PŘÍRODOVĚDECKÁ FAKULTA UNIVERZITY PALACKÉHO
KATEDRA INFORMATIKY

DIPLOMOVÁ PRÁCE

Adaptivní uživatelské rozhraní „Boulevard“ v textovém editoru

Adaptive User Interface “Boulevard” for Word Processor

2011 Zdenek Eichler

Anotace

Aplikace nabízejí uživatelům stále více funkcionality. Uživatelská personalisace aplikací je jeden z nástrojů k řešení tohoto problému. Tato diplomová práce je zaměřená na vývoj Boulevardu — adaptivního kontejneru akcí, implementovaného v textovém editoru OpenOffice.org Writer. Boulevard se řadí mezi adaptivní uživatelská rozhraní. Boulevard funguje na principu sledování chování uživatele pomocí loggeru nazvaném Interceptor a je řízen expertním systémem. V závěru práce je popsán také test použitelnosti Boulevardu.

Annotation

Applications are growing in terms of the offered functionality. The personalization has been proposed to address the issue. The purpose of this thesis is development of Boulevard, an auto-adaptive container, implemented in the OpenOffice.org Writer. Boulevard, an adaptive user interface, collects and evaluates information about the user's behavior by a logger called Interceptor. Boulevard is controlled by the expert system, which perform the adaptation. Finally, we performed a usability test of Boulevard.

Děkuji mamince a tatínkovi, kralce a kloučkovi. Děkuji mé sestře. Děkuji Adélce. Děkuji svému „vůdci“ Martinu Dostálovi.

Contents

List of Figures

List of Tables

1 Introduction

Usability of computers from user's perspective tends to decrease instead of increase. Reportedly, the excess of provided functionality is a substantial source of dissatisfaction [44]. We are talking about *software bloat* [39], *creeping featurism* or *feature war*. All of these terms are a little different in meaning, but they describe the same phenomenon: a trend of newer software applications to require more resources like a faster CPU and more memory and provide a high number of unused features by most of users. Human-Computer Interaction (HCI) community is particularly interested in such a phenomenon and there have been some attempts to address the above-mentioned phenomenon, such as personalization, adaptive user interfaces, recommender systems, etc. Our interest focuses on word processing applications, as they are used by various types of users, from beginners to professionals and as the above-mentioned phenomenon affect word processors as well. This trend is obvious from Figures 2 and 3.

1.1 Aims

We aim to implement an adaptive container of personalized user interface named Boulevard in a word processor and to verify its applicability by usability tests. Boulevard has come into existence in our previous work focused on adaptive user interfaces supervised by Martin Dostál, who I and Jakub Černek worked with. This work continues in the development of Boulevard, and provides new, a more advanced, implementation of Boulevard. We consider the implementation of Boulevard in an existing "bloated" application as particullary important for more relevant testing of its usability in therms of ecological validity[1].

Thesis is organized as follows: Details about Boulevard are in Chapter 3. Why we have decided to use OpenOffice.org Writer is explained in Chapter 6. An important part of work on every adaptive user interface is a user activity logger. We developed our own logger called OpenOffice.org Interceptor, further referred to as Interceptor, for more details see Chapter 7. The last part of work is a usability test of Boulevard (Chapter 10), which was not performed in the previous work properly.

1.2 What is Boulevard

Boulevard, an adaptive container, is a new user interface element (an addition to the menus and toolbars), which automatically collects and orders the user commands frequently and/or and recently used. The interaction styles and parameters used to activate commands are also considered. Since Boulevard's user interface changes automatically without a direct user's invitation, therefore

[1]The study must follow the real-life conditions. Do not confuse with external validity.

Boulevard ranks among adaptive user interfaces. Current Boulevard implementation is interesting also from an engineer's point of view, since the integration of Boulevard in the OpenOffice.org Writer. An important part of Boulevard is a logger, which we consider as the fundamental part of any adaptive user interface. Our logger called OpenOffice.org Interceptor is unique for several reasons: (1) combines both common used logging techniques: macro-recorder-based logging and user interface events (and accessibility API)-based logging. (2) The interceptor provides an Application Programming Interface (API), which makes the real-time interaction with an adaptive user interface or user testing software possible.

