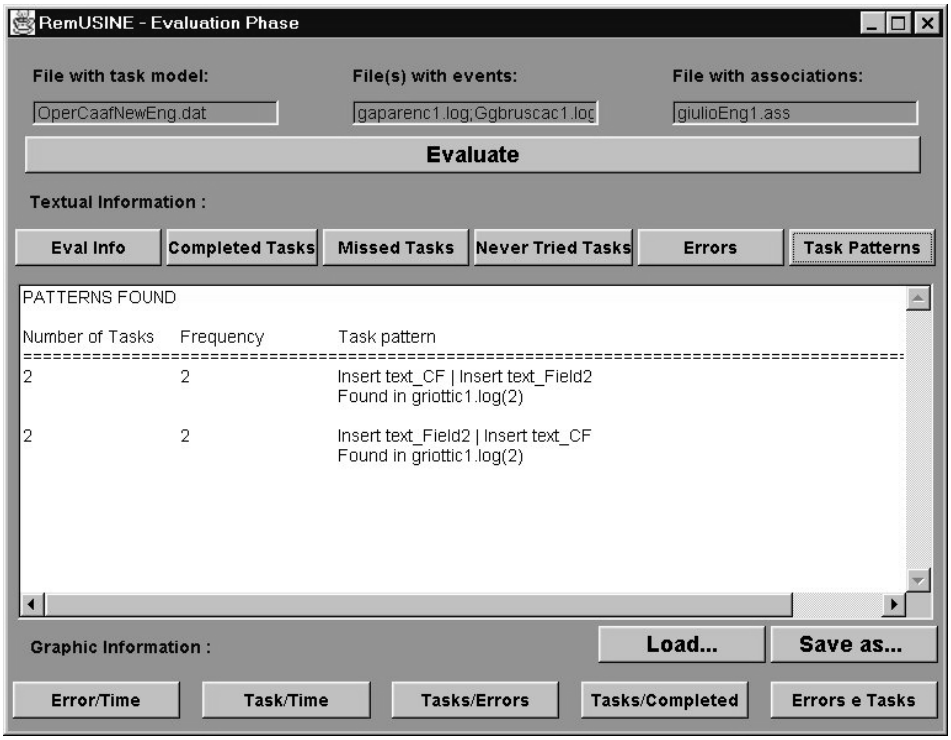

Fig. 5. An example of task patterns detected.

and another containing the number of tasks accomplished, missed and never attempted (see for example Fig. 7).

It is possible to provide all this information related to a single user session or to groups of user sessions. To apply the tool to groups of sessions, it is also useful to identify any abnormal behaviour occurring in a session. For example, a task that was performed over a long time just because the user was interrupted by external factors (such as answering a telephone call) during its accomplishment.

The tool allows evaluators also to identify situations where the user shows difficulties on how to progress in the interaction for example by selecting the on-line help.

6. The analysis of the tool results

Identifying errors in the analysis of the logs indicates a mismatch between the user behaviour and the task model associated with the application.

The reasons for this mismatch can be various: in some cases the user interface imposes illogical constraints, so it has to be changed in order to support a more flexible task model closer to that of the user. For example, the user interface imposes a sequential order

