Consiglio Nazionale delle Ricerche

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

Fabio Paternò, Giulio Ballardin

Technical Report
CNUCE - B4 – 1999 - 003

CNUCE
Pisa

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

F.Paternò, G.Ballardin
CNUCE-C.N.R.

Via S.Maria 36
Pisa, Italy

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

Keywords: Usability engineering, automatic tools for usability evaluation, model-based design and evaluation.

1. Introduction

While it is generally recognised the importance of usability engineering, i.e. the usability evaluation of interactive applications based on the use of structured methods, there is still a lack of what Nielsen called (Nielsen, 1993) computer-aided usability engineering. 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 usually they provide rather limited support. More precisely, work on providing some tool support for measuring interactive use of an application has been developed since various years ago (see for example Olsen and Halversen, 1988), however, 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 relevant models can give useful information in usability evaluation. In 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 either the specification of a user interface design or the concrete interactive software artefact. However, there is a lack of proposals able to integrate the relevant information gathered by these two types of approaches.

Besides, interesting possibilities are created by the continuos penetration of Internet connections in most working (and not only working) environments. This can have an impact also on how to perform usability evaluation and give a support for overcoming one common problem in empirical testing that is the cost required to move users or evaluators to allow them to meet for performing a usability evaluation experiment.

In this paper we present our solution to these issues which consists of a method, RemUSINE, that is supported by a relative 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 supports the possibility of remote usability evaluation (that was not considered in such previous experiences),
- it provides a new method, and a new related automatic tool, able to give richer set of information, including that concerning a compared analysis of a group of 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 a an industrial software development environment.

In the paper we first discuss the previous works developed in the usability evaluation area to indicate where our contribution can be located and how it provides an original contribution with respect to them, 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, finally, we 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.

M*odel-based approaches* to usability evaluation use some models, usually task or user models, to support this evaluation. They often aim (John and Kieras, 1996) 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 first interface design. The designer should then use an engineering model (like GOMS) to find the usability problems of the interface. While model-based evaluation is useful to highlight relevant aspects in the evaluation, it can be limiting not considering empirical information because the possible predictions in some cases can be denied by the real user behaviour. Thus, it is important to find methods that allow designers to apply meaningful models to some 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 with respect to that contained in the logs of the user actions performed during work sessions by users. Recently, some work has been developed with the purpose to derive GOMS models from analysis of user logs (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 it can have some limitations too. It requires long observations of users' behaviour. Often these observations are supported by video that can be annotated by some tool. Even observing video describing user behaviour, either in work places or in usability laboratory, can take a lot of time to designers (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 dependent on the ability of the evaluator or requiring multiple evaluators, or missing some relevant problems (Jeffries et al., 1991).

In the last years there has been an increasing interest in remote usability evaluation (Hartson et al., 1996). It has been defined as usability evaluation 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 and this implies the use of software applications and the possibility to exchange information on their use by remote connections;
- *the cost and the rigidity of traditional laboratory-based usability evaluation*, a well equipped usability laboratory costs a considerable amount of money for the equipment required and there is the problem that 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 than 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 industries and so such an introduction is likely to occur only if the use of cheap methods, not requiring expensive tools (such as eye-gazing 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 data containing indication of the interactions performed by the user. These data are analysed later on, for example using pattern recognition techniques, however usually the results obtained are 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 among them. To support this analysis, task models should be refined to indicate precisely how tasks should be performed following the design of the application considered.

3. The RemUSINE Method

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

- to give designers the possibility to support the evaluation of many users without requiring a heavy involvement of designers;
- to support the evaluation gathering information on the users' behaviour at their work place without using expensive equipment;
- to apply powerful and flexible task models in 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 requiring a strong effort from the evaluator to identify problematic parts.

These three relevant results are obtained by our RemUSINE method following an approach that is described in this paragraph.

We aim to evaluate applications that can be used by many users located in different places. The users' behaviour can be detected by using automatic logging tools that are able to store in files all the user-generated events during the work sessions. Logging tools are able to get this information without disturbing users during their 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:

- *the log files with the user interactions*, by the support of a logging tool it is possible to automatically generate a file storing all the events performed by a user during a work session. One or more of these files have an additional use that is the creation of the log-task table;
- *the log-task association table*, the purpose of this table is to create an association between the physical events, that are generated by the user while interacting with the application considered, 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.

In our method we can distinguish three phases:

- *the preparation part*, 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.

