ELSEVIER

Interacting with Computers 12 (2000) 281–311

Interacting with Computers

www.elsevier.com/locate/intcom

Integrated support for working with guidelines: the Sherlock guideline management system

D. Grammenos, D. Akoumianakis, C. Stephanidis*

Institute of Computer Science, Foundation for Research and Technology—Hellas Science and Technology Park of Crete, GR-71110 Heraklion, Crete, Greece

Abstract

For a number of years, the primary medium for propagating human factors input to interactive system development has been paper-based guideline reference manuals. However, in the recent past, a number of tools for working with guidelines have emerged to ease the tasks of: (i) accessing and retrieving guidelines, (ii) applying recommendations to design prototypes, and (iii) facilitating a more effective human factors input to the early stages of system development. This paper presents a new way for working with guidelines and discusses the functionality, properties, typical use and evaluation of a supporting tool environment, the Sherlock Guideline Management System. Sherlock builds upon and extends the results of previous efforts to address state of the art requirements and problems, as highlighted by recent practice and experience in the use of the current generation of guideline management systems. In particular, Sherlock provides an integrated environment for articulating and depositing guidelines, accessing past experience, propagating guidelines/recommendations to the user interface development life-cycle, and facilitating the automatic usability inspection of tentative design. Thus, Sherlock fosters persistency of organisational knowledge on guidelines and evolution of the accumulated design wisdom. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Guideline management system (GMS); Tools for working with guidelines; Automatic usability inspection; User interface design support

1. Introduction and background

Guidelines constitute a popular means for integrating human factors input into the life-cycle of interactive computer-based products and services. In the mainstream human computer interaction (HCI) field, guidelines have been in use for a number of years and have contributed substantially to the development of a human factors culture within

* Corresponding author. Tel.: + 30-81-391741; fax: + 30-81-391740.
E-mail address: cs@ics.forth.gr (C. Stephanidis)

commercial organisations and institutions. The term guideline, in the present context, entails all forms of abstract or concrete recommendations that may be used to design interactive software. Such guidelines may be expressed: as general and domain independent recommendations [1]; as platform-specific style guides [2–5]; or as experience based usability heuristics [6,7]. Guidelines are typically documented in reference manuals, standards or may be part of the corporate culture and practice of an organisation (i.e. customised corporate design wisdom).

The primary use of guidelines is during the early phase of design of an interactive system and provide a means of utilising the accumulated design experience, as consolidated by the organisation, or as available in the general literature (i.e. software ergonomics, human factors, etc). As such, guidelines offer guidance towards a desired end, as opposed to a pre-packaged or fixed solution. Typically, a design team will consult the guideline reference manual several times before reaching the stage of a high fidelity prototype, which can be subsequently used for further usability analysis through user-testing, inspections or otherwise.

Despite the sound human factors input propagated into interactive software development life-cycle through guidelines, a number of problems have also been encountered. In what follows, an attempt is made to briefly summarise some of the main shortcomings in employing guidelines during the user interface design process, and to identify means of potential improvement.

First, a review of the available guideline reference manuals reveals that in the majority of cases the consolidated material is expressed as either general or platform specific rules. In the former case, guidelines are independent of any particular context, while in the latter case, they relate to a particular library of interaction objects, such as Windows95, and provide guidance on the physical and in some cases the syntactic aspects of interaction. Domain-independent guidelines typically raise a compelling need for contextual interpretation; a task which is both interaction and collaboration intensive. Moreover, any interpretation effort is bound by the capability, experience and breadth of the knowledge of the designer (or the specialist involved).

A second drawback is that guidelines are not always experimentally validated or the experimental evidence is inconclusive (e.g. research results available on the use of speech versus keying). In general, the currently available experimental work, though substantial, does not cover the broad range of alternative and novel design options that become available by advanced interaction technologies. Moreover, due to radical changes that occur in the mainstream Information Technology industry, some of the past experimental results rapidly become invalid or out of context.

Thirdly, guidelines are often difficult to communicate to developers. In particular, recommendations derived from guidelines are not always comprehensible or easily appropriated by the development team. This is not only due to the typically demanding task of implementing these recommendations, but also due to the doubts that are frequently expressed regarding the validity of a particular recommendation in a given design case.

Another potential shortcoming is that design recommendations derived through different sets of guidelines are often conflicting (sometimes, such conflicts may be encountered even within the same set of guidelines). In other words, one guideline may invalidate another guideline. At the same time, guideline documents or reference manuals offer no natural way of resolving ambiguities, which typically leads to arbitrary decisions by the design team.

In certain cases of domain specific guidelines, the special vocabulary used poses additional problems (e.g. accessibility guidelines for disabled users). This arises from the language used in these documents, which is not always comprehensible by the designers or the developers of user interfaces. As a consequence, additional training is usually required before the development team can effectively and efficiently use a guideline manual and implement the relevant recommendations.

In addition to the above shortcomings, several studies investigating the use of guidelines by designers and developers have concluded that guidelines are frequently ignored. This is partly attributed to the fact that such knowledge is not easily exploitable by user interface designers [8], and partly due to the view that guidelines and style guide documents are inefficient ways of communicating human knowledge factors to the designer [9,10]. This is especially the case when guidelines are not translated into unambiguous design specifications with explicit content and scope, or when the guidelines reference manual becomes voluminous.

The appreciation of these shortcomings has led many researchers to consider alternative media for making guidelines more usable during the early design phases. One recent approach in this direction is the development of tools for working with guidelines. This paper falls within this line of work and aims to address the issue of potential improvements upon the currently available design practice and methodologies for propagating human factors design input into the user interface development life-cycle. In this context, the present work reports on a new framework and supporting tools environment for the consolidation, retrieval and propagation of guidelines into user interface design and development activities.

The paper is structured as follows: the following section reviews the results of recent work related to tools for working with guidelines. The analysis identifies available systems, their primary scope and purpose, their underpinning assumptions, and provides the motivation and rationale for the work undertaken which is described in subsequent sections of this paper. Then, the supporting tool environment is presented, in terms of its architecture and functionality. Subsequently, the results of a user-based evaluation of the Sherlock Guideline Management System are presented. The paper concludes with a summary and discussion of the outcomes of the present work.

2. Related work

In order to address some of the problems identified in the previous section, but also as an attempt to provide facilities for automated evaluation of interactive components in software applications, a number of recent research efforts have investigated the possibility of providing computer-based support for working with guidelines. The normative perspective in these efforts has been to facilitate more effective and efficient human factors input in early design activities. Work in this area has addressed a range of issues related to the access and retrieval of guidelines, the consolidation of guidelines into rule-based components and their subsequent integration within a user interface management system, as well as the possibility of critiquing tentative designs based on a selected subset of encoded guidelines. In the following, we review some of these efforts, with the intention to identify their underpinning assumptions, scope and type of human factors support provided, as well as some of their shortcomings.

2.1. Tools for working with guidelines

Tools for working with guidelines can be broadly classified into four main categories, namely, tools for access and retrieval of guidelines, expert system components for automated user interface evaluation, computational critics for assessing tentative designs, and tools for working with experience-based usability heuristics.

The first category aims towards the development of tools for accessing and retrieving guidelines organised either as a database or hypertext. These tools have emerged recently as an approach to making designers aware of existing guidelines and style guides as reported in documents. Representative examples of this class of tools are SIERRA [11], GuideBook [12], and HyperSAM [13].

Another approach is the development of knowledge based systems for the evaluation of a user interface and its subsequent refinement by the designer. Indicative systems which have been developed to pursue this line of work include Reisner's early work [14] on assessing simplicity and consistency of commands represented in a BNF grammar, the work by Blesser and Foley [15], the KRI/AG system [10], IDA [16] and the EXPOSE system [17].

More recently, there has been work in the area of critiquing tentative designs [18] and either reporting (the lack of) compliance against guidelines, or actively applying guideline prescriptions so as to automatically update a design. The primary focus of these systems is to empower designers of user interfaces when designing low level details of a user interface. In the literature, there have been a few reported efforts illustrating the use of this technique in particular design domains [19–21].

Finally, there have been efforts aiming to consolidate past experience of an organisation into a usability cases repository that can be used to recall past design problems and solutions given, as well as to support human-factors knowledge persistence and evolution, as the organisation's expertise in a particular area grows and expands. An example of this category of systems is the Mimir prototype [6,7]. The objective of this approach builds on the notion of creating a critical mass of design materials from an existing collection of guidelines and providing tools for contextualising them into the organisation's line of activities, as well as depositing new experience which can be revised and reused in future design problems.

2.2. Motivation and rationale of the present work

The present work is motivated by the normative perspective that tools for working with guidelines should provide a collaborative, extensible and evolutionary medium, offering more than mere access to guideline reference manuals or hypertext retrieval. In this endeavour, we seek to address the current limits in automated user interface evaluation [22,23], with the intention to extend the range of functionality and the scope of existing tools, towards an integrated computer-based platform for working with either general, platform-specific or experience-based guidelines. To this effect, the current work builds upon the relative merits of each category of tools described above and provides a framework and a supporting tool environment which: (i) integrates alternative perspectives (i.e. guidelines access and retrieval, automated evaluation, expert critiquing and use of experience-based rules) and (ii) empowers both the design team during the early phases of prototypical development and the usability analyst when inspecting prototype versions to assess compliance against an agreed set of rules or a standard.

3. The Sherlock framework for working with guidelines

In order to provide a contextual account of the development of Sherlock, we propose that before presenting the technical details of the system, we briefly elaborate the specific objectives and requirements driving the work being presented. The objectives of Sherlock were four-fold. First, it was considered important to provide facilities for accommodating a broad range of guidelines, including general-purpose guidelines, standards recommendations, experience-based heuristics and corporate styleguides. To this effect, Sherlock is totally open as to the source or the range of the guidelines considered relevant to a particular design problem. However, the system does require that guidelines are consolidated and become embedded into a guideline knowledge server that can be interrogated by one or more clients. In case of multiple guideline sources or different versions of a particular reference manual, Sherlock provides mechanisms for identifying ambiguities resulting from conflicting or duplicate guidelines and offers guidance to the designer so as to resolve the conflict.