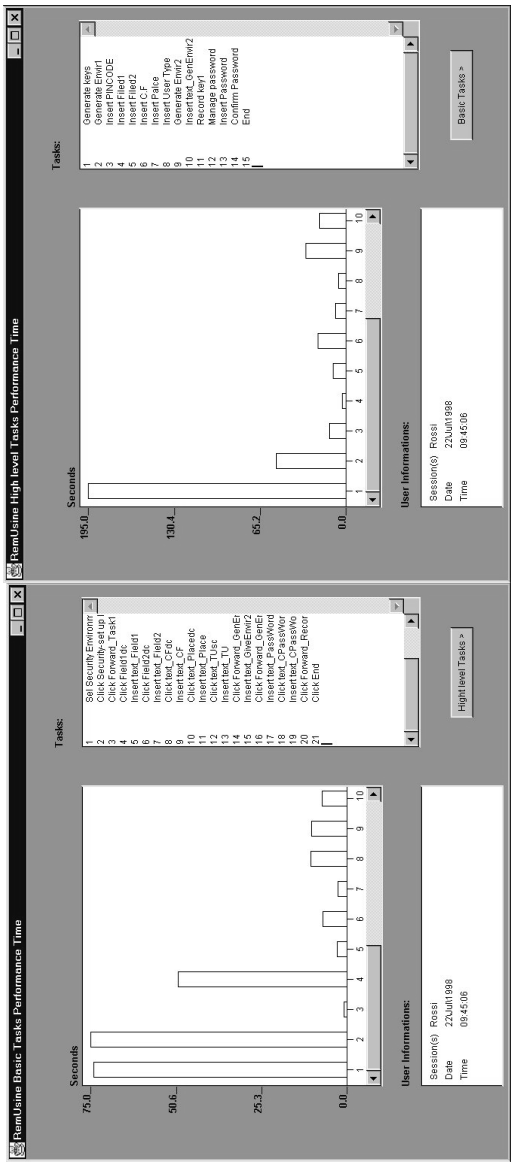

Fig. 6. A diagram indicating task performances.

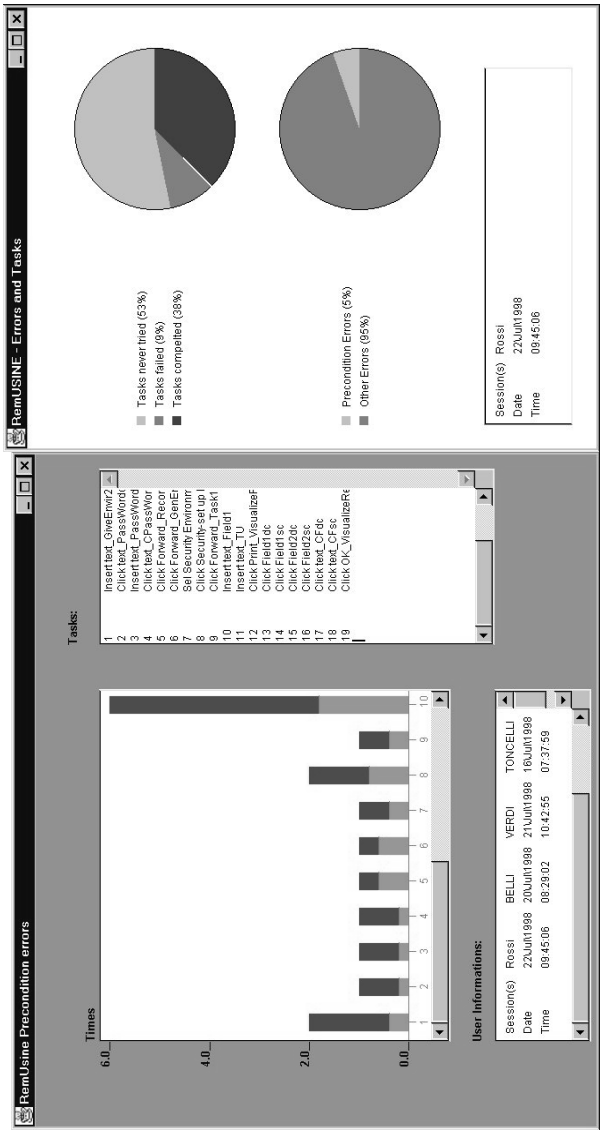

Fig. 7. Representation of errors performed in a session.

between two tasks, while they could be performed in any order because there is no logical dependency between their execution. In other cases, the task model associated with the interactive application describes the desired behaviour, but the design of the user interface is not effective and the user does not understand how to perform the desired task, thus changes have to be introduced. For example, using more explicative labels.

For each error identified, it is useful to prepare a short report structured in four fields:

- *The problem*: an indication of the problem highlighted by the error identified.
- *The identification*: a description of how the error has been identified.
- *The cognitive motivation*: a description of the possible cognitive problems that can have generated the problem.
- *The solution proposed*: an indication for improving the user interface design so as to avoid new occurrences of the problem detected.