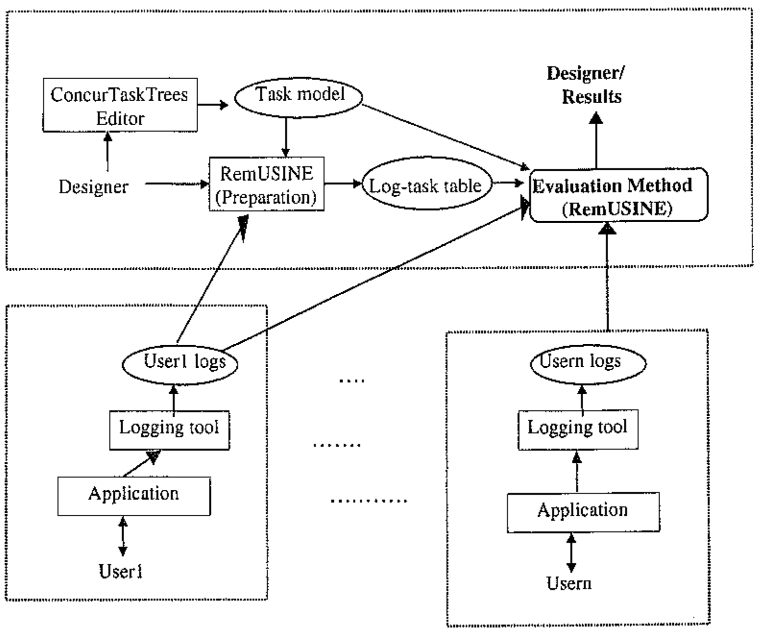


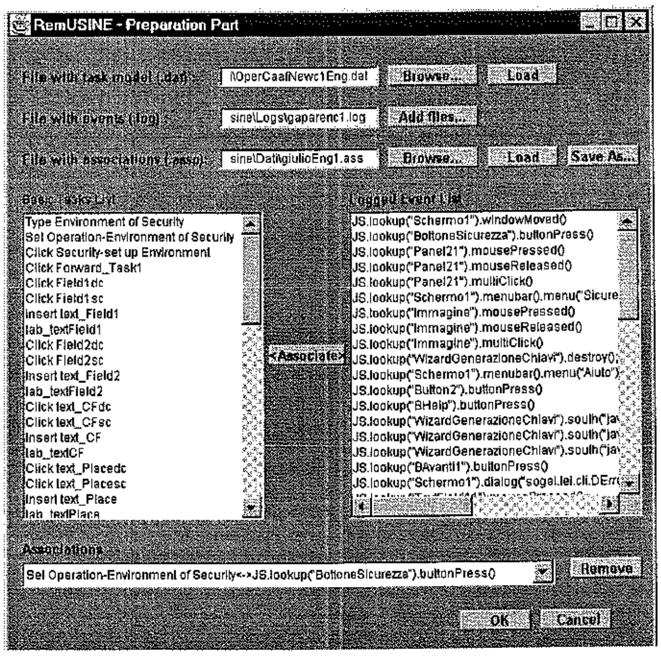
Figure 1: The environment allowing evaluators to use RemUSINE.

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 identify automatically 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 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 part

There are various tools available to collect automatically data on the user-generated events during an interactive session, for example, JavaStar (http://www.sun.com/suntest/JavaStar/JavaStar.html) or QCReplay (http://www.centerline.com/productline/qcreplay/qcreplay.html). They are able to provide files that indicate the events occurred and when they occurred. The events considered are mouse click, text input, mouse movements, and similar. When interaction techniques 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.



File with task model (.dat): \OperCaalNewc1Eng.dat

File with events (.log): sine\Logs\gaparenc1.log

File with associations (.asso): sine\Dati\giulioEng1.ass

Basic Tasks List

```
Type Environment of Security
Set Operation-Environment of Security
Click Security-set up Environment
Click Forward_Task1
Click Field1dc
Click Field1sc
Insert text_Field1
lab_textField1
Click Field2dc
Click Field2sc
Insert text_Field2
lab_textField2
Click text_CFdc
Click text_CFsc
Insert text_CF
lab_textCF
Click text_Placedc
Click text_Placesc
Insert text_Place
lab_textPlace
```

Logged Event List

```
JS.lookup("Schermo1").windowMoved()
JS.lookup("BottoneSicurezza").buttonPress()
JS.lookup("Panel21").mousePressed()
JS.lookup("Panel21").mouseReleased()
JS.lookup("Panel21").multiClick()
JS.lookup("Schermo1").menubar().menu("Sicure
JS.lookup("Immagine").mousePressed()
JS.lookup("Immagine").mouseReleased()
JS.lookup("Immagine").multiClick()
JS.lookup("WizardGenerazioneChiavi").destroy();
JS.lookup("Schermo1").menubar().menu("Aiuto")
JS.lookup("Button2").buttonPress()
JS.lookup("BHelp").buttonPress()
JS.lookup("WizardGenerazioneChiavi").south("ja
JS.lookup("WizardGenerazioneChiavi").south("ja
JS.lookup("WizardGenerazioneChiavi").south("ja
JS.lookup("BAvanti1").buttonPress()
JS.lookup("Schermo1").dialog("sogei.lei.cli.DErr
```

Associations

Set Operation-Environment of Security<->JS.lookup("BottoneSicurezza").buttonPress()

Figure 2: The tool support for the preparation part.