A second objective was to provide a system that would augment the design process by supporting access and retrieval of guidelines, *silent* and *active* critiquing, error reporting and reuse of past experience. Sherlock offers a range of functionalities to facilitate the above. More specifically, usability problems encountered by the system are linked directly to an online version of the guideline reference manual in use, thus enabling the designer to immediately diagnose which guideline was violated by a particular design defect. Critiquing is facilitated by assessments of tentative designs and can take the form of either *silent* (i.e. identification and reporting of design defects) or *active* (i.e. identification and correction of design defects) critiquing. The responsibility, as to what mode of critiquing is to be employed lies solely with the designer. Additionally, design problems detected are linked to past design cases in which the same or similar problem was encountered and the system informs the designer as to how the design defect had been addressed.

The third technical objective driving the development of Sherlock was the need for inter-operation of the guideline management system with an existing and popular user interface development system. This would bring human factors design input, embodied in guidelines, closer to user interface development and implementation. To facilitate this, Sherlock was implemented as an *add-in* to the Visual Basic Development Environment. This decision was based on the popularity of Visual Basic amongst academic and industrial practitioners. It should be noted however, that Sherlock follows a client/server model, which means that the server (containing the guideline knowledge) may be exposed to a broader usage context than that of the Visual Basic environment.

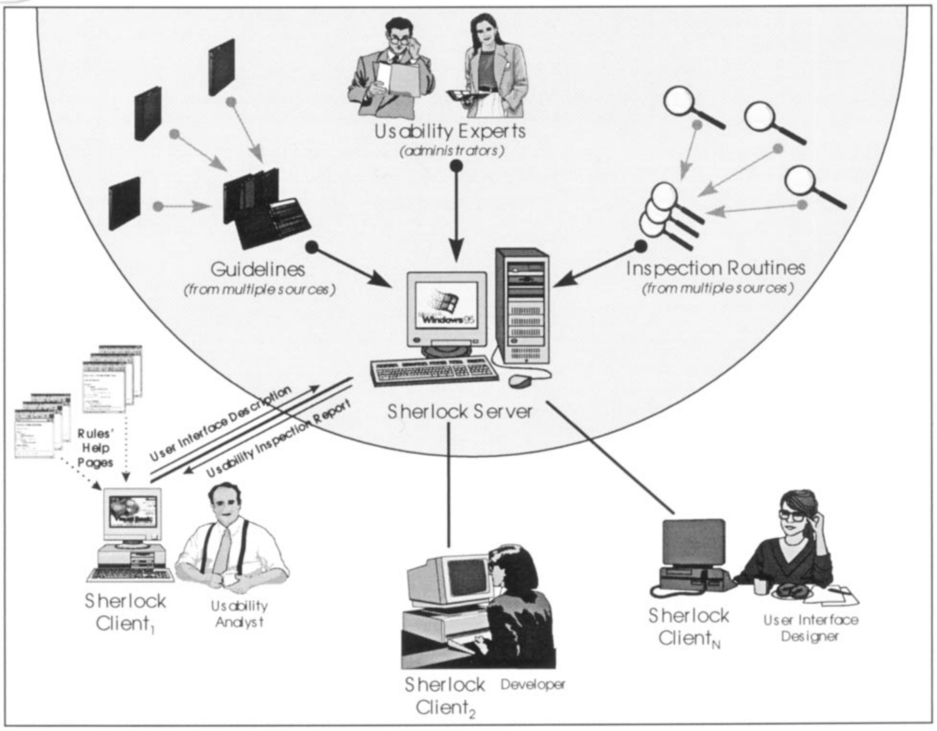


Fig. 1. Sherlock overview.

Finally, the fourth objective of the present work was to construct a system which would provide an evolutionary and extensible medium for working with guidelines. This means that the system should be tailorable and customisable to the changing needs of an organisation, while providing a powerful mechanism for accumulating organisation-wide experiences and design wisdom. Such a requirement immediately raises the issue of extensibility which concerns the hooks offered by a system to allow for the introduction and depositing of new material. As shown in the following sections, Sherlock implements an explicit extensibility model with a dedicated set of function calls allowing the content of the server to be updated and refined.

3.1. System overview

A contextual overview of the Sherlock system is depicted in Fig. 1. The shaded area identifies the Sherlock server and highlights typical activities undertaken by the intended users. The server integrates guidelines (as rules) and inspection routines from various sources, local or remote to the server. Typical users of the server, are usability professionals, including design teams introducing corporate style guides. On the other hand, Sherlock clients can be used by developers, user interface designers or usability experts. The distinctive characteristic of this context model is that clients send user interface descriptions for inspection to the server, while the server reports back to them the identified design deficiencies derived through usability inspection.

Sherlock is a client/server guideline management system (GMS). The server and client modules can reside on the same or different computers that are connected through the Internet or through a company's intranet and communicate using the TCP/IP protocol. The server module runs under Microsoft Windows 95 and the client module is an add-in to the Visual Basic 5.0 Integrated Development Environment (IDE). Both modules are implemented using ActiveX technologies that enable Sherlock to appropriate some of their important properties such as:

- *interoperability* of components regardless of their creator;
- *language independence* so that components written in different languages can work together;
- *component versioning* to enable alteration or upgrade of a component without affecting other components and applications that make use of it;
- *transparent use of the network* allowing the distribution of applications.

Sherlock takes advantage of these properties to provide an extensibility model in which modifications or updates of (i) the rule base, (ii) the inspection routines and (iii) the system components, are easy to make and are immediately propagated to the client in a way transparent to the application user.

As already pointed out, the server's role is the inspection of user interfaces on the basis of specific usability guidelines and the reporting of possible guideline violations. Additionally, the server is also responsible for keeping the clients up-to-date, whenever the rule base is updated, keeping track of guideline violations encountered, as well as for consolidating knowledge about users' solutions to usability problems. The client's role is to send a textual description of a user interface to the server for inspection and then report the inspection results to the developer.

3.2. The rules

In the context of the Sherlock GMS, a rule is derived through the interpretation of a relevant guideline and is embedded—in an application-specific format—in the Sherlock rule base. In order to provide easy, intuitive and effective rule handling by the user, the embedded rules are classified according to two criteria; namely, the rule's *class* and *inspection type*.

A rule's *class* is an attribute derived from the observation that collections of guidelines usually have a hierarchical structure that can be represented as a tree. Therefore, this attribute depicts the full path from the root of a guidelines collection tree to the leaf containing the specific guideline. For example, the ISO 9241 standard [24] contains 17 parts each of which is further decomposed into sections which, in turn, contain subsections that are groups of thematically-related guidelines. The root of the tree is ISO 9241, the branches are the parts, the sections, the subsections and the leaves are the guidelines. Thus, for example, a rule's class can be: *ISO 9241/Part 12/5. Organisation of Information/5.9 Labels*.

The rules are not structured according to any particular guideline. The structure of the

Table 1
Rule inspection types

Inspection type	Description	Example
1. Automatic	This category contains rules that can be automatically evaluated by the server.	'*Field labels and labels for other screen elements including text boxes, list boxes, combination boxes and icons, are consistently located adjacent to the displayed field, group or screen element*' (ISO 9241: Part 12/5. Organisation of Information/5.9 Labels/5.9.5 Label Position) In this case the system can decide whether this rule is violated or not by examining the given interface description and propose corrective actions.
2. Semi-automatic	This category contains rules that refer to a particular interface element e.g. to a label, but additional external knowledge is needed in order to decide whether this rule is violated or not. In this case, Sherlock acts as a guideline "reminder" and presents the situation as a possible rule violation but it is left to the user to decide whether this is a problem or not.	'*Field labels explain the content of the fields*'. (ISO 9241: Part 12/5. Organisation of Information/5.9 Labels/ 5.9.3 Designation) The system can deduce that this rule is relevant, since the interface does contain a label, but there is no way of deciding whether the label explains the specific field.
3. By the user	This category contains high-level rules that the system cannot infer if they are relevant. These rules are usually general recommendations and they refer to the interactive behaviour of the application or to user characteristics	'*Information should be located to meet user expectations*' (ISO 9241: Part 12/5. Organisation of information/5.2 Organisation of information/5.2.1 Location of information)

available rules is dynamically created, every time Sherlock starts, by decomposing the *class* property of each rule.

Inspection types fall into three categories, namely '*Automatic*', '*Semi-automatic*' and '*By the user*'. This classification is grounded on the notion that automatic evaluation is possible for only a subset of the available guidelines [22]. The reasons for this are two-fold. Firstly, there are guidelines that require task-specific knowledge or end-user information which cannot be reliably inferred by the system. Then, a large set of guidelines examine the interactive behaviour of the application. Since Sherlock is operating on the user interface design environment, only static characteristics of the interface can be acquired. Because of these requirements, three inspection types were identified. These are described in Table 1.

Most—if not all—guideline documents contain guidelines that belong to all of the above inspection types. This is an additional reason why a tool for working with guidelines should be capable of integrating any of these categories.