1.3 Brief History of User Interfaces

To show the current status of user interfaces and their problems, it is good to know about their history in the first place. The very first type of user interface was a batch interface, which was not interactive at all, the user only entered the parameters for a batch task and after the task was completed the interface showed the result. There was no more interaction with the user. Such an interface was used strictly by expert users. This user interface was typical for mainframes and nowadays it is a "dead" interface.

Another important type of user interface is a command line, which was introduced in 1969, when telnet was introduced in RFC 15 [19]. The origin of a command line interface came from a teletypewriter (TTY). These interfaces served not only for message switching, but also as the first remote terminals for mainframes. This already implies a first real interaction between user and computer. Nowadays, command line interfaces are used by experienced users, and mostly in professional applications, such as Cisco systems IOS (Internetwork Operating System). A command line is also used by interactive program languages, e.g., LISP. However, the command line is not very user-friendly, because the user has to remember commands and keystrokes, which are mostly different in every system. On the other hand, the command line is believed to be faster than GUI (Graphical User Interface) and also is easy to implement, which might be the reason why it is still a favorite interface for professionals and it is used in simple embedded devices.

A breakthrough the from user's point of view was the Graphical User Interface (GUI), especially WYSIWYG (What You See Is What You Get). The Development of WYSIWYG text editors started in 1974, when Xerox developed the Bravo text editor, the very first text editor which displays text with formatting [26]. Bravo was designed for the Alto computer, which was never sold commercially, but served as an starting-point for the Xerox Star computer introduced in the 1981. The Star was a major breakthrough in user interfaces. It was document-centric, so the user did not interact with the program but with the document, of course from the user's point of view. The text editor contained in the

Star computer was based on Bravo; moreover it was controlled by icons, therefore the user did not have to remember dozens of creepy shortcuts and commands, but could click directly on the icon representing such a command, which was a big breakthrough. This new interface reportedly greatly reduced the learning time needed to start using the computer [49]. Xerox was not only a great innovator of user interfaces, but also has invented a lot of today used technologies, such as the Ethernet.

Most of today's applications use the WIMP (Windows, Icons, Menus and Pointing Device) paradigm, which was developed by Xerox, widely popularized and a further improved by Apple Macintosh released in 1984. WIMP together with the direct object manipulation and WYSIWYG is much more user-friendly than a command line. After that there has not been much innovation in user interfaces. WIMP is still the most popular interface today. In earlier systems, there were many implementation issues with the GUI, so the development was often oriented from technology to a user interface (does not concern the Apple and the Xerox). However, with today's technologies we can reverse the development: from a user interface to technology, which increases the chances to develop a good user-friendly interface.

1.4 Ribbon User Interface

After Microsoft Office 2003, Microsoft obviously realized that there is no way to add more items to toolbars or menus. Microsoft performed an extensive but unpublished research on how users use the Office suite, which resulted in the Ribbon User Interface, see Figure 1 (image by Richard Ericson). Ribbon User Interface is organized in tabs containing icons, simple menus and some parts of dialogs. In the WIMP paradigm, there is one user command available at multiple places (for example, in a toolbar and a menu), in contrast to Ribbon, where is one user command available only at one place. This reduces the visual complexity. Another important Ribbon feature is context behavior. The screen content, especially an opened tab changes according to the current context (the cursor position in the document). Ribbon interface belongs to static interfaces, even through there is some kind of dynamic behavior, like context sensitivity.

There was a study on the user acceptance of the Microsoft Ribbon user interface performed by Martin Dostál [27]. Dostál concludes that Ribbon user interface is well accepted by a new users to Microsoft Office, on the other hand, users who switched from previous versions of Microsoft Office found Ribbon user interface as worse than previous one (WIMP-based).