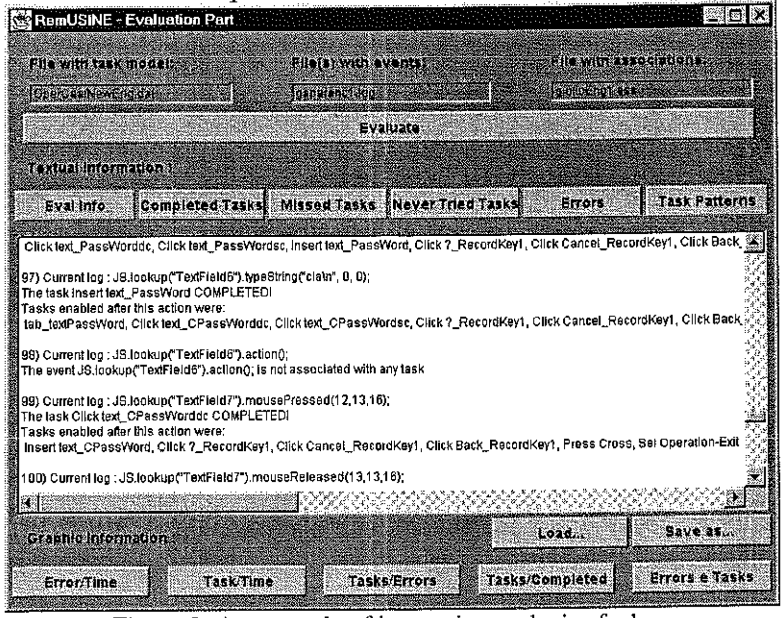
In the preparation phase (as you can see in Figure 2) the evaluator can load one log file and one file with the task model. The lists of elements contained in the two files appear in two different, parallel columns. In the task-related part only the basic interaction tasks appear, as they are the only elements that can be associated with logged events. While in the list of basic tasks each of them is indicated only once, the number of times that one specific event can appear on the related list depends on the session considered and the number of times the user performed it during such a session.
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. Once a basic task has been associated with the relative event, it will disappear from the list of tasks that thus indicates only those tasks that still need to be associated with the relative 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 can 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 trace 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:

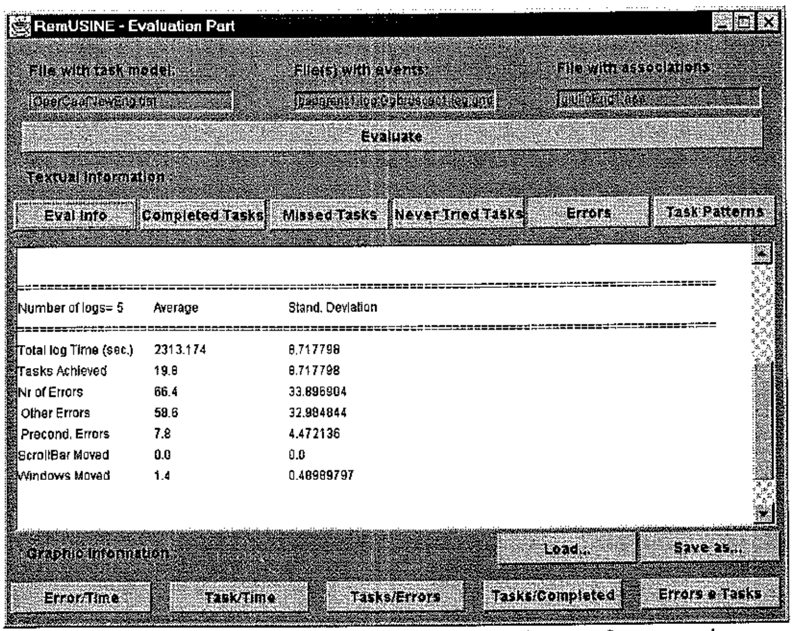
- *interactive analysis of a log of events*, it is possible to interactively execute the log of events with RemUSINE. For each event the tool is able to indicate what task is associated with it, what other tasks were available when it occurred, if the task associated had preconditions verified when the event occurred, and, in case such preconditions were not satisfied, what tasks had to be performed before in order to satisfy them.



```
Click text_PassWorddc, Click text_PassWordsc, Insert text_PassWord, Click ?_RecordKey1 , Click Cancel_RecordKey1 , Click Back_

97) Current log : JS.lookup("TextField6").typeString("cla\n", 0, 0);
The task Insert text_PassWord COMPLETED!
Tasks enabled after this action were:
 tab_textPassWord, Click text_CPassWorddc, Click text_CPassWordsc, Click ?_RecordKey1, Click Cancel_RecordKey1, Click Back

98) Current log : JS.lookup("TextField6").action();
The event JS.lookup("TextField6").action(); is not associated with any task

99) Current log : JS.lookup("TextField7").mousePressed(12,13,16);
The task Click text_CPassWorddc COMPLETED!
Tasks enabled after this action were:
 Insert text_CPassWord, Click ?_RecordKey1, Click Cancel_RecordKey1 , Click Back_RecordKey1, Press Cross, Set Operation-Exit

100) Current log : JS.lookup("TextField7").mouseReleased(13,13,16);
```

Figure 3: An example of interactive analysis of a log.

- *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 Figure 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.