The cognitive cause of the error can be identified by analysing the possible phases of a user interaction according to the Norman's model (Norman, 1988).

- *Intention*, the user intended to perform the wrong task. An example of intention problem is when the user tries to send an electronic form without filling all the mandatory fields. So, the intention was wrong according to the state of the application.
- *Action*, the task the user intended to perform was correct but the actions supporting it were wrong. An example of action error is when the user wants to answer positively to a question but instead of pressing the y key s/he selects the t key which is just beside.
- *Perception,* the user has difficulties in perceiving or correctly perceiving the information that is provided by the application. A perception problem is when the user takes a long time to find the user interface element necessary to perform the next task.
- *Interpretation*, the use misinterpreted the information provided by the application. An interpretation problem is when there is a More Info button but the user misunderstands for what topic more information is available.

7. The case study and its task model

In this section, we introduce a case study where we applied the RemUSINE approach to parts of an application developed by a software company. The purpose of this application is to provide a web interface to companies that must electronically register to a national centre to be authorised in their activities.

The final task model included 107 tasks structured into 8 levels with 69 basic tasks. We considered only a part of the application: that implemented in Java to support some user interactions. We did not evaluate the final application but an advanced prototype where some features were not completely included. We used JavaStar to log user events during their sessions.

The application supported various features: there was a Security Environment allowing a fast and safe registration by Internet, it supported exchange of documents and requests in a protected way, and some documentation on the application was available for the users.

The part of the application that we considered was mainly structured in a set of presentations linearly ordered with the possibility for the user to go backward and forward in such a linear order. At each presentation, the user had to provide some information. To help the user in the navigation a diagram representing all the phases, and highlighting the current phase in such a representation, was included in the various user interface presentations.

In the session tests all the users received the same goal to achieve: to register a company using some predefined data that were provided to them initially by the evaluator.

The first levels of the task model describe that the application first presents some general information, then asks users to choose what they want to access: the part of the application that supports set up of a secure access or other parts of the national centre web site or general documents. At any time, the session can be disabled by either a window manager command or a dedicated button. More detailed descriptions of parts of the task model of the application will be described in the next section.

8. Examples of usability problems found

To illustrate how our method works, we can consider some examples gathered from our case study. By these simple examples, we can show how our method works once it has calculated the preconditions for all the tasks. Given a task model specified in ConcurTaskTrees the tool automatically identifies the preconditions for each task. In this context, the preconditions of a task are those tasks that must be performed beforehand to allow its correct accomplishment.

We show the part of the log where an error was identified, then we discuss how the error occurred and was found by showing the corresponding user interface and the associated part of the task model. In the part of the log that we show we have removed useless information for our tool.

8.1. The user forgot to fill a mandatory field

As it is described in the task model relevant to the part of the application considered in this paragraph, users can handle some information concerning their enterprise. This task can be disabled ([> operator) when the user wants to cancel the editing performed and go

Table 1
Example of log

```
1   JS.lookup (“Code”).typeString(“501d636”, 0, 0);
2   JS.delay (55690);
3   JS.lookup (“Identify_Number”).multiClick(4,12,16,1);
4   JS.lookup (“Identify_Number “).typeString(“BPLL-001-HMF”, 0, 0);
5   JS.lookup (“Address”).multiClick(3,10,16,1);
6   JS.lookup (“Address”).typeString(“Street H. Smith, 111 London”, 0, 0);
7   JS.lookup (“User_Type”).multiClick(3,11,16,1);
8   JS.lookup (“User_Type “).typeString(“d10”, 0, 0);
9   JS.lookup (“Forward”).buttonPress();
10  JS.lookup (“Screen1”).dialog(“cli..”, “Error”).button(“OK”).buttonPress();
```