1.5 The Current Status - What is the Edge of the WIMP Paradigm?

Today's applications are much extensive and more complex than before. How-

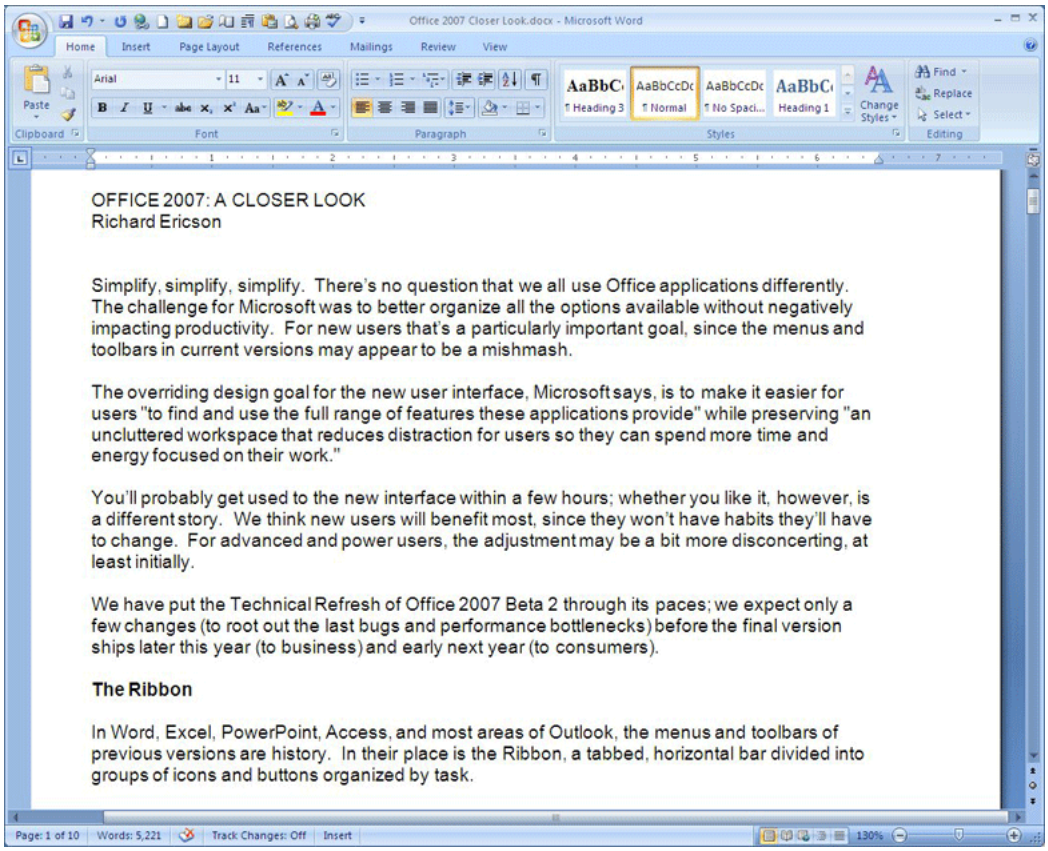


Figure 1: The Ribbon User Interface

ever, in my opinion, WIMP was designed for much smaller applications than we have today. Thanks to today's high-resolution displays and high-performance processors we can now have too many features and items on the screen. Increasing the visual complexity is a problem for users, since they just lose themselves in all the offered functionality. The applications loses its original elegance. In this context, we talk about two phenomenons:

- Increasing demands on the hardware (hardware requirements).
- Increasing functionality of the software.

The first phenomenon is not much an issue, since the development of hardware is always ahead. The second phenomenon already appears to be a problem since users are not getting better every year like hardware. Users lose themselves in the massive functionality offered by software. The growing functionality of software seems like a good thing at first sight: to offer users more options and features than the previous versions. However, there should be some limitations. The common causes for the increasing functionality are several, for instance:

- Applications have often long history, their development lasts for decades. Functionality is more often added than removed. Remember the Figures 2 and 3.
- Marketing reasons induces the need to offer more functionality than the previous version and competing products.
- Widely-used applications, such as word processors, are intended for many diverse users.