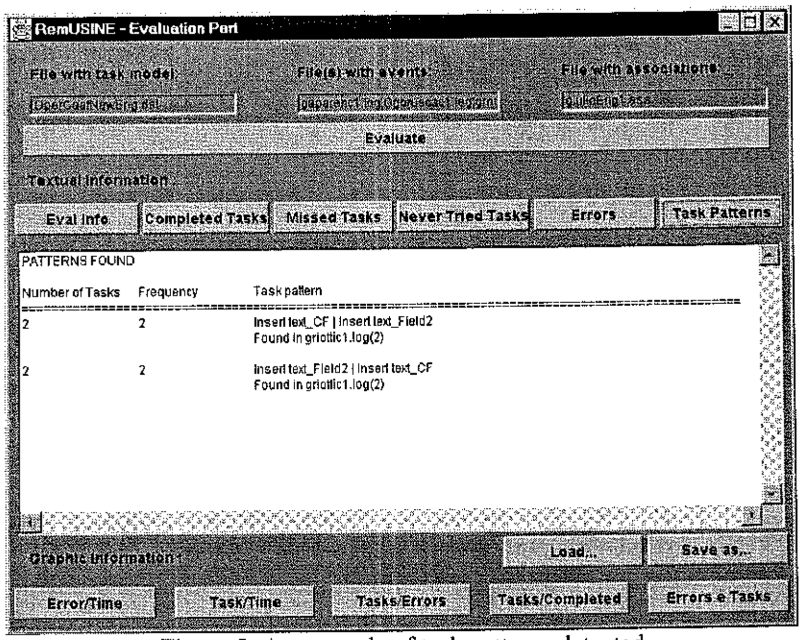


```
==========================================================================
Number of logs= 5     Average          Stand. Deviation
==========================================================================
Total log Time (sec.)  2313.174        8.717798
Tasks Achieved         19.8            8.717798
Nr of Errors           66.4            33.896904
 Other Errors          58.6            32.984844
 Precond. Errors       7.8             4.472136
ScrollBar Moved        0.0             0.0
Windows Moved          1.4             0.48989797
```

Figure 4: An example of general information of a set of user sessions.

The more detailed information about the tasks include:

- the display of the accomplished tasks and how many times they are performed. The display of the tasks the user tried to perform but failed because their preconditions were not satisfied, and how many times each task failed.
- the 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.
- display of all the errors divided into precondition errors and others.
- the display of the task patterns found (specific sequences of tasks) among the accomplished tasks (see Figure 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
- the display of the entire result from the evaluation in temporal order. It is also possible to save this result in a file and load at a later moment.

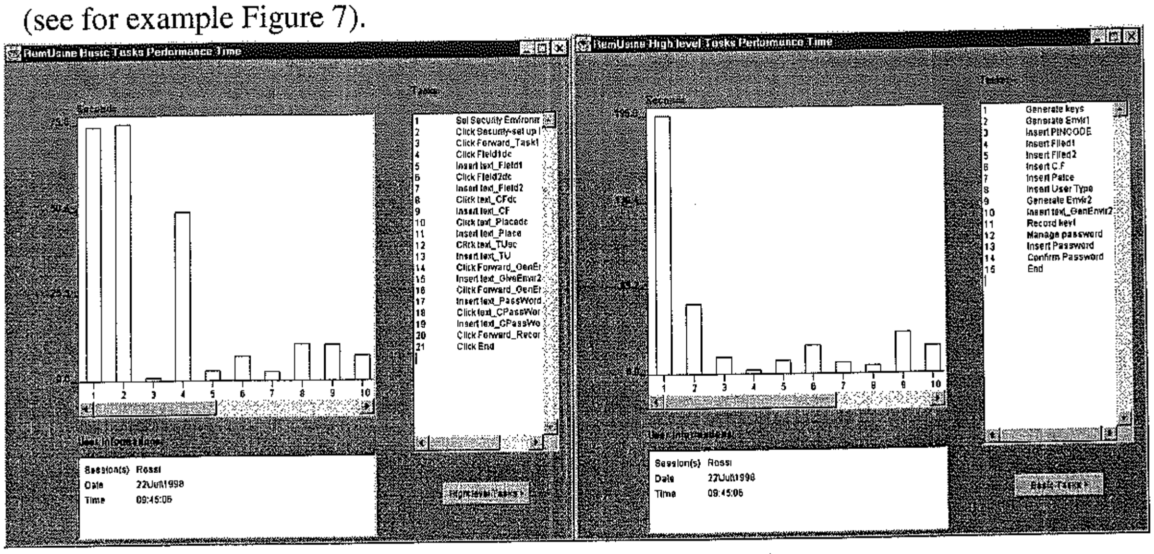


```
PATTERNS FOUND

Number of Tasks   Frequency   Task pattern
================================================================
2                 2           Insert text_CF | Insert text_Field2
                              Found in griottic1.log(2)

2                 2           Insert text_Field2 | Insert text_CF
                              Found in griottic1.log(2)
```

Figure 5: An example of task patterns detected

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 Figure 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 & Tasks* pie chart containing the different types of errors and their percentage, and another containing the number of tasks accomplished, missed and never attempted (see for example Figure 7).