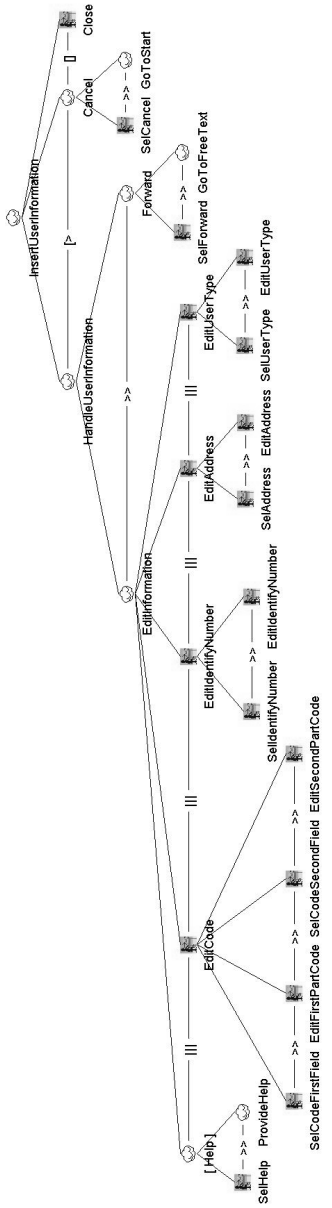

Fig. 8. A part of the task model.

Fig. 9. The layout of the first example.

back to the starting presentation or when s/he closes the session ([ ] is the operator indicating that a choice between two tasks has to be performed) (Fig. 8). Once the user has provided the requested information s/he can go forward in the application. Editing information means selecting some fields (code, identifier, address, user type) and then ($\gg$ is the sequential operator) providing the relative information. The editing of the various fields can be done in any order ($|||$ is the concurrent operator). The help task is optional (optional tasks have their name in squared brackets).

In the excerpt of log that we consider, the user had to fill all the fields of the "User

Table 2
Second example of log

```
1   JS.lookup ("Password").typeString("abc123\n", 0, 0);
2   JS.lookup ("Password").fkey(40,0); /* Down */
3   JS.lookup ("R_Password").mousePressed(4,9,16);
4   JS.lookup ("R_Password").mouseReleased(4,12,16);
5   JS.lookup ("R_Password").typeString("abc123", 0, 0);
6   JS.lookup ("Forward").buttonPress();
```

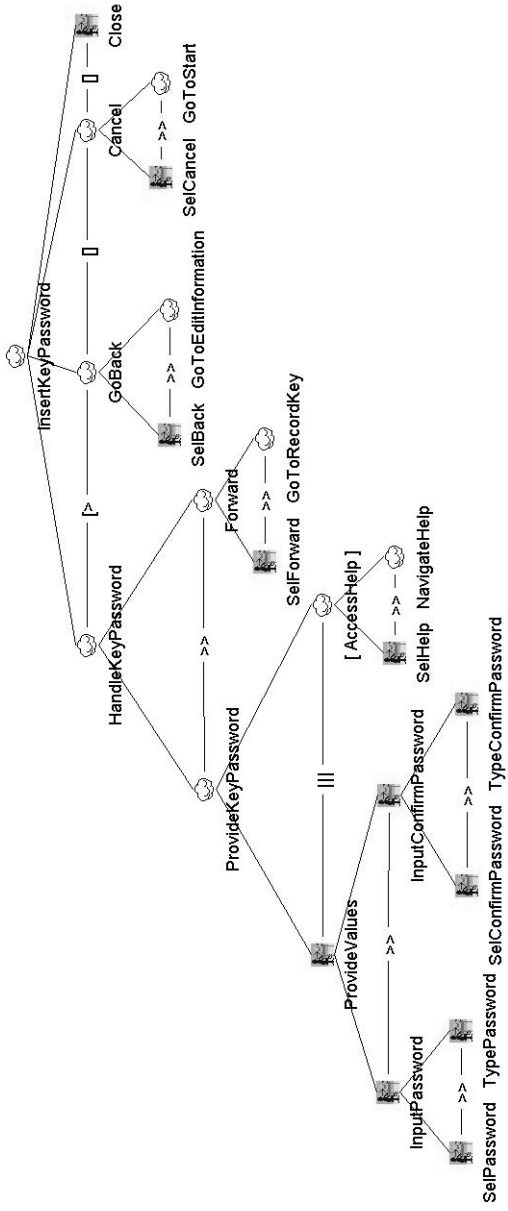

Fig. 10. The task model of the second part of the application considered.