Figures 2 and 3 (by Martin Dostál) expose the development of the toolbars and the menus size in Microsoft Word in time. The question is, what is the limit? An interesting study focused on Microsoft Office users and bloatware was done by McGrenere and Moore, with a poignant title "Are We All in the Same Bloat?" [44]. Another interesting study, done by Kaufman and Weed, has a name: "Too much of a good thing?: identifying and resolving bloat in the user interface" [39].

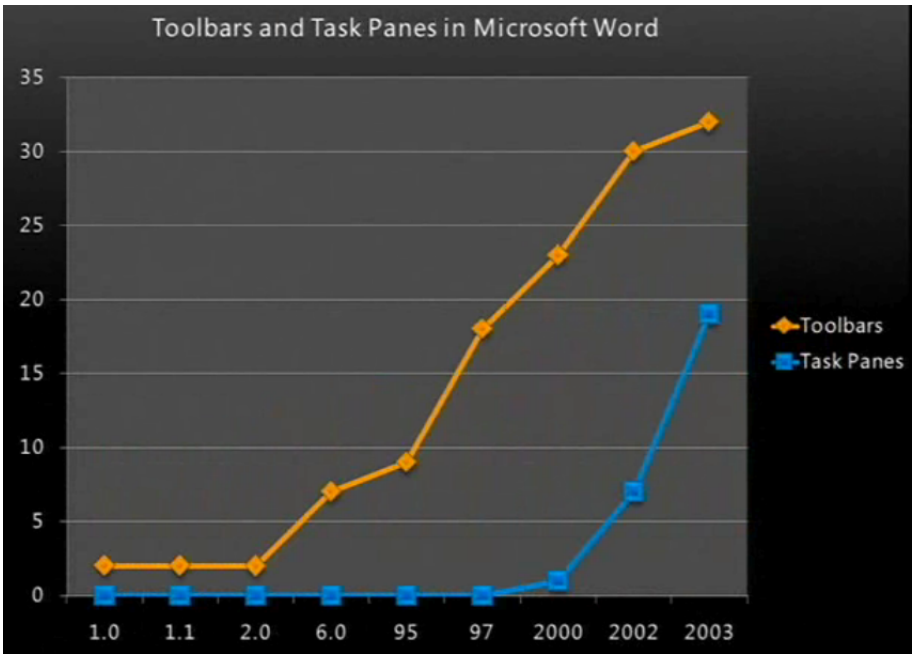


Figure 2: Toolbars growth in Microsoft Word

1.5.1 Level Structured Interfaces

One approach to achieve better usability of complex software is reducing functionality by offering different versions of the software. One version can focus on beginners and another at professionals. Possible is also offer "version" behavior