RemUsine Basic Tasks Performance Time
RemUsine High level Tasks Performance Time
Seconds
Tasks
1
2
3
4
5
6
7
8
9
10
High level Tasks
Basic Tasks

```
1    Sel Security Environm
2    Click Security-set up
3    Click Forward_Task1
4    Click Field1dc
5    Insert text_Field1
6    Click Field2dc
7    Insert text_Field2
8    Click text_CFdc
9    Insert text_CF
10   Click text_Placedc
11   Insert text_Place
12   Click text_TUdc
13   Insert text_TU
14   Click Forward_GenEr
15   Insert text_GiveEnvir2
16   Click Forward_GenEr
17   Insert text_PassWord
18   Click text_CPassWor
19   Insert text_CPassWo
20   Click Forward_Recor
21   Click End
```

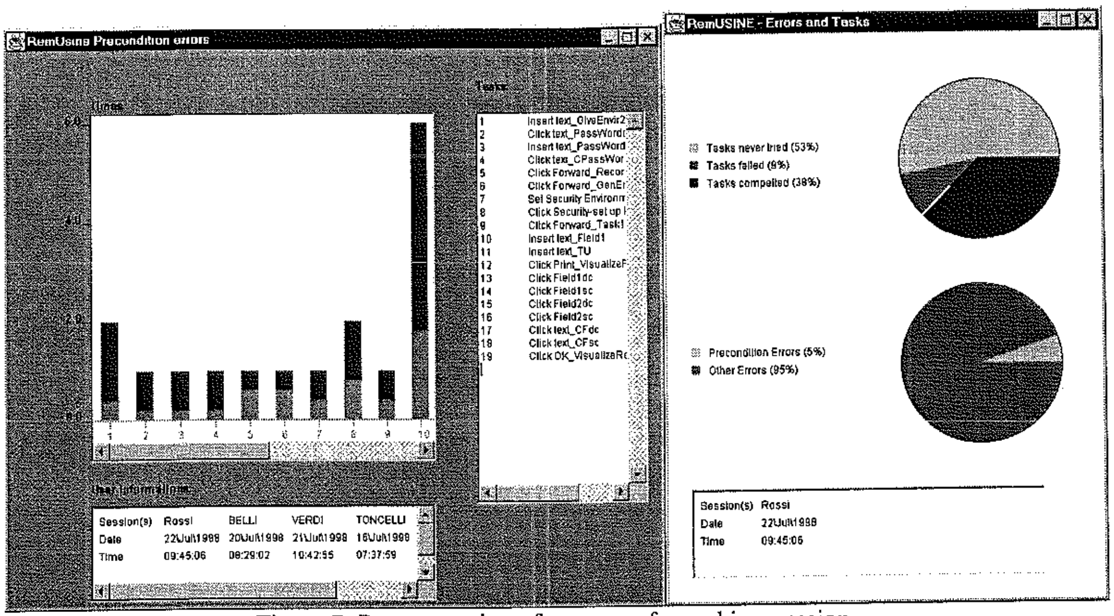
```
Session(s)   Rossi
Date         22Jul1998
Time         09:45:06
```

```
1    Generate keys
2    Generate Envir1
3    Insert PINCODE
4    Insert Filed1
5    Insert Filed2
6    Insert C.F
7    Insert Palce
8    Insert User Type
9    Generate Envir2
10   Insert text_GenEnvir2
11   Record key1
12   Manage password
13   Insert Password
14   Confirm Password
15   End
```

```
Session(s)   Rossi
Date         22Jul1998
Time         09:45:06
```

Figure 6: A diagram indicating task performances.

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 is useful also to identify if in any session there was some abnormal behaviour. For example, a task that was performed in a long time just because the user was interrupted by external factors during its accomplishment.



1	Insert text_GiveEnvir2
2	Click text_PassWord
3	Insert text_PassWord
4	Click text_CPassWor
5	Click Forward_Recor
6	Click Forward_GenEr
7	Set Security Environm
8	Click Security-set up
9	Click Forward_Task1
10	Insert text_Field1
11	Insert text_TU
12	Click Print_VisualizeF
13	Click Field1dc
14	Click Field1sc
15	Click Field2dc
16	Click Field2sc
17	Click text_CFdc
18	Click text_CFsc
19	Click OK_VisualizeRe

Session(s)	Rossi	BELLI	VERDI	TONCELLI
Date	22Jul1998	20Jul1998	21Jul1998	16Jul1998
Time	09:45:06	08:29:02	10:42:55	07:37:59

Session(s)	Rossi
Date	22Jul1998
Time	09:45:06

Figure 7: Representation of errors performed in a session.

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 constraints that are not motivated from a logical point of view so it has to be changed in order to support a more flexible task model closer to that of the user. In other cases the task model associated with the interactive application describes the desired behaviour but the design of the user interface is not sufficiently effective and the user has problems in understanding how to perform the desired task, thus changes have to be introduced, for example using labels more explicative.

We found useful for each error identified 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,
- *Action*, the task the user intended to perform was correct but the actions supporting it were wrong,
- *Perception*, the user perceived the wrong information,
- *Interpretation*, the use misinterpreted the information provided by the application.

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. 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. A perception problem is when the user selects an image which is not interactive whereas a comprehension 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 industry. The purpose of this application is to provide a web interface to companies that have to 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 having some predefined data for the company that were provided to them initially by the evaluator.

In the first levels of the task model relative to the application considered it is described that first the application presents some general information, next there is a choice among three possibilities: accessing the part of the application supporting 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 explain 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 is able to automatically identify the preconditions for each task. In this context the preconditions of a task are those tasks that have to 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 relative part of the task model. In the part of the log that we show we have removed the information useless for our tool.

8.1 The user forgot to fill a mandatory field

As it is described in the task model relative 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 back to the starting presentation or when s/he closes the session ([] is the choice operator). 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 (>> 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 bracktes).