Information" Frame (see Fig. 9). More precisely, the user types the first part of the code (action 1)(Table 1). Between each couple of user actions the logging tool provides information on the amount of time passed (action 2), we will not show the other similar information for sake of brevity. Then the user selects the Identify Number field (action 3) and provides the identity number (action 4). Similarly, the user selects address and user type fields and provides the relevant values (actions 5-6-7-8) and, finally, selects the forward button (action 9) and s/he gets an error message (action 10).

To summarise, the problem was the user did not fill the second field of the code, highlighted in Fig. 9, before moving to the next phase of the application. RemUSINE detected the error because the user selected the Forward button without first filling both the fields associated with the code as indicated in the task model. This was an interpretation problem. The user did not understand that both fields should have been filled. A possible solution is to add a label indicating that it is mandatory to fill both fields of the code. Another solution is to disable the Forward button until the user has provided all the mandatory information.

8.2. The user selects a wrong key to perform a task

In this second example (Fig. 10), the user had to fill both text fields in the frame "Key Password" (Fig. 11). S/he wrote (Table 2) his/her password "abc123" (action 1) in the field "Password", and then s/he pressed the ArrowDown key (action 2) to select the second field. But this was a wrong operation, so s/he had to select with the mouse the "R_Password" field (actions 3 and 4) and re-write the password "abc123" (action 5).

The user attempted to select the next field using the ArrowDown key. Thus in this session, s/he performed a wrong action to perform the SelConfirmPassword task. RemUSINE detected the error. The user intention was wrong because s/he thought s/he was allowed to use the key. Here there are two possible solutions to improve the user interface: either adding a label indicating that it is mandatory to select a field by mouse before filling it or allowing the use of additional keys such as ArrowDown.

8.3. The user selects non-interactive part of the user interface

In this example, the user starts the insertion of the user information. If we look at the log, in Table 3, we can note that then s/he starts to select the image on the left side of the frame to find some information (actions 1–6). But this is not possible. So, using the window manager command in the top-right side of the frame, s/he dismisses it (action 7), and asks help in the main menu (action 8).

In the log the Multiclick event is associated with double-clicking. In this example, the problem was that the user tried to interact with the left panel (see Figs. 9 and 11) that was not interactive. RemUSINE detected the problem because there were various events not associated with any task. This was a user interpretation problem. The user saw the area with the various phases and s/he probably thought that it could have been used to move to the next one. A solution is to add a label indicating that the panel is used only to give feedback on the state of the application and that the back and forward buttons should be used to move to different parts of the application.

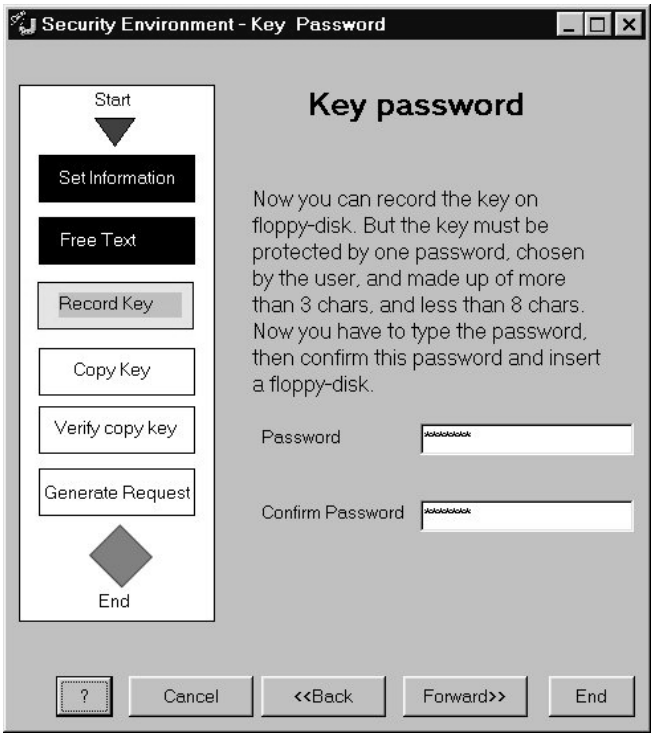

Fig. 11. The user interface of the second example.