The Sherlock rule base is independent of the physical implementation of the rules. This practically means that any type of rule can be integrated. In order to achieve this

Table 2
Rule profile

Property	Description	Examples/set of values
Key	A unique string description that identifies a rule. This feature is to be used only by application programmers.	“DGRule1.2.3”
Title	The rule’s title. Though not imposed by the application, rules belonging to the same class should have distinct titles.	“5.9.5 Label Position”
Class	A string that contains the full path of the class this rule belongs to (see Section 3.2 Rules). A class can contain other classes and/or rules. The rules classes are neither fixed, nor explicitly declared. Sherlock builds dynamically a class tree by analysing the class information of each rule’s profile. Because of this fact there are no empty classes in the tree.	“ISO 9241/Part 12/5. Organisation of Information /5.2 Organisation of information”
Description	A short textual description of the rule.	“Field labels and labels for other screen elements including text boxes, list boxes, combination boxes and icons, are consistently located adjacent to the displayed field, group or screen element”
Severity rating	A rating that denotes the severity of a usability problem detected. Severity ratings are used to help developers prioritise the fixing of usability problems but also to get an overall picture of the interface’s usability.	0—“No Problem” 1—“Cosmetic Problem” 2—“Minor Problem” 3—“Problem” 4—“Major Problem” 5—“Usability Catastrophe”
Inspection type	(see Section 3.2 Rules)	—“Automatic” —“Semi-Automatic” —“Inspection by the User”
Related theory page	This attribute is actually a URL of a web page. Each rule can be associated with a web page—or even a whole web site- that contains relevant information, such as, theoretical background, in-depth description, violation and correction examples. These pages are presented in a customised web browser with specific features that assist navigation through collections of diverse and dispersed web pages about rules.	http://www.guidelines.com/heuristics/labels.html
Reference	A bibliographic reference to the source of the rule.	“Microsoft Corporation. The GUI Guide: International Terminology for the Windows Interface. Redmond, WA: Microsoft Press, 1993. ISBN 1-55615-538-7, pp. 73”

Table 2 (*continued*)

Property	Description	Examples/set of values
Permission	Sherlock enables users to ask for specific rules or set of rules to be ignored during the usability inspection. For a variety of reasons, as for example lack of expertise, teaching of a new set of rules, or inappropriateness, the system administrator might decide that specific users or user groups should not have this privilege over specific rules. This attribute contains a list of users that have the permission to select whether a rule will be ignored or not during the inspection.	—“All” —“None” —“Administrator, User123”
Enabled	The default state of a rule when added to the rule base. A rule’s state can be altered only by users who are included in the rule’s permission list (see above).	—“True” —“False”
Conflicts with	A list of rule keys belonging to rules that are known to conflict with the current one.	“MyRule1, Heuristic1.2.3, ISO9241-5.4.3”
Author	The rule’s *author*.	“John Doe”
Source component	The name of the DLL component in which this rule resides. This piece of information, along with the Author, is used for helping administrators resolve conflicts between components that have overlapping sets of rules	“myExtension.dll”

implementation-independence, a short profile must be completed for every rule by the rule author. This profile contains the pieces of information depicted in Table 2. The procedure for identifying ambiguities and conflicts between rules uses this profile, and thus, it is not fully automated, since it relies on human input; but this is the only way that such semantic information can be elicited.

3.3. *Sherlock server*

The server module is the core of the Sherlock GMS and has an open and easily extensible architecture. In the current version, Sherlock uses the TCP/IP protocol for communicating with clients, but since messages are plain ASCII text, other ways of network communication can also be used, e.g. e-mail.

The server’s main task is two-fold: the inspection of user interfaces according to specific usability guidelines, and the subsequent reporting of possible guideline violations. There are no rules or inspection routines embedded in the server module. These parameters reside in external modules (Dynamic Link Libraries or DLLs) that can be created by any programming language that has the ability of creating ActiveX DLLs. Extension DLLs can reside locally on the server site, or be dispersed over the Internet. The inspection is performed upon an abstract user interface object hierarchy that is built by the client module. This means that the data handled by the server are platform-independent. Thus, the same server can accommodate clients integrated in different development platforms without any modifications or reprogramming of the system.

The server is also responsible for: keeping the clients up-to-date whenever the rule base is updated; keeping track of guideline violations encountered; and, consolidating knowledge about users' solutions to usability problems.

Typically, users of the server module are expected to be usability experts, that can:

- serve as administrators of the system, upgrading the rule base by using third-party components, customising the rule base for specific end-users, refining and updating the rules' descriptions and help pages;
- act as third-party component providers by building new rule components for extending the rule base;
- build focused and effective training programmes for customer companies by studying the database of frequent usability problems and past solutions.[1]

The Sherlock server comprises five basic components (Fig. 2), which are briefly discussed below.

3.3.1. The user interface composer

The *user interface composer* takes as input a textual user interface description received by a client through the communication module and converts it into a hierarchical structure that is later used by the *usability inspector module*. The root of this structure is a *project*. A project contains *forms* (*windows*) which in turn contain *controls*. Some controls, that are usually referred to as *container controls*, can contain other *controls*, either *plain* ones or other *containers*.

A thorough study of a substantial set of guidelines from a variety of different sources, reveals that a considerable percentage of them refers to specific *controls*, such as *labels*, *buttons*, etc. For this reason, the *user interface composer* provides to the rules programmer a set of functions for easy and direct access to collections of related controls, in order to avoid the time-consuming and error-prone task of browsing through the whole interface structure.

3.3.2. The client/server communication module

This module is built using Windows Sockets and is responsible for the communication between the client(s) and the server. Incoming messages are delivered to the recipient components and their reception is acknowledged to the sender. Outgoing messages are sent and retransmitted as necessary. We are currently extending the functionality of this component with additional functions, such as data compression and encryption.

Input to this module are messages sent by clients over the network. Such messages can be: requests for connection and disconnection, descriptions of hierarchical user interface elements for inspection, the client's list of disabled rules and the client's usability problems and correction history (with the user's permission). The output of the *communication module*, are messages sent from the server to a specific client. These messages

[1] In a future version of Sherlock, such tutorials and training sessions will be offered on-line through the client module.

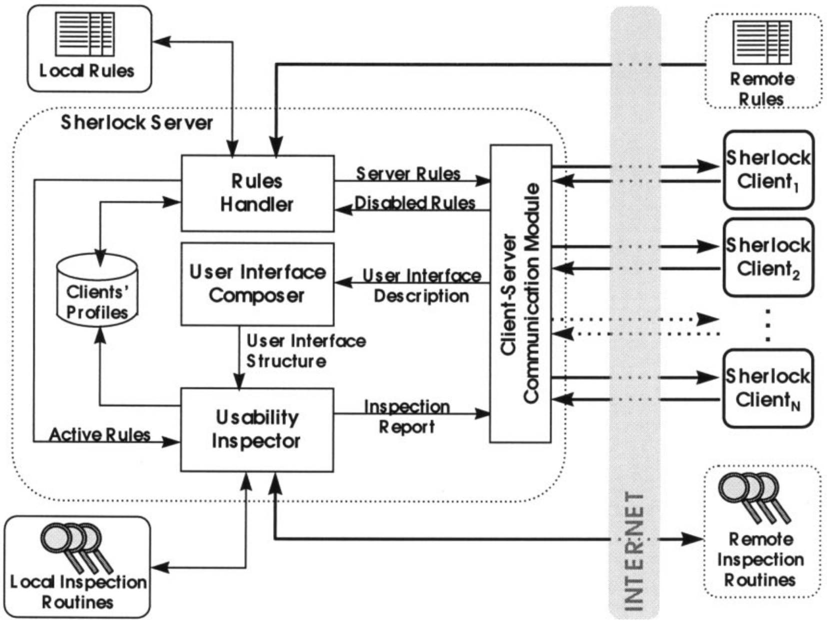


Fig. 2. Sherlock server architecture.

include: connection acceptance and rejection, a list of inspection results and a list of the server's new version of rules.

3.3.3. The clients' profiles database

This database is a repository of client-related information, which is summarised in Table 3.

3.3.4. The rules handler module

At the start-up of the server, the *rules handler* initialises the server's rules collection by using all the available sources of rules. Sources of rules can be local or remote DLLs. Each DLL contains a list of rules and a set of the respective inspection routines. The rules handler interrogates the DLL to find out which rules it contains. In case some DLLs contain overlapping sets of rules (e.g. two different DLLs have a set of rules about ISO 9241-Part17) the rules handler prompts the administrator to prioritise the rules' sources and can keep track of user preference for automatic conflict resolution (e.g. when there are more than one sources for rules about the ISO 9241, always prefer the DLL at the ISO server).

New sources of rules can be easily added by declaring the name and location of a DLL. With the help of the rules handler, the administrator can browse through the rules that this DLL offers, and select a relevant subset for inclusion in the server rules. When the rule base changes, the rules handler notifies the clients and sends them an updated version of the rules.

Table 3
Clients' profiles database content

Table	Description
Communication history	When and for how long a client was connected to the server, unsuccessful connection attempts (e.g. due to wrong password), the number of requests for evaluation made, the size of data exchanged, etc. This information is mainly used for logistic and security reasons but it can also be used for more efficient load balancing in case more than one Sherlock severs exist. For example, clients that are known to produce high payloads can be distributed across different servers.
User data	Data used for user identification. Name, password, etc.
Client's last version of the disabled rules list	As mentioned before, every client has a list or disabled (inactive) rules that should not be taken into account during the inspection process. In order to minimise the size of the communication data this list is sent to the server only when it has changed. The server stores this list in the clients' profiles database and retrieves it whenever needed.
Past knowledge	Information about frequently encountered usability problems and corrections provided by the user.

Extending the rule base is another important functional objective of this module. To this effect, Sherlock provides an open and extensible environment for accommodating a broad range of guidelines, that may range from very specific platform-dependent styleguides to high-level usability standards.

The rule base and the relevant inspection routines can be extended by ActiveX DLL components. Such components can be developed in any programming language that supports ActiveX DLL programming and can reside on the same computer as the Sherlock server, or on any other computer connected to the Internet. This feature offers a very flexible and effective mechanism for updating, since a component author has just to replace the component at his site without also having to redistribute it to customers.

The Sherlock extensibility model describes the specific programming interface (*set of functions*) that extension DLLs must support. The only constraint imposed on the rules' developer is the existence, and calling conventions, of the routines used for declaring new rules and for reporting usability inspection results. The actual implementation of the inspection routines is totally unconstrained.

A typical extension DLL contains a class named 'Connect', which is used by the server for creating and communicating with new extension components. The 'Connect' class should contain (at least) the following two routines:

1. *Public Sub Rules_Declare* (*Server as clsServer*): this routine is used to declare to the server the rules that this component contains. The body of this routine consists of a list of calls to the *Rule_Declare* function (Table 4) of the server, which describes a rule's profile, as depicted in Table 2.
2. *Public Sub Evaluate* (*UITree As clsUITree, ruleKeys As String*): this routine is invoked by the server during inspection, to ask the DLL component to execute the routines that are responsible for evaluating the interface against the rules that are contained in the *ruleKeys* argument (this argument contains a list of rule keys). The *UITree* parameter is

Table 4
Declaring a rule

```
Function Rule_Declare   (Key, Title, Description, Class, Severity,
                        InspectionType, Optional Enabled, Optional
                        RelatedTheoryPage = "⟨None⟩", Optional Reference =
                        "⟨None⟩", Optional Permission = "All", Optional
                        ConflictsWith = "⟨None⟩", Optional Author = "⟨Unknown⟩",
                        Optional SourceComponent = "⟨Unknown⟩") As Long
```

Note: To simplify the function's parameters the arguments' data types have been ommited as well as whether they are passed 'by reference' or 'by value'.
Example:

```
Public Sub Rules_Declare(Server As clsServer)
                         Call Server. Rule_Declare ("Sheur1.1.2", "1.1.2
                        Missing colon ':' ", "Field labels should be followed by
                        a colon (:)", "Sherlock Heuristics/1.Layout/1.1
                        Labels", SRProblem, IT Automatic, True,"
                        http://www.server.com/HelpPages/Heuristics/Layout/
                        MissingColon.html", "Sherlock Style Guide, pp. 67",
                        "All", "⟨None⟩", "Me", "SherlockHeuristics")
End Sub
```

a reference to a *User Interface Tree* structure that will be used by the inspection routines (Table 5) to report any rules violations found.