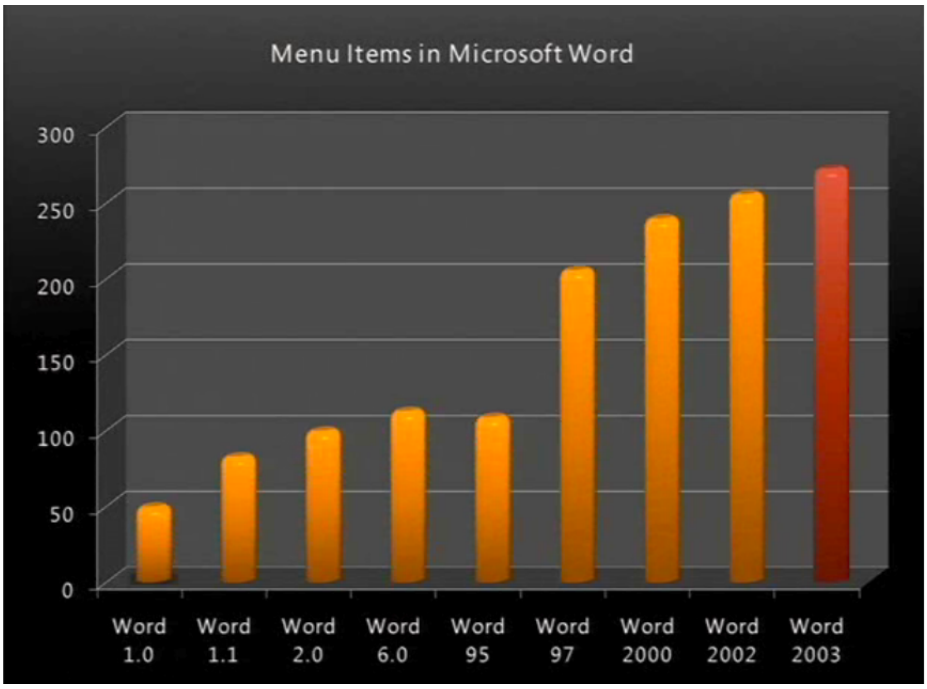


Figure 3: Menu growth in Microsoft Word

in one application at the same price. Such an application must provide an interface to change the complexity of the user interface. The application can then "grow" with the user. A similar approach was used by Joanna McGrenere in [43], where she introduced the so-called Multiple interface, which is demonstrated on the Microsoft Word 2000. The Multiple interface concept provides easy switching between two user interface versions: a full, not personalized, interface and a limited personalized interface. She found that only 20 % of participants marked the full, not personalized interface, as their first choice. The disadvantage of this approach is that there is probably no such thing as an "average user" and a groups of users. These findings resulted from the survey on word processors users performed by Martin Dostál [29]. Joel Spolsky uses the Vilfredo Pareto's 80/20 rule and claims that: *"80% of people use 20% of features. Unfortunately, it is never the same 20%"* [50]. There is a problem how to build a version ladder; which feature is intended for a beginner or an advanced user, as we know that users are so different. Another problem with the version control is the switching between versions. In commercially different versions a user must buy a new version of the product.

1.6 Personalization

Other approach is personalization; we differentiate between two basic kinds of personalization: adaptive and adaptable. Both adaptable and adaptive ap-

proaches are about customization of software from its default configuration. The adaptable approach stands for customization performed by the user and whereas the adaptive approach stands for customization performed by the computer. Just for precise understanding, by customization we do not understand a customization of software by developers, for example, like we "customized" OpenOffice to support Boulevard.

1.6.1 Adaptable User Interfaces

The adaptable customization is widely supported by today's applications, e.g., Microsoft Office 2003 and OpenOffice.org. Although many applications today support personalization, we believe that most users do not personalize. There was one interesting study on how many users really personalize word processors and how many users know how to personalize. Such a study has been done on the WordPerfect word processor; they concluded that 92% of users perform customization, however, they consider even a change of zoom level as customization, not only changes to the user interface [46]. Another study was performed on UNIX users, how they customize [42]. Just to remind, customization in UNIX systems is usually performed through editing configuration files, which is not very user-friendly. The outcome of the study was that only a small number of users perform customization, mostly because it is too hard to accomplish.

Advantages of an adaptable user interface:

- The user has everything under control, no change can happen in the user interface without user's direct invitation.
- Most software is customizable.

Disadvantages of an adaptable user interface:

- Personalization is mostly difficult and a less experienced user does not have to know how to do it. Some users do not know that such a thing is even possible. In a study focused on text-editor users it was found that programmers use customization much more than secretaries (today referred to as personal assistants) [47]. This causes a paradox: Less experienced users need to reduce the visual complexity, and thus customize more than advanced users; but they usually do not know how to do that.
- Personalization is not a simple task and costs user's time and energy, therefore the user can decide not to perform personalization [42]. This disadvantage can be avoided by making customization easier. An example of not a user-friendly customization is OpenOffice.org, where the user must edit a list, which is later interpreted by OpenOffice.org as a content and layout of toolbar. A good example of a user-friendly customization is Apple Pages, where the user can perform customization by direct manipulation with objects.