Table 3
The third example of the log

1	JS.lookup (“Image”).mousePressed(65,82,16);
2	JS.lookup (“Image”).mouseReleased(69,81,16);
3	JS.lookup (“Image”).multiClick(45,78,16,1);
4	JS.lookup (“Image”).multiClick(45,78,16,1);
5	JS.lookup (“Image”).multiClick(45,78,16,1);
6	JS.lookup (“Image”).multiClick(45,78,16,1);JS.lookup(“Image”). MultiClick(45,78,16,1);
7	JS.lookup (“WizardUserInformation”).destroy();
8	JS.lookup (“Security”).menubar().menu(“Help”).item(“UsefulInformation”).action();

9. Comparing RemUSINE with empirical video-based evaluation

It can be interesting to compare our approach with empirical video-based evaluation. The software company where we performed our experiment also applied such a technique to analyse the sessions evaluated with RemUSINE. They had an equipped usability laboratory where they performed user testing. The sessions were video-recorded for later analysis using MUSIC (Bevan, 1995) tools.

The time required for applying our usability evaluation method (which will be indicated

as $\mathbf{t}_{\text{RemUSINE}}$) can be divided into five parts:

1. Record of logs using a logging tool such as *JavaStar* o *QCReplay*. It will be indicated as $\mathbf{t}_{\text{Logging}}$.
2. Developing of the task model, indicated as $\mathbf{t}_{\text{Build}}$.
3. Solution of problems in using RemUSINE, problems can be caused by incomplete task models, or incomplete association between logs and basic tasks, or other problems, $\mathbf{t}_{\text{Problems}}$.
4. Generation of the results $\mathbf{t}_{\text{Results}}$.
5. Analysis of the results provided by RemUSINE, indicated as $\mathbf{t}_{\text{Analysis}}$.

Consequently, we have that:

$$\mathbf{t}_{\text{RemUSINE}} = \mathbf{t}_{\text{Logging}} + \mathbf{t}_{\text{Build}} + \mathbf{t}_{\text{Problems}} + \mathbf{t}_{\text{Results}} + \mathbf{t}_{\text{Analysis}}$$

Now we can consider each component of this time. The $\mathbf{t}_{\text{Logging}}$ time is only the time required by the operator to perform the tasks, we do not require effort from the evaluator to supervise the sessions but only from the user and the test designer (if any). The users' sessions can be run in parallel whereas with video-based evaluation evaluators have to observe the users and so if only one evaluator is available then s/he has to run sequentially the tests.

The $\mathbf{t}_{\text{build}}$ depends on the designer/evaluator (often the same person has the two roles). It has to be spent only once to build the task model of the application considered. It is independent from the number of users that will be used in the test phase. It depends on the complexity of the application, the knowledge of the application that the person developing it has and the experience in task modelling of this person. In some cases, the task model can be built before the evaluation phase to support the design phase. In these cases, it does not require additional time. In our experience having the application available (we consider medium-large applications), the time required to develop 95% of the corresponding task models can vary from half a day to a few days. We say 95% of the model because often during the evaluation of the applications evaluators may discover that some small refinements are necessary.

It is difficult to give a quantitative indication of $\mathbf{t}_{\text{Problems}}$. In our experience, all the problems related to the preparation phase of RemUSINE are removed after analysis of 5-6 sessions that usually takes as long as the sessions themselves.

$\mathbf{t}_{\text{Results}}$ can be neglected as RemUSINE can give its results in a few minutes.