Sherlock does not impose any constraints upon the way the related evaluation routines will be selected, executed or implemented. Sherlock just informs the DLL about which rules it expects evaluation results for. This particular design enables the DLL programmer to optimise the code of the evaluation routines. The implementation allows the programmer to evaluate more than one rules using the same routine. Such a property can substantially decrease the inspection time, as well as the lines of code written, through re-usability; moreover, it supports structured programming and easier and more effective code management.

Finally, any usability problems found are reported to the server through calling the *UP_Report* routine (Table 6). If the DLL attempts to report a problem regarding a rule which was not included in the request made by the server, or a rule that does not exist, the server simply ignores the report and saves this violation information for administrative use.

As an illustrative example, let us assume that a new rule is to be accommodated. The description of the rule is "Field labels should be followed by a colon (:)". The code in Table 4 would enable the server to detect that the new rule has been introduced and the code in Table 6 would report a violation of this rule.

3.3.5. The usability inspector module

As mentioned earlier, there are no inspection routines embedded in the server module. Sherlock relies on external routines that are provided with the extension modules. The *usability inspector module* selects the routines that should be executed depending on (a) which rules are active and (b) the partial user preferences—in case more than one modules

Table 5
Calling the evaluation routines

In the simplest case, the evaluation routine can be a big ‘Case’ clause, for example:

```
Public Sub Evaluate (UITree As clsUITree, ruleKeys As String)
Dim colRuleKeys As New Collection, ruleKey As Variant
‘First create a list containing the ruleKeys
  ruleKeysList_Create(colRuleKeys, ruleKeys)
‘For each ruleKey call the related routine
  For Each ruleKey In colRuleKeys
    Select Case ruleKey
      Case “SHeur1.1.1”
        Call LabelTextAllInCaps (UITree)
      …
      Case “SHeur1.2.1”
        Call OKCancelPosition (UITree)
      …
      Case “SHeur4.1”
        Call FeedbackProvision (UITree)
    End Select
  Next
End Sub
```

In other cases, the same routine may be used to evaluate more than one rules, e.g.:

```
Public Sub Evaluate (UITree As clsUITree, ruleKeys As String)
Dim colRuleKeys As New Collection, RelatedRuleKeys As Variant,
    colRelatedRules as New Collection
‘First create a list containg the ruleKeys
  ruleKeysList_Create (colRuleKeys, ruleKeys)
‘then collect ruleKeys for rules evaluated by the Routine1 in a list
RelatedRulesList_Create (colRuleKeys, colRelatedRules, Routine1)
  ‘and call the routine
  Call Routine1 (UITree, colRelatedRules)
‘then collect ruleKeys for rules evaluated by the Routine2 in a list
RelatedRulesList_Create (colRuleKeys, colRelatedRules, Routine2)
‘and finally call the routine
  Call Routine2 (UITree, colRelatedRules)
‘etc.
End Sub
```

offer inspection routines for the same rule. Then, it activates the relevant DLLs and asks them to inspect the given user interface for specific rules violations. Each DLL, reports rule violations directly to the inspector module. At the end of this stage, a report is compiled and is sent to the client.

3.4. Sherlock client

The main characteristic of the client’s design (Fig. 3) is simplicity. Users should not be overwhelmed by a highly complicated application. Sherlock was intended to be as easy to use as a spell checker, a goal that, arguably, has been accomplished; in the simplest scenario of use, all the user has to do is press the ‘Evaluate’ button and then review the evaluation report.

Table 6
Reporting usability problems

```
Public Sub UP_Report (RuleKey, UIComponent, ShortDescription, Solution,
Optional LongDescription = “⟨None⟩”, Optional pPossibleSideEffects =
“⟨None⟩”)
e.g.
Call Uitree.UP_Report (“SHeur1.1.2”, tFormNode.frmName & “.” &
tControlNode.ct1Name, “The label should be followed by a colon ‘:’ ”, “Convert
it to: “& tCaption & “:”, “The label” & tFormNode.frmName & “.” &
tControlNode.ctlName & “should be followed by a colon (“& tCaption &”)”,
“⟨None⟩”)
```

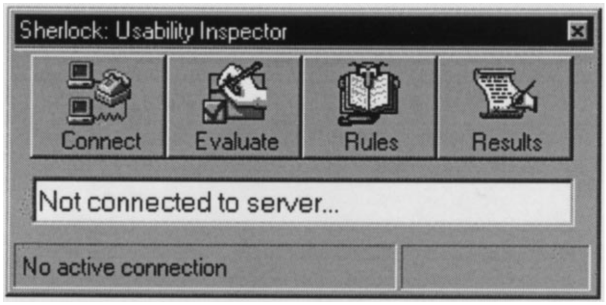
The client’s main role is to compile a textual description of a user interface created in the Visual Basic 5.0 Integrated Development Environment (VB 5.0 IDE), send it to the server for inspection and then report the inspection results to the user. A synopsis of the results is presented in the form of a list. The user can browse through this list and get detailed information about each violation found, such as the cause and the severity of the problem, recommended actions, theoretical background and examples, as well as past solutions. Finally, the client enables the user to disable subsets of inspection criteria from the server’s rule base.

Typical users of the client module include developers/programmers who want to have a “*just-in-time*” evaluation of the interface they are working on, user interface designers who wish to obtain on-line and relevant reference of guidelines related to the interface being designed, as well as usability analysts that can use Sherlock as a support tool for expert usability evaluation.

The Sherlock client comprises six basic components (Fig. 4), which are briefly discussed below.

3.4.1. The client/server communication module

This module is practically identical to the server’s *communication module*. Input to this module are messages sent from the server, such as connection acceptance and rejection, inspection results and a list of the server’s new version of rules. The module’s output



Not connected to server...

No active connection

Fig. 3. The Sherlock client.

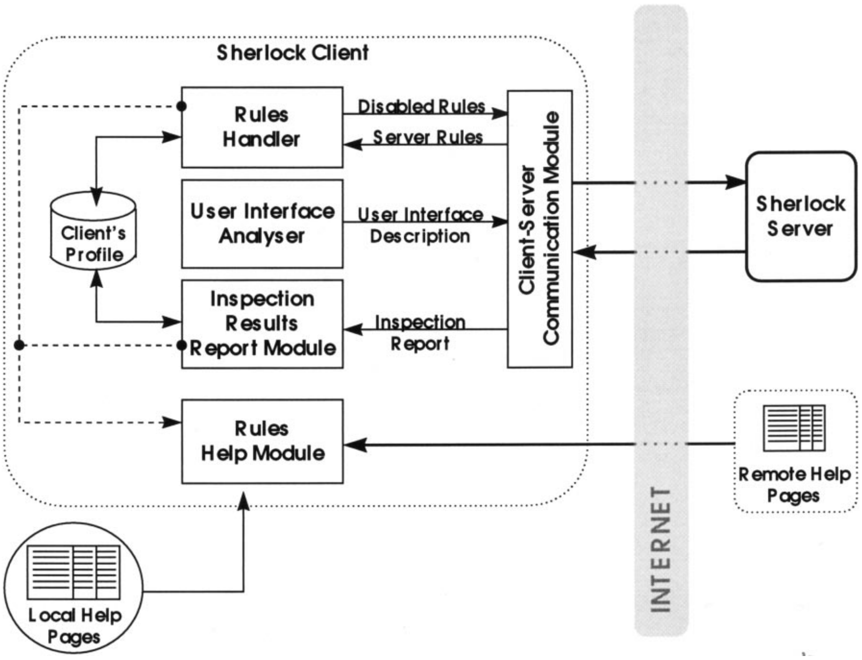


Fig. 4. Sherlock client architecture.

consists of requests for connection and disconnection, descriptions of hierarchical user interface elements for inspection, the client's list of disabled rules and the client's usability problems and correction history.

3.4.2. The client's profile database

This database is used to store:

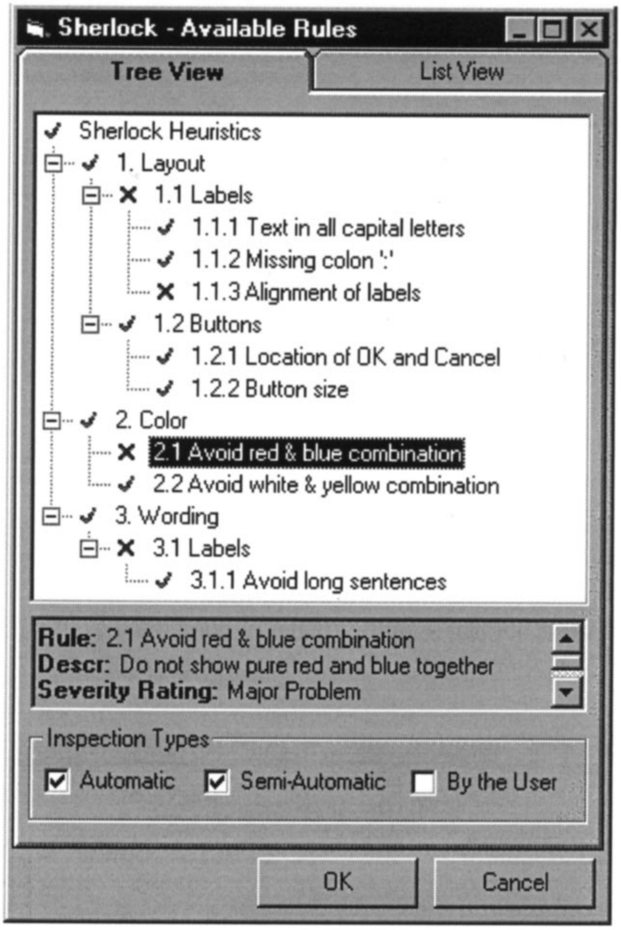
- *Client's last version of the disabled rules list*: this data enables the restoration of the user's last selection of disabled rules when the client is invoked.
- *Client's last version of the server's rules list*: in order to minimise network communication, the client keeps a local version of the server's rules. Whenever these rules change, the server informs the client, and the client updates its database.
- *Past knowledge*: information about frequently encountered usability problems and corrections provided by the user.