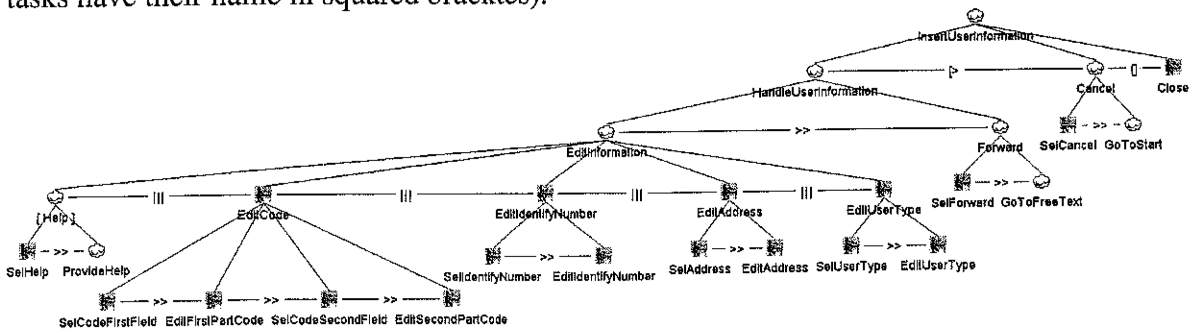
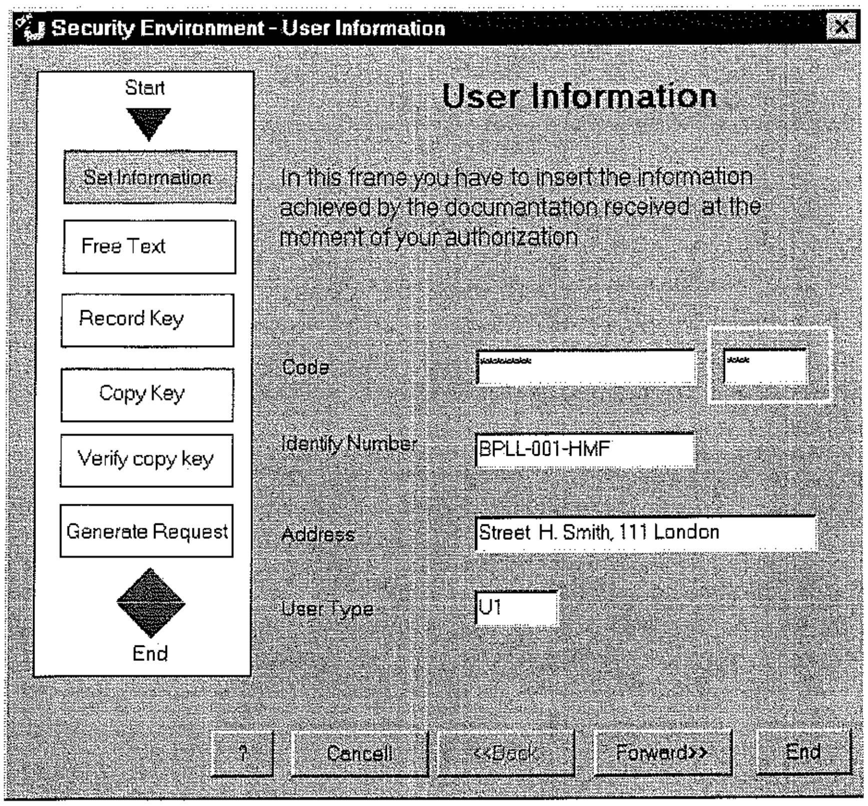


Figure 8: A part of the task model.

In the excerpt of log that we consider the user had to fill all the fields of "User Information" Frame (see figure 9). More precisely, the user types the first part of the code (action 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 identify number (action 4). Similarly then the user selects address and user type fields and provides the relative values (actions 5-6-7-8) and, finally, selects the forward button (action 9) and s/he gets an error message (action 10).

```
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();
```

Table 1: Example of log.



User Information

In this frame you have to insert the information achieved by the documantation received at the moment of your authorization

Code: ********** | ***
Identify Number: BPLL-001-HMF
Address: Street H. Smith, 111 London
User Type: U1

Figure 9: The layout of the example 1.

To summarise the problem was the user did not fill the second field of the code highlighted in Figure 9. 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.

8.2 The user selects a wrong key to perform a task

In this second example of excerpt of log, the user had to fill both text fields in the frame "Key Password". S/he wrote 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 (action 3 and 4) and re-write the password "abc123" (action 5).

```
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();
```

Table 2: Second example of log.