- User's needs can change in time, thus repeated customization is needed.
- Users switch can cause very unpleasant and painful experience in the case when a user starts using someone else's personalized software, especially for less experienced users.

1.6.2 Adaptive User Interfaces

By adaptivity we understand an automatic customization of a user interface without direct user's invitation, operated by the computer. There is usually some kind of "intelligence" behind.

There are more approaches of Adaptive User Interfaces (AUIs) distinguished by its purpose and motivation. We consider as AUI adaptation of the user interface motivated by better usability to the regular users.

There is another approach, adaptation of the user interface to a device or a platform, such interfaces are sometimes called multi-platform or meta-interfaces. A meta-language is often used for describing the user interface model, for example UIML, XIML, ISML or XUL. The list of the most such meta-languages can be found in [8], the list contains about 40 such languages, which are mostly similar, XML-based. A meta-language serves only as a descriptor of a user interface model, but there must be some kind of intelligence (algorithm) behind, which transforms the meta-language to a particular user interface. This approach has growing popularity in HCI community, probably because of growing numbers of various mobile devices. An example of such software is ADUS (ADaptive User interface System) [1].

Another important approach is to make a user interface more usable for people with disabilities. This task is difficult, since there are many types of disabilities and their combinations. An example of such an interface is SUPPLE [14].

The advantages of adaptive user interfaces:

- No skill requirements for a user to do the personalization. Such personalization could be, thanks to its automatic behavior, usable for a less experienced user.
- Such personalization can be done more often and without cost of time and energy.
- In case of a user switch, or fundamental changes in the user's behavior, the software adapts (customizes) the interface automatically to the new situation.

The disadvantages of adaptive user interfaces:

- The user can be confused and may not understand, what is happening with the interface. Such behavior is sometimes referred to as annoying and is considered as the main problem in adaptive user interfaces.

- The user interface, which is constantly changing, is hard to learn. The user must look for commands all the time, which costs time and energy.

There is another kind of intelligent user interfaces, recommender systems [37]. In the context of user interfaces they are often called agents. Such an agent observes the user's behavior and habits. Agents usually do:

- Offer to the user the potentially useful functionality.
- Detect the user's goals and provide some kind of help (e.g., wizard).
- Detect the user's workflow and offer automation of such routines.
- Teach the user new skills, we call such an agent pedagogical.

A common difference between AUI and an intelligent agent is that AUI offers the user functionality used by the user and changes the user interface, while an agent offers the user functionality not used by the user, which can be useful for the user, and does not change the user interface. However, there are also many combinations of both approaches. An example of a recommender system, which is not represented as an agent, is the Google's "Did you mean" feature.

Microsoft Office Assistant The Microsoft Office Assistant (also know as Clippy, see Figure 4) is probably the best known intelligent agent, the most hated at the same time, which was a target of various jokes. Smithsonian Magazine called Clippy "one of the worst software design blunders in the annals of computing" [23]. Clippy was not well accepted by users and was removed from following versions of Office. One reason can be the "smart guy" factor. To our knowledge, there is not any study focused on Microsoft Office Assistant evaluation.

The most frequent argument against AUIs is that automatic adaptation can disorient the user, and thus is worse than good for the user. We will describe the current status and interesting Adaptive User Interfaces with examples.

Split menus Split menus (Figure 5) is not an adaptive user interface, but static. We are mentioning split menus here since they inspired a lot of adaptive user interfaces related to the menu adaptation. Split menus, introduced by Sears and Shneiderman [48], divide a menu into two sections: the high-frequency section contains the most frequently used items and should not contain more than four items. The low-frequency section contains all other items. Items in both sections should be sorted by frequency, when there are not many items, and alphabetically when there are many items. Sears and Shneiderman performed an evaluation and found that split menus were not worse than ordinary menus and in most cases split menus were even faster. Split menus inspired also other items than menus, for example the drop-down list in some software, such as the OpenOffice.org Writer. Similarity to split menus is obvious, see Figure 6 .