3.4.3. The rules handler

The client's *rules handler* is the part of the system that is responsible for the visualisation and handling of the rule tree. The rules are presented in two forms:

3.4.3.1. Tree view

The rules are presented in a tree structure (Fig. 5), resembling the standard view of

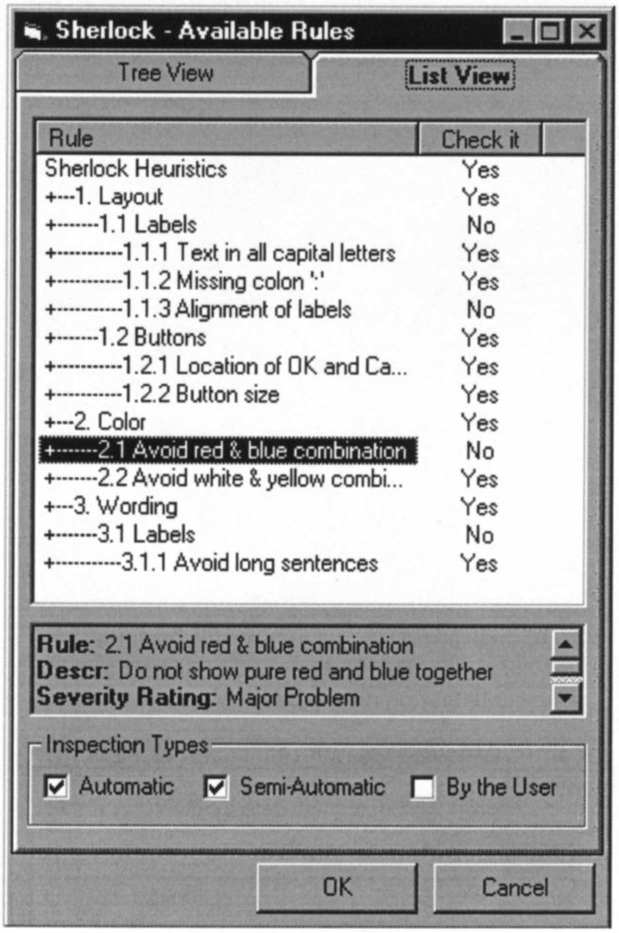

Sherlock - Available Rules
Tree View
List View
Sherlock Heuristics
1. Layout
1.1 Labels
1.1.1 Text in all capital letters
1.1.2 Missing colon ':'
1.1.3 Alignment of labels
1.2 Buttons
1.2.1 Location of OK and Cancel
1.2.2 Button size
2. Color
2.1 Avoid red & blue combination
2.2 Avoid white & yellow combination
3. Wording
3.1 Labels
3.1.1 Avoid long sentences
Inspection Types
Automatic
Semi-Automatic
By the User
OK
Cancel

Rule: 2.1 Avoid red & blue combination
Descr: Do not show pure red and blue together
Severity Rating: Major Problem

Fig. 5. Tree view of the rules.

Windows Explorer. Rule classes are presented as tree branches, while rules as tree leaves. Active rules and classes are preceded by a green tick mark (✓), while the disabled ones are preceded by a red diagonal cross mark (×). If the current user does not have permission to modify a rule's state, then the rule's preceding symbol is enclosed in a red circle (⊘, ⊗). The user can toggle the state of a rule (provided that permission is granted) simply by clicking on it.

When a rule is selected, its profile is presented in the text box underneath the tree. If the selected rule is known to conflict with one or more already active rules, the system informs the end-user accordingly. Then, the end-user has three options: (a) disable all the rules that are in conflict with the new one; (b) do not activate the new rule and; (c) activate the new rule as well. In the latter case, when the inspection results are presented in the Inspection Results Window (Section 3.4.5), an extra heading named 'Conflicts With' is added; under



Rule	Check it
Sherlock Heuristics	Yes
+---1. Layout	Yes
+-------1.1 Labels	No
+-----------1.1.1 Text in all capital letters	Yes
+-----------1.1.2 Missing colon ':'	Yes
+-----------1.1.3 Alignment of labels	No
+-------1.2 Buttons	Yes
+-----------1.2.1 Location of OK and Ca...	Yes
+-----------1.2.2 Button size	Yes
+---2. Color	Yes
+-------2.1 Avoid red & blue combination	No
+-------2.2 Avoid white & yellow combi...	Yes
+---3. Wording	Yes
+-------3.1 Labels	No
+-----------3.1.1 Avoid long sentences	Yes

Rule: 2.1 Avoid red & blue combination
Descr: Do not show pure red and blue together
Severity Rating: Major Problem

Inspection Types
[✓] Automatic [✓] Semi-Automatic [ ] By the User

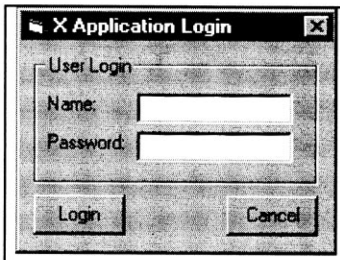
Fig. 6. List view of the rules.

this heading, the conflicting recommendations appear. When a rule class is selected, this text box contains the full path of the class and the number of rules it contains. Below the description box, there is a frame containing three check buttons. Each of these buttons represents one inspection type. When a button is checked, rules belonging to the relevant inspection type are enabled (and presented), otherwise the rules are disabled (and hidden). After configuring the rule tree, the user can either append preferences to the client's profile, or just cancel the changes made and reset the rule tree to the last saved one.

3.4.3.2. List view

The rules are presented as a list (Fig. 6). Root classes do not have a prefix. Child classes have as prefix a number of minus signs ' − '. The number of minus signs equals the depth of the class in the rule tree multiplied by three. The list has two columns. On the left one,

Table 7
Sample textual description of a VB project

Sample project	Textual description
 X Application Login User Login Name: Password: Login Cancel *Note: Some parts of the adjacent description have been truncated to keep it sort*	

```
Begin Project prjLogin
Begin Form frmLogin
 Caption = "X
Application Login"
 Height = 240
 Width = 320
Begin CommandButton cmdLogin
    Caption    =   "Login"
    Height    =   25
    Left    =   224
    TabIndex    =   6
    Top    =   184
    Width    =   81
End CommandButton
Begin CommandButton cmdCancel
      ...
End CommandButton
Begin Frame frmLogin
    Caption    =   "User Login"
      ...
    Begin TextBox txtPassword
      ...
    End TextBox
    Begin TextBox txtName
       Height    =   19
       Left    =   64
       TabIndex    =   2
       ToolTipText    =   "User's name"
       Top    =   24
       Width    =   97
    End TextBox
    Begin Label lblPassword
      ...
    End Label
    Begin Label lblName
...
    End Label
  End Frame
End Form
End Project

This file was created by:
* Sherlock: The Usability Inspector® *
```

classes and rules are listed. On the right one, there is a ‘Yes’ or a ‘No’ clause, describing whether the rule or class is active. In case that the user does not have permission to modify the state of specific rules or classes of rules, the clause is preceded by an equal sign (‘ = Yes’, ‘ = No’). The rest of the controls are shared between both presentation styles.

3.4.4. The user interface analyser

The *user interface analyser* is a non-interactive module of the client. Its function is to parse a Visual Basic project and create a textual description of the user interface (Table 7).

Table 8
Description of a rule violation

Attribute	Description
Problem area	The component(s) of the interface related to the rule violation.
Usability problem	A short description of the usability problem.
Severity rating	See Table 2.
Class	See Table 2.
Inspection type	See Table 2.
Solution	A suggested corrective action.
Long description	A more detailed description of the problem.
Possible conflicts	A list of detected problems, that might conflict with the current one.
Status	The status of a problem can be: 1. Active 2. Not applicable 3. Fixed—in this case, if information about how it was fixed is provided by the user, it is presented as well