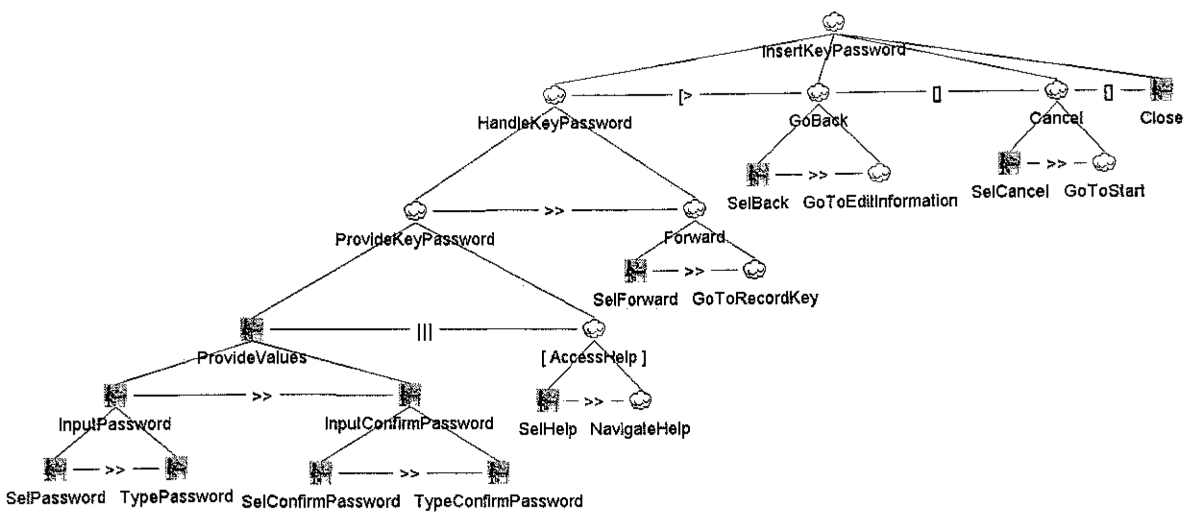
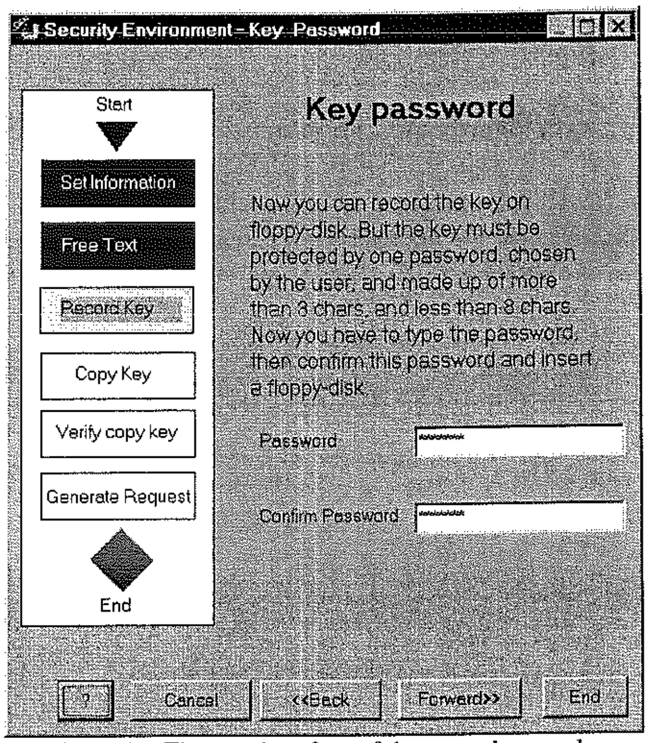


Figure 10: The task model of the second part of the application considered.

The user attempted to select the next field by 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 such 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.



Key password

Now you can record the key on floppy-disk. But the key must be protected by one password, chosen by the user, and made up of more than 3 chars, and less than 8 chars. Now you have to type the password, then confirm this password and insert a floppy-disk.

Password ********

Confirm Password ********

Figure 11: The user interface of the second example.

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 destroys it (action 7), and asks help in the main menu (action 8).

```
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();
```

Table 3: the third example of the log.

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 Figure 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.

9. Comparing RemUSINE with empirical video-based evaluation

It can be interesting to compare our approach with an approach often used such as empirical video-based evaluation. The software company where we performed our experiment applied also such a technique to analyse the sessions evaluated with RemUSINE. They had an equipped usability laboratory where performing 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 $\mathbf{t_{RemUSINE}}$) can be divided in five parts:

1. Record of logs using a logging tool such as *JavaStar* o *QCReplay*. It will be indicated as $\mathbf{t_{Logging}}$;
2. Developing of the task model, indicated as $\mathbf{t_{Build}}$;
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, $\mathbf{t_{Problems}}$;
4. Generation of the results $\mathbf{t_{Results}}$;
5. Analysis of the results provided by RemUSINE, indicated as $\mathbf{t_{Analysis}}$.

Consequently we have that:

$$\mathbf{t_{RemUSINE} = t_{Logging} + t_{Build} + t_{Problems} + t_{Results} + t_{Analysis}}$$

Now we can consider each of this time. The $\mathbf{t_{Logging}}$ time is only the time required by the operator to perform the tasks, it does not require effort from the evaluator 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 t_{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 has the person developing it and the experience in task modelling of such a 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 new additional time. In our experience having the application available (we consider medium-large applications) the time requested for developing 95% of the corresponding task models can vary between half day or a week. 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 $t_{Problems}$. In our experience all the problems are removed after an analysis of 2-4 sessions that lasts as long as the 2-4 sessions.
$t_{Results}$ can be neglected as RemUSINE can give its results in a few minutes.
$t_{Analysis}$ depends on the ability of the evaluator and it is proportional to the number of users. In average it has the same duration than the session because the tool helps in identifying the problematic parts.
Thus, we can conclude that:
$t_{RemUSINE} \cong t_{build} + t_{Problems} + t_{Analysis}$

As we can see the total time depends only in a limited way on the number of users, only for the third factor.
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 **K** ($3 \leq K \leq 10$) these factor. In our case study it was 5 times.
Indicating with T_{Reg} the time required for the evaluation we obtain the evaluation time is:
$T_{VideoAnalysis} = K * t_{Logging} *$ **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. Figure 12 gives a qualitative description that there is a certain number of user sessions after that our method is better in terms of time.