$\mathbf{t}_{\text{Analysis}}$ depends on the ability of the evaluator and is proportional to the number of users. On an average it has the same duration as the session because the tool helps in identifying the problematic parts. Thus, we can conclude that:

$$\mathbf{t}_{\text{RemUSINE}} \cong \mathbf{t}_{\text{build}} + \mathbf{t}_{\text{Problems}} + \mathbf{2} * \text{number of sessions} * \mathbf{t}_{\text{Logging}}$$

As we can see, only in the third factor's time increases with the number of users. Now we can compare the time required by our method and that required by video-based analysis. In Nielsen (1993), the time required for video-based analysis is between 3 and 10 times the session duration. We can indicate with $\mathbf{K}$ ($3 \leq \mathbf{K} \leq 10$) these factor. In our case study it was 5 times.

Indicating with $\mathbf{t}_{\mathrm{Reg}}$, the time required for the user session, we obtain the evaluation time is:

$$\mathbf{T}_{\mathrm{VideoAnalysis}} = \mathbf{K} * \mathbf{t}_{\mathrm{Reg}} * \text{ number of sessions}$$

Note that we do not report any fixed time to compare our method with the best case of the video-based method. Fig. 12 illustrates that with more than a certain number of user sessions, our method is better in terms of time.

Here we assume average session duration to be 30 min. We can assume that one complete session analysis requires 2.5 h, thus in one day (7.5 h) three sessions are analysed, and in one week (37.5 h) 15 sessions are analysed with video-based analysis. Consequently, 10 sessions would require 25 h.

If we consider the same sessions using RemUSINE, assuming two days to develop the task model (15 h) and 3 h to solve the problems, then 1 session analysis with RemUSINE requires 19 h, 3 sessions require 21 h, 10 sessions require 28 h and 15 sessions require 33 h. Thus, RemUSINE starts to be convenient, from a time point of view, after 12 sessions.

However, the time requested is not the only parameter to take into account when comparing evaluation methods. Different methods may find different problems. In our case, we found good overlap among the issues raised by the two methods. However, some user clicking was detected by RemUSINE but not in video-based analysis because the user movements in performing them were minimal.

In addition, the video-based analysis is also more expensive as it requires a usability laboratory with the relevant equipment and the commercial software for supporting analysis of videos.

10. Conclusions and future work

We have described our method to support usability evaluation with the associated automatic tool. We have seen that the final complete application is not required, because the method can be applied on prototypes of part of an application. The approach proposed requires the development of task models. The notation we use for this purpose allows us to overcome some limitations of task modelling (scalability and rigidity). The notation is tool-supported, thus making the process of development and analysis of task models easier. It also provides a rich set of operators to describe temporal relationships among tasks that allow designers to describe multiple sequences of tasks to reach the same goal or concurrent performance of multiple tasks. For sake of brevity, in the paper, we have included small excerpts from the case study considered. In Paternò and Mancini (1999), there is a description of how we developed the task model of an adaptable hypermedia containing museum information and supporting three user models.

We have found the method suitable for graphical applications with a well-defined set of tasks to support (examples are graphical editors, on line services for citizens or enterprises, user interfaces for cellular phones). In these cases, it provides a