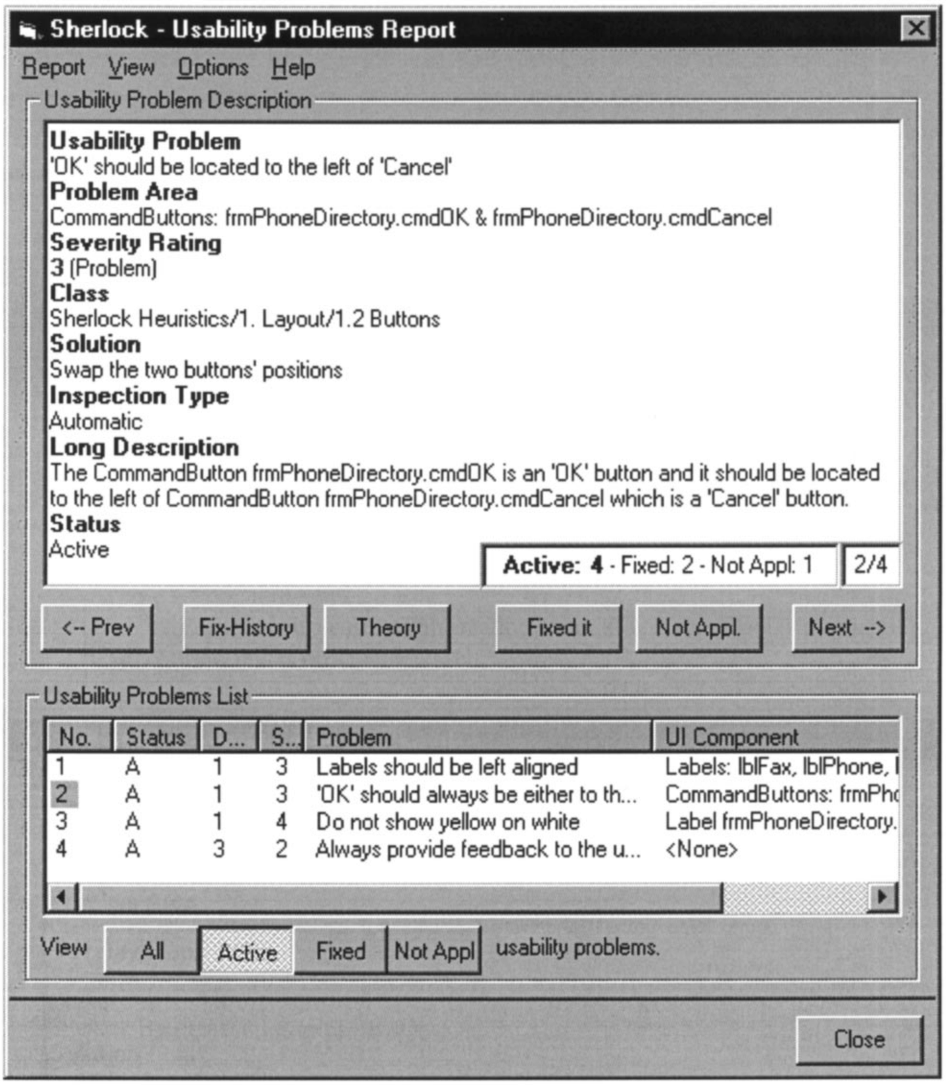
In order to minimise the data sent over the network, default property values are used. The client keeps a table of default values for a range of properties of a form or a control. When the textual description is created, a property is appended to it, only if it differs from the default value. This technique has been shown in practice to reduce the description's size up to 80%.

3.4.5. The inspection results report module

As Jeffries states in Ref. [25], the most critical aspect of usability inspection, is communicating the results to the developer. It is quite often the case that problems are overlooked because of lack of understanding, or sometimes, false alarms occurring that lead to changes that will have no positive impact on usability and might even make the application less usable. In this respect, Sherlock provides an in-depth analysis of each rule violation detected (Table 8), that helps the developer acquire an accurate understanding of the area and facilitates a more informed judgement on the usability issues involved.

The *inspection results report* window comprises three parts (Fig. 7). In the upper part, a single problem is presented using all the available information. In the middle of the window, a set of push buttons allows the user to (a) move to the next/previous problem, (b) access background information about the problem (such as related theory and examples), (c) view a history of previous solutions to the same problem (Fig. 8) and (d) classify the current problem as '*Fixed*' or '*Not Applicable*'.

When a rule violation is detected, a usability problem is reported that is automatically classified as '*Active*'. Browsing through the inspection report, the user can explicitly declare a problem as '*Fixed*' and optionally provide information on the actual steps taken to tackle the problem (Fig. 9); alternatively the user may decide that the violation reported was '*Not Applicable*' to the specific interface. This classification data is stored in the user's profile as part of the inspection history. All the classification actions are reversible. The user can easily move a '*Fixed*' or a '*Not Applicable*' problem back into '*Active*' state. In this case, all the related modifications made to the user's profile are automatically undone. The above approach to problem classification was selected in order to assist the developer in


Sherlock - Usability Problems Report
Report View Options Help
Usability Problem Description
<-- Prev
Fix-History
Theory
Fixed it
Not Appl.
Next -->
Usability Problems List
View All Active Fixed Not Appl usability problems.
Close

Usability Problem
'OK' should be located to the left of 'Cancel'
Problem Area
CommandButtons: frmPhoneDirectory.cmdOK & frmPhoneDirectory.cmdCancel
Severity Rating
3 (Problem)
Class
Sherlock Heuristics/1. Layout/1.2 Buttons
Solution
Swap the two buttons' positions
Inspection Type
Automatic
Long Description
The CommandButton frmPhoneDirectory.cmdOK is an 'OK' button and it should be located to the left of CommandButton frmPhoneDirectory.cmdCancel which is a 'Cancel' button.
Status
Active

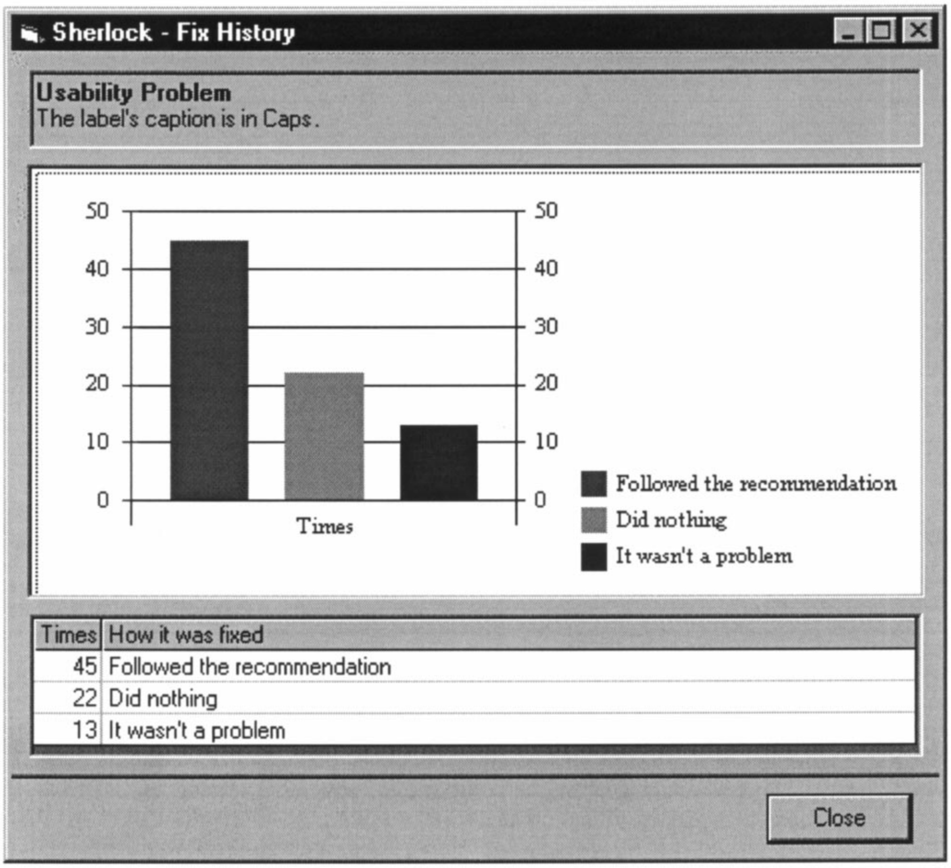
Active: 4 - Fixed: 2 - Not Appl: 1 2/4

No.	Status	D...	S...	Problem	UI Component
1	A	1	3	Labels should be left aligned	Labels: lblFax, lblPhone, l
2	A	1	3	'OK' should always be either to th...	CommandButtons: frmPhc
3	A	1	4	Do not show yellow on white	Label frmPhoneDirectory.
4	A	3	2	Always provide feedback to the u...	<None>

Fig. 7. Inspection results report.

organising the task at hand, by providing a quick overview of the problems found, the ones which have been taken care of, those ignored, as well as those still pending.

The lower part of the *inspection results report* window is a list that contains one of the following, depending on the user's choice: (a) all the problems found, (b) the '*Active*' problems, (c) the '*Fixed*' problems, (d) the '*Not Applicable*' problems. Each list entry represents a single problem detected, and contains a short description of the problem, its status, diagnosis type and severity, the user interface component(s) related to the violation and, finally, a numerical identifier which correlates a list entry with the (sequence number



Usability Problem
The label's caption is in Caps.

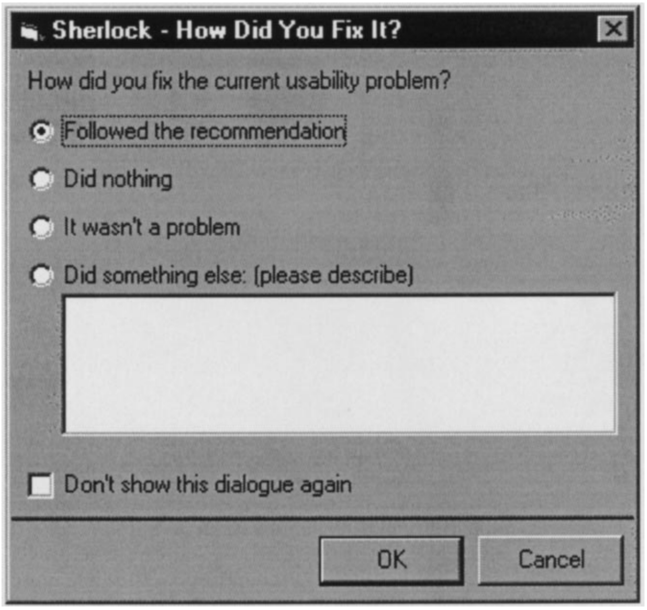
Times	How it was fixed
45	Followed the recommendation
22	Did nothing
13	It wasn't a problem

Fig. 8. Reviewing deposited information.

of the) rule presented in the main part of the window. The list can be sorted by any one of these attributes. The user can retrieve more details about a specific usability problem simply by clicking on it.

3.4.6. The rules help module

The *rules help module* is a customised web browser (Fig. 10) that presents rule-related information to the user. The main problem with this type of information is that, since it comes from different sources, it does not have a specific structure or format. The format can be "homogenised" through the use of common web page design guidelines, but there is no way for creating an explicit structure, since the related data is changing dynamically and may be distributed over the Internet. This is why, this module creates on-the-fly a "*table of contents page*", based upon the *Class* information in the rules' profiles, which presents the underlying (implicit) structure of the information. This "*table of contents page*" is always accessible to the user, through a dedicated browser button. In addition to


Sherlock - How Did You Fix It?

How did you fix the current usability problem?

[✓ Followed the recommendation]
Did nothing
It wasn't a problem
Did something else: (please describe)

☐ Don't show this dialogue again

Fig. 9. Depositing information about fixing a problem.