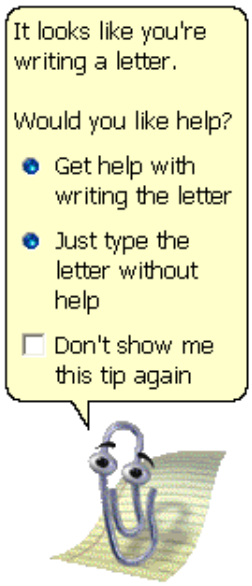


Figure 4: Microsoft Office Assistant

Microsoft Adaptive Menus In Microsoft Office 2000 Office Assistant (Clippy) was removed and new adaptive menus (Figure 7 source: silicoholic.com) were introduced, the first commercial implementation of an adaptive user interface. The concept is similar to Split menus, but adaptive and there is a different display of the low-frequency section and the high-frequency section. Menus have two display modes: (1) an adapted mode with about a half or a third of items. (2) A Full mode with all menu items. The menu starts in the adapted mode and the user can switch to the second full-feature mode by a long observation of menu or by clicking on a special menu item with an arrow symbol. Items in adapted mode are selected and sorted by an observation of user's behavior but the exact algorithm is unknown. Arguments against Adaptive menus are: (1) a difficult start-up - which items should be placed into the first personalized mode, remember that there are big differences between users, (2) when user did not find the requested item in first mode, they must search for it again in the second full mode, which contains all items from the first mode, but in different order and mixed together with other unused items. The concept of the Adaptive menus was abandoned in Microsoft Office, just like Office Assistant, in Office 2007.

An interesting menu-related paper [35] compares static, adaptive and adaptable menus. They find out that an adaptive menu was slower than a static menu, an adaptable menu was not slower than a static one. Another menu-related work [16] describes an implementation of adaptive menus to the Eclipse (an integrated development environment), unfortunately, they did not perform the usability test on volunteers.

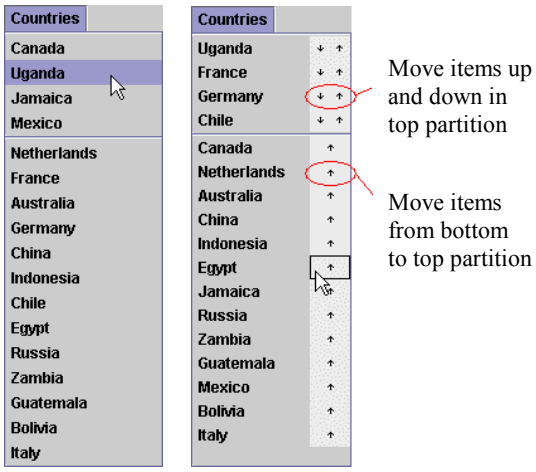


Figure 5: Split menus: static on the left and adaptable on the right

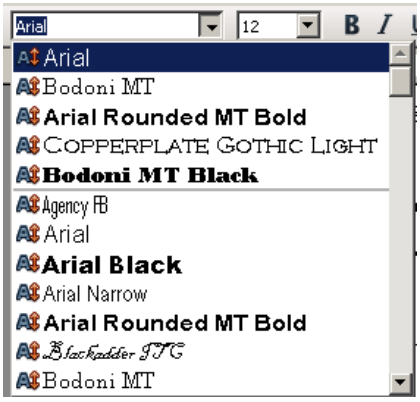


Figure 6: The font drop-down list in OpenOffice.org Writer

Aida In 1993 AIDA was presented [24], which was an adaptive system for interactive drafting and CAD systems. The system managed the contents of toolbars and menus; the user had two ways to add new items: (1) use a command from the command-line, such a command was added to the user interface. (2) The system observed user's behavior also in the drawing area — some recognized and repeated operations were also added to the user interface, as a macro.