If we assume to analyse session whose average duration is 30 minutes. We can assume that one complete session analysis requires 2.5 hour, thus in one day (7.5 hours) three sessions are analysed, and in one week (37.5 hours) 15 sessions are analysed with video based analysis. Consequently, 10 sessions would require 25 hours.
If we consider the same sessions, assuming three days to develop the task model (22.5 hours) and 2 hours to solve the problems then we would have that 1 session analysis with RemUSINE requires 25 hours, 3 sessions require 26 hours, 10 sessions require 29.5 hours and 15 sessions require 32 hours. Thus, RemUSINE starts to be convenient, from a time point of view, after 10-15 sessions to analyse.

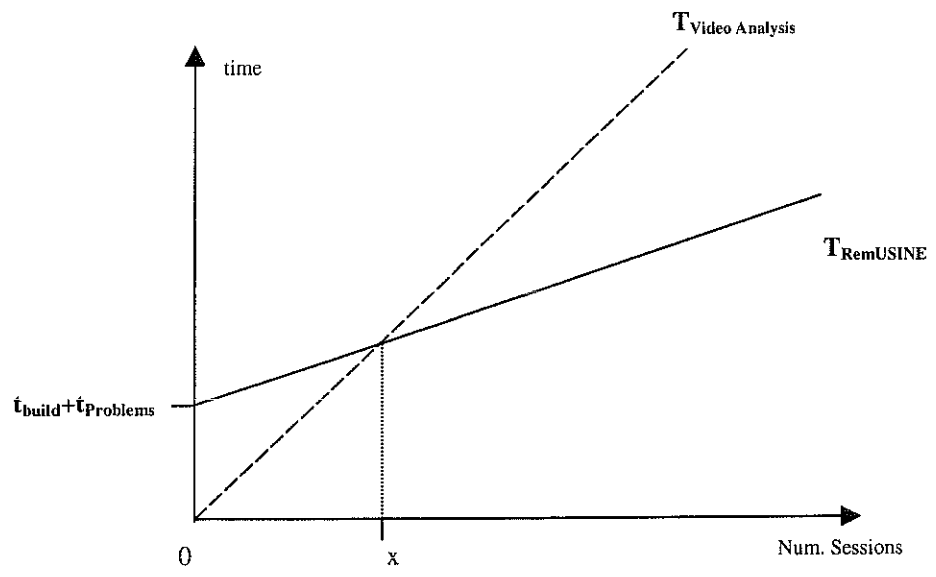


Figure 12: comparison of time requested in the two usability methods.

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

Besides, the video-based analysis is also more expensive in terms of money as it requires a usability laboratory with the relative 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 relative automatic tool. We have seen that it is not required to have the final complete application because the method can be applied on prototypes of part of an application.
Future work will be dedicated to integrate screen dumps, automatically taken (not by cameras but directly from the screen) when some types of events occur, with 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 requested to 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 as both client and server were running on the same host during the test phase and also the system reaction was very fast. 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 and Rodney, 1996) or system delay in our method.

REFERENCES

Badre A. N., Guzdial M., Hudson S. E. and Santos P. J., "A user interface evaluation environment using synchronized video, visualizations and event trace data", Software Quality Journal 4, 101-113 (1995).

Bevan N. (1995). Measuring usability as quality of use. *Software Quality Journal 4*, pp 115-130. Chapman & Hall.

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, N.J., 1983

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

Harel, A., (1999), Automatic Operation Logging and Usability Evaluation, Proceedings HCI International, Munich, August 1999.

Hartson, R., Gray, P., (1992) "Temporal Aspects of Tasks in the User Action Notation", Human Computer Interaction, Vol.7, pp.1-45.

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

Scott Hudson, Bonnie John, Keith Knudsen, Michael Byrne, (1999) "A Tool for Creating Predictive Performance Models from User Interface Demonstrations", to appear in CHI Letters, v1, n1, November 1999.

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

John, B., Kieras, D., (1996) Using GOMS for User Interface Design and Evaluation: Which Technique?, ACM Transactions on Computer-Human Interaction, Vol.3, N.4, December 1996, pp.287-319.

Lecerof A., Paternò F., (1998) Automatic Support for Usability Evaluation, IEEE Transactions on Software Engineering, Vol.24, N.10, October'98.

Nielsen J., (1993) *Usability Engineering*. Boston: Academic Press, 1993.

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. (1999) Model-Based Design of Interactive Applications, Springer Verlag, 1999.

Paternò F., Ballardin G., (1999) Model-Aided Remote Usability Evaluation, Proceedings INTERACT'99, Edinburgh, September 1999.

Wilson S., Johnson P., Kelly C., Cunningham J. and Markopoulos P. (1993). Beyond Hacking: A Model-based Approach to User Interface Design. *Proceedings HCI'93*. In: *People and Computers VIII, Proc. of HCI'93 Conf.*, Cambridge: CUP.