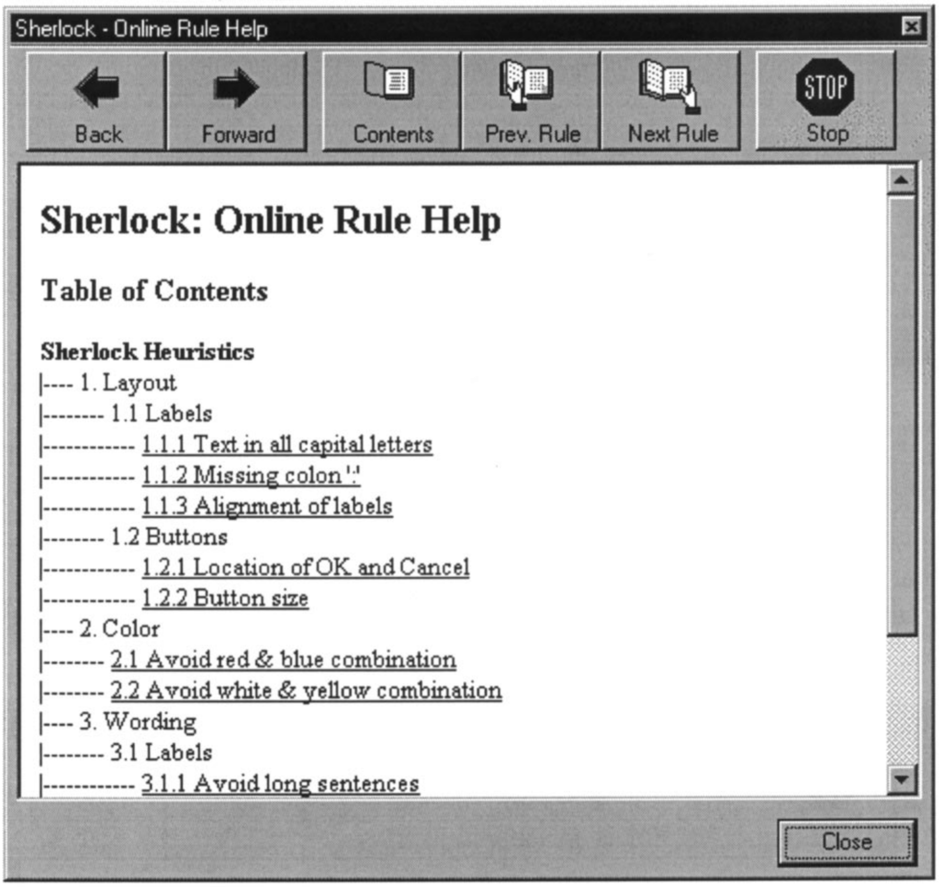
the traditional ‘Back’ and ‘Forward’ buttons, this browser provides ‘Previous Rule’ and ‘Next Rule’ buttons, that are used for navigating the information structure.

4. Evaluation

Sherlock was evaluated by eight expert users with substantial experience in user interface design. All users have a Bachelor’s, a Master’s or PhD degree in Computer Science, and three to eight years experience in the field of human–computer interaction. The user group consisted of five males and three females whose age ranged from twenty-five to forty years.

Since there are no existing systems depicting equivalent functionality with Sherlock, the focus of the evaluation was: (a) to verify that the implemented tool was actually functioning as intended, (b) to identify and to correct usability problems of its user interface, (c) to assess its usability and usefulness, as perceived by potential end-users, and (d) to elicit user comments for improving the current design and extending the system’s functionality. In order to focus on the tool, rather than the rules, only a small sample was used. The rules were selected in such a way that every aspect of the system’s functionality could be explored during the evaluation.

The evaluation was an assessment of the subjective opinion of the eight users regarding the tool. A number of instruments were investigated, including the QUIS questionnaire [26], the SUMI questionnaire [27] and the IBM Usability Satisfaction Questionnaires [28]. The selected technique was the IBM Usability Satisfaction Questionnaires. These



Sherlock: Online Rule Help

Table of Contents

Sherlock Heuristics

```
|---- 1. Layout
|-------- 1.1 Labels
|------------ 1.1.1 Text in all capital letters
|------------ 1.1.2 Missing colon ':'
|------------ 1.1.3 Alignment of labels
|-------- 1.2 Buttons
|------------ 1.2.1 Location of OK and Cancel
|------------ 1.2.2 Button size
|---- 2. Color
|-------- 2.1 Avoid red & blue combination
|-------- 2.2 Avoid white & yellow combination
|---- 3. Wording
|-------- 3.1 Labels
|------------ 3.1.1 Avoid long sentences
```

Fig. 10. Rules help module.

questionnaires constitute an instrument for measuring the user's subjective opinion in a scenario-based situation. Two types of questionnaires are typically used; firstly, After-Scenario Questionnaire (ASQ), is filled in by each participant at the end of each scenario (so it may be used several times during an evaluation session), while the other one, namely Computer System Usability Questionnaire (CSUQ) is filled in at the end of the evaluation (one questionnaire per participant).

The primary criteria used to select the IBM Computer Usability Satisfaction Questionnaires as opposed to other questionnaires, include the following. Firstly, these questionnaires are available for public use, whereas the alternatives require the acquisition of a license from their vendors. Secondly, and perhaps most importantly, the IBM Computer Usability Satisfaction Questionnaires have shown to be extremely reliable (0.94). With respect to SUMI, it should be mentioned that it is equally reliable, but this reliability is derived from an increased number of questionnaire items. Thirdly, the IBM Computer Usability Satisfaction Questionnaires do not require any special software since they are

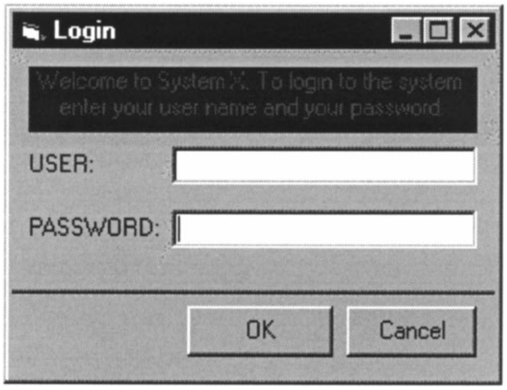

Login
Welcome to System X. To login to the system enter your user name and your password
USER:
PASSWORD:
OK
Cancel

Fig. 11. Sample login interface used during evaluation.

not computationally demanding. Fourthly, the time required to analyse the results is substantially less in the case of the IBM Computer Usability Satisfaction Questionnaires. Finally, another important determinant was the fact that the IBM Computer Usability Satisfaction Questionnaires have been in use for several years now at various industrial cites and research centres.

The result of the subjective evaluation with the IBM Computer Usability Satisfaction Questionnaires is a set of psychometric metrics which can be summarised as follows:

- ASQ metric provides an indication of a participant's satisfaction with the system for a given scenario;
- OVERALL metric provides an indication of the overall satisfaction score;
- SYSUSE metric provides an indication of the system's usefulness;
- INFOQUAL metric is the score for information quality;
- INTERQUAL metric is the score for interface quality.

The design of the evaluation followed a two phase process. The first phase involved the selection of a suitable technique (see above), the preparation of suitable scenarios and instruction materials, and the identification of the subjects. The second phase involved the collection and analysis of data, the identification of usability problems and the compilation of a set of recommendations. A small post-evaluation questionnaire was also used to provide information on the background of the subjects.

4.1. Evaluation procedure

All subjects were provided with the same software and accompanying material and each of them filled in the post-evaluation questionnaire. Two different simple interfaces were constructed in the VB 5.0 IDE. The first one, was a *login form* (Fig. 11). The second one was a data entry form, from a *phone directory* application (Fig. 12).

Sherlock was equipped with a set of nine rules, belonging to four different classes,

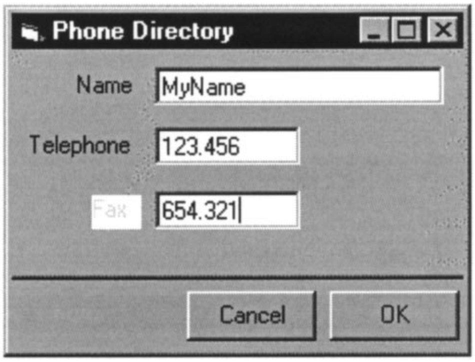


Fig. 12. Sample phone directory interface used during evaluation.

namely, Layout, Colour, Wording and Feedback. All these rules were applicable to the interfaces built. The rules are presented in Table 9. At first, a five minute demonstration of the Sherlock GMS was given to each subject separately. Then, each subject had to execute the two scenarios described below.

Scenario 1: Minimal interaction with the system
The subject was presented with a VB 5.0 IDE in which the *login form* user interface was already loaded. Before activating Sherlock, each subject was asked to inspect the user interface using their own experience and write down on paper any usability problems identified. Then, each subject had to activate Sherlock, connect to the server and ask for evaluation. When the evaluation data was received, the subject had to try to fix the problems reported, without having to report how this was done. Finally, when the subjects considered that they had successfully corrected the problems reported, they had to re-submit the interface to evaluation and interpret the report received.
Scenario 2: Interaction with various parts of the system
The subject was presented with the second user interface, and once again was asked to evaluate it, without Sherlock's assistance, and write down their assessment. Then, after activating Sherlock, each subject had to use the *rules handler* to deactivate any rules considered irrelevant, as well as the rules having a '*Semi-automatic*' or '*By the user*' inspection type. After that, each subject would issue an evaluation request.

When the evaluation data was received, the subject had to try to fix the problems reported, but also provide information to the system about how this was accomplished, following Sherlock's functionality as described in Section 3.4.5. The subject also had to review history and background information about at least two of the problems reported. During the execution of the scenario, the subject had to classify a problem as '*Not Applicable*', but later move it back to the list of '*Active*' problems. Finally, when the subject considered that all problems reported were corrected, the subject had to re-submit the interface to evaluation and interpret the report received. In case that the same or new problems appeared in the list, the subject was allowed to iterate through the same process, for one more time.