Flexcel 1993 was a good year for adaptive user interfaces, because in this year FLEXCEL was also introduced [45], an adaptive user interface for Microsoft Excel, which was chosen for the same reason as why we have chosen the OpenOffice.org — to perform tests on real software, not a prototype. FLEXCEL was developed in two stages: FLEXCEL I and FLEXCEL II. The key features were:

- An adaptation toolbar — a toolbar dedicated to control Excel adaptation.

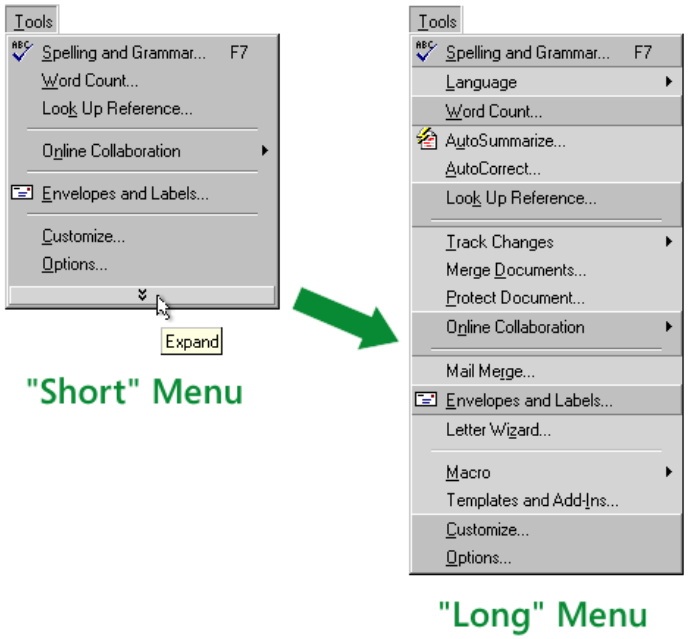


Figure 7: Microsoft Adaptive Menus

- Creating new menu items representing existing commands, but with the user defined parameters.
- A teaching agent, which tells the user tips and hints.

FLEXCEL is probably the most advanced adaptive interface in existing software, however, it was not well accepted by users. One reason was that users do not often personalize well, for example, when they were naming new menu items, they sometimes just quickly entered nonsense characters. FLEXCEL was using LISP rule based system with CLOS (Common Lisp Object System) objects. Because of poor interaction between Excel and LISP, the response to the system was very slow.

In another interesting study [38], they performed a study that examines the relative aspects of predictability and accuracy on usability of an adaptive user interface — adaptive toolbar in Microsoft Word. They conclude that the increasing predictability and accuracy strongly improved user's satisfaction. The above-mentioned we can summarize as some general statements:

- An adaptive user interface must be reliable, otherwise it is worse than no adaptation at all.
- To profit from the adaptation, there must be a considerable difference in frequency of use of commands.

- Users do not like the fully adaptive interfaces, they prefer mixed-initiative user interfaces. The control of the user interface must be very user-friendly.

2 Previous Work

The Boulevard implementation in this work is not the very first, but the second. The first implementation ([20] and [34]) was a prototype (Figure 8) of a simple RTF (Rich Text Format) editor with Boulevard, which we programmed together with Jakub Černek. This prototype was written in Common Lisp, using the CAPI (Common Application Programmer's Interface) [7] and LISA (Lisp-based Intelligent Software Agents) [6] libraries. We decided to make such a prototype mainly due to logging issues in common applications (see Chapter 7), which is not a issue in prototypes at all. The prototype was unfortunately not very stable, I guess mostly because of the expert system. Another drawback of the first prototype is more important: our prototype provided limited functionality, it was not "bloatware" at all. For example, the toolbar provided complete functionality of the RTF editor. Such a prototype was not suited for user testing in terms of ecological validity. However, this first prototype showed the big potential of Boulevard. So we decided to cut off the disadvantages of the first implementation, especially implementing Boulevard to the existing, "bloated" office suite.