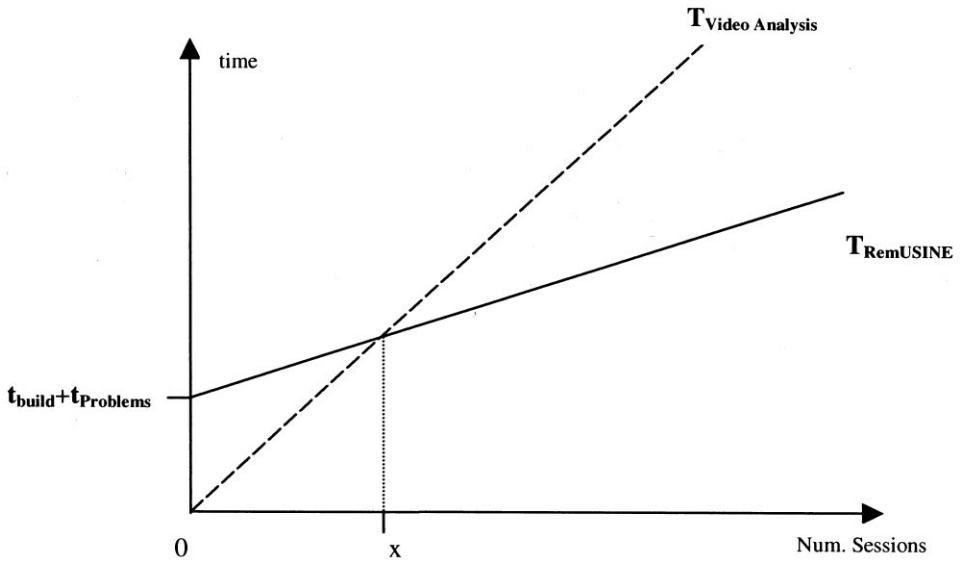

Fig. 12. Comparison of time requested in the two usability methods.

useful evaluation (especially for medium-large applications with dynamic dialogues and many users).

This method needs to be integrated with other techniques in applications where it is difficult to automatically identify the user's goals. Such difficulties may arise mainly for two reasons: the first is that the application has a flat structure, supporting any tasks at any time, even in concurrent ways, with low or no dependency among them. The second possibility occurs when modelling the tasks supported by the application is difficult (for example, in highly adaptive interfaces that have particularly complicated rules determining the changes in the set of tasks supported).

Future work will integrate screen dumps, automatically taken (not by cameras but directly from the screen) when some types of events occur, into the logs of events. This visual information can be helpful during the identification and analysis of the errors detected.

In our method, when we compute the time required by the user to perform tasks we do not distinguish between time spent by the system and that spent by the user either in internal cognitive activities or in interacting with applications. This was because the system time was very small in the application considered, since both client and server were running on the same host during the test phase. However, especially if our method is used to evaluate remote web applications where the time spent because of network delay can be considerable, then it becomes important to consider it. Thus, we plan to integrate techniques to identify the time spent in network (an example is in Fuller (1996)) or system delay in our method.

References

Bevan, N., 1995. Measuring usability as quality of use. Software Quality Journal 4, 115–130.

Byrne, M., Wood, S., Noi Sukaviriya, P., Foley, J., Kieras, D., 1994. Automating interface evaluation. Proceedings of CHI'94, pp. 232–237.

Card, S., Moran, T., Newell, A., 1983. The Psychology of Human–Computer Interaction, Lawrence Erlbaum, Hillsdale, NJ.

Fuller, R., 1996. Measuring User Motivation from Server Log Files, Designing for the Web: Empirical Studies, Microsoft Campus, 1996, available at http://www.micorsoft.com/usability/webconf/fuller/fuller.htm

Harel, A., 1999. Automatic operation logging and usability evaluation. Proceedings International, HCI, Munich, August 1999.

Hartson, R., Castillo, J., Kelso, J., Kamler, J., Neale, W., 1996. The network as an extension of the usability laboratory. Proceedings of CHI'96, pp. 228–235.

Hudson, S., John, B., Knudsen, K., Byrne, M., 1999. A tool for creating predictive performance models from user interface demonstrations. CHI Letters 1 (1).

Jeffries, R., Miller, J.R., Wharton, C., Uyeda, K.M., 1991. User interface evaluation in the real world: a comparison of four techniques, , Proceedings CHI'91 Conference, ACM Press (pp. 119–124).

Lecerof, A., Paternò, F., 1998. Automatic support for usability evaluation. IEEE Transactions on Software Engineering 24 (10), 863–888.

Nielsen, J., 1993. Usability Engineering, Academic Press, Boston.

Norman, D., 1988. The Psychology of Everyday Things, Basic Books, New York.

Olsen, D.R., Halversen, B.W., 1988. Interface usage measurements in a user interface management system. Proceedings UIST'88, pp. 102–108.

Paternò, F., Ballardin, G., 1999. Model-aided Remote Usability Evaluation, Proceedings INTERACT'99, IOS Press, Edinburgh (pp. 434–442).

Paternò, F., 1999. Model-based Design of Interactive Applications, Springer, Berlin (ISBN 185233-155-0).

Paternò, F., Mancini, C., 1999. Designing usable hypermedia, Empirical Software Engineering, vol. 4, Kluwer, Dordrecht (pp. 11–42).