Table 9
Rules used for evaluation

Title	Description	Class	Inspection type
1.1.1 Text in all capital letters	Do not use text in all capital letters	Sherlock Heuristics/ 1. Layout/ 1.1 Labels	Automatic
1.1.2 Missing colon ‘:’	Field labels should be followed by a colon (:)	Sherlock Heuristics/ 1. Layout/ 1.1 Labels	Automatic
1.1.3 Alignment of labels	Labels should be left aligned	Sherlock Heuristics/ 1. Layout/ 1.1 Labels	Automatic
1.2.1 Location of OK and Cancel	‘OK’ should always be either to the left or over ‘Cancel’	Sherlock Heuristics/ 1. Layout/ 1.2 Buttons	Automatic
1.2.2 Button size	Buttons that are at the same level should have the same size	Sherlock Heuristics/ 1. Layout/.1.2 Buttons	Automatic
2.1 Avoid red & blue combination	Do not show pure red and blue together	Sherlock Heuristics/ 2. Color	Automatic
2.2 Avoid white & yellow combination	Do not show yellow on white	Sherlock Heuristics/ 2. Color	Automatic
3.1.1 Avoid long sentences	Avoid using too long sentences	Sherlock Heuristics/ 3. Wording/ 3.1 Labels	Semi automatic
4.1 Feedback provision	Always provide feedback to the user	Sherlock Heuristics/ 4. Feedback	By the user

Table 10
After-scenario questionnaire (ASQ) results (range from 1—highest—to 7)

	User 1	User 2	User 3	User 4	User 5	User 6	User 7	User 8
Scenario 1	2	3	3.33	1.33	2.33	3	3.67	2.67
Scenario 2	2	3.33	3.67	2	3	3	4	3.33

4.2. Evaluation results

The evaluation results are summarised in Tables 10 and 11. The slight increase in the scores observed between the two scenarios in Table 10, reflects their difference in required user interaction load.

The overall attitude of the users towards the system was positive, as can be seen in Table 11. Since none of them had used any similar tool in the past, their main comment was that this type of tool is very useful for their task and that they would undoubtedly use one, if it was to become available.

The strong points of the system were found to be its ease of learning, ease of use and overall efficiency and effectiveness in completing the evaluation scenarios. Although the users found the system's interface to be pleasant and intuitive, they also offered helpful comments towards further simplifying and enhancing it, which will be used in upgrading the system.

The identified weak points of the system mainly concerned the limited documentation and on-line help facilities provided. This was a known shortcoming of the prototype system, attributed to restricted resources at development time, which dictated very limited and focused help facilities. Another identified weakness, was the need for a comprehensive undo facility. Furthermore, some of the users had specific requests for additional functionality and system capabilities they would have liked to see supported in future versions of the system.

In general, the evaluation offered valuable insight into the functional and the interaction characteristics of the system and reinforced the belief that there is an actual need and demand for computerised tools for working with guidelines to support the tasks of user interface design and evaluation.

5. Discussion and concluding remarks

The Sherlock GMS is an attempt to integrate several concepts and approaches into an

Table 11
Computer system usability questionnaire (CSUQ) results (range from 1—highest—to 7)

	User 1	User 2	User 3	User 4	User 5	User 6	User 7	User 8
SYSUSE	2	2.75	2.63	1.38	1.63	2.13	2.63	2.25
INFOQUAL	2.86	4	4.43	1.86	3.14	3.57	4	3.23
INTERQUAL	2	3	2	2.33	2	4	3	2.67
OVERALL	2.3	3.21	3.16	1.74	2.26	3	3.05	2.58

environment for working with guidelines. The work was motivated from a pragmatic inquiry into the content and scope of a wide range of guideline reference manuals, style-guides and design heuristics, as well as previous efforts, reporting shortcomings and obstacles in the use of guidelines.

There is common ground shared between Sherlock and other available tools for working with guidelines, but there are also distinctive differences. The most prominent characteristics of Sherlock in this respect, relate to its orientation towards user-centred design and include: (i) the facilities offered for the support of iterative development life-cycle; (ii) the provision of early and direct evaluation feedback; (iii) the provision of an integrated tool environment allowing designers, developers and usability experts to collaboratively address the demanding issue of usability in modern software engineering.

Most importantly, however, Sherlock extends the scope of previous attempts to provide tools for working with guidelines by offering explicit support for contextualising guidelines, developing corporate styleguides, facilitating design memories and traces of past experience. In summary, Sherlock is a tool fostering knowledge persistence and evolution in the area of human factors and guidelines.

The evaluation of the system confirmed the above claims and revealed the added value resulting from the integration of such tools in the organisational structure and their effective use to serve real design problems. It is believed that such tools are highly useful for the early phases of user-centred design and hold the promise of substantially improving upon current practice. Moreover, such benefits need not necessarily be appropriated by a particular type of tool, but from a collection of instruments aiming to provide comprehensive support for exploratory design activities and tight design–evaluation–(re)design feedback loops. On-going work aims towards facilitating such an objective by integrating Sherlock with other design augmentation tools, currently under development, to offer more comprehensive support for design rationale, evaluation and usability assessment.

References

[1] S.L. Smith, J.N. Mosier, Guidelines for designing user interface software, Report No. MTR-10090, ESD-TR-86-278, Bedford, MA:MITRE Corp., 1986.

[2] Open Software Foundation, OSF/Motif Style Guide, Prentice-Hall, London, 1993 Revision 1.2.

[3] Microsoft, The Windows™ Interface Guidelines for Software Design, Microsoft Press, Redmond, WA, 1995.

[4] Apple Computers, Macintosh Human Interface Guidelines, Addison Wesley, Reading, MA, 1992.

[5] IBM, Object-Oriented Interface Design: IBM's Common User Access Guidelines, QUE, 1992.

[6] S. Henninger, K. Heynes, M. Reith, A framework for developing experience-based usability guidelines, in: Conference Proceedings of DIS'95, University of Michigan, August 23–25, ACM, 1995, pp. 43–53.

[7] S. Henninger, C. Lu, C. Faith, Using organisational learning techniques to develop context-specific usability guidelines, in: G. Van der Veer, A. Henderson, S. Coles (Eds.), Conference Proceedings of Designing Interactive Systems: Processes, Practices, Methods and Techniques (DIS'97), Amsterdam, August 18–20, ACM Press, New York, NY, 1997, pp. 129–136.

[8] L. Tetzlaff, D. Schwartz, The use of guidelines in interface design, in: CHI'91 Conference Proceedings Human Factors in computing Systems, New Orleans, Louisiana, April 27–May 2, 1991, pp. 329–333.

[9] N. Bevan, N. Macleod, Usability measurement in context, Behaviour and Information Technology 13 (1/2) (1994) 132–145.

[10] J. Lowgren, T. Nordqvist, Knowledge-based evaluation as design support for graphical user interfaces, in: ACM CHI'92 Conference Proceedings, 1992, pp. 181–187.
[11] J. Vanderdonckt, Accessing guidelines information with SIERRA, in: INTERACT'95, 1995, pp. 311–316.
[12] K. Ogawa, K. Useno, GuideBook: design guidelines database for assisting the interface design task, SIGCHI 27 (2) (1995) 38–39.
[13] R. Iannella, HyperSAM: a management tool for large user interface guideline sets, SIGCHI 27 (2) (1995) 42–43.
[14] P. Reisner, Formal grammar and human factors design of an interactive graphics system, IEEE Transactions on Software Engineering SE-7 (2) (1981) 229–240.
[15] T. Blesser, J. Foley, Towards specifying and evaluating the human factors of user-centered interfaces, Proceedings of ACM Conference on Human Factors in Computing Systems (CHI'82), ACM Press, New York, NY, 1982 pp. 309–314.
[16] H. Reiterer, IDA: a design environment for ergonomic user interfaces, in: INTERACT'95, 1995, pp. 305–310.
[17] P. Gorny, EXPOSE: An HCI-counselling tool for user interface design, INTERACT'95, 1995, pp. 297–304.
[18] G.B. Silverman, Building a better critic: recent empirical results, IEEE Expert April (1992) 18–25.
[19] D. Nitsche-Ruhland, D. Zimmermann, CritiGUI—knowledge based support for the user interface design process in Smalltalk, in: B. Blumenthal, J. Gornostaev, C. Unger (Eds.), Human–Computer Interaction, Proceedings of the 5th International Conference, EWHCI95, Moscow, Russia, July 3–7, Springer, Berlin, 1995, pp. 179–188.
[20] G. Fisher, A. Lemke, T. Mastaglio, A. Morch, Using critics to empower users, in: CHI'90 Conference Proceedings, 1990, pp. 337–347.
[21] U. Malinowski, K. Nakakoji, Using computational critics to facilitate long-term collaboration in user interface design, in: CHI'95, 1995, pp. 385–392.
[22] C. Farenc, V. Liberati, M.-F. Barthet, Automatic ergonomic evaluation: what are the limits?, in: J. Vanderdonckt (Ed.), Proceedings of the 2nd International Workshop on Computer-Aided Design of User Interfaces CADUI'96, Namur 5–7 June 1996, Presses Universitaires de Namur, Namur, Belgium, 1996, pp. 159–170.
[23] A. Cohen, D. Crow, I. Dilli, P. Gorny, H.-J. Hoffman, R. Iannella, K. Ogawa, H. Reiterer, K. Ueno, J. Vanderdonckt, Tools for working with guidelines, SIGCHI 27 (2) (1995) 30–32.
[24] ISO 9241, Ergonomic requirements for office work with visual display terminals, International Standards Organisation, 1994.
[25] R. Jeffries, Usability problem reports: helping evaluators communicate effectively with developers Chapter 11, in: J. Nielsen, R.L. Mack (Eds.), Usability Inspection Methods, Wiley, New York, NY, 1994, pp. 273–294.
[26] B. Schneiderman, Designing the User Interface: Strategies for effective Human–Computer Interaction, 2, Addisson Wesley, Reading, MA, 1992.
[27] J. Kirakowski, M. Corbett, SUMI: the software measurement inventory, British Journal of Educational Technology 24 (1993) 210–212.
[28] R.J. Lewis, IBM computer usability satisfaction questionaires: psychometric evaluation and instructions for use, International Journal of Human–Computer Interaction 7 (1) (1995) 57–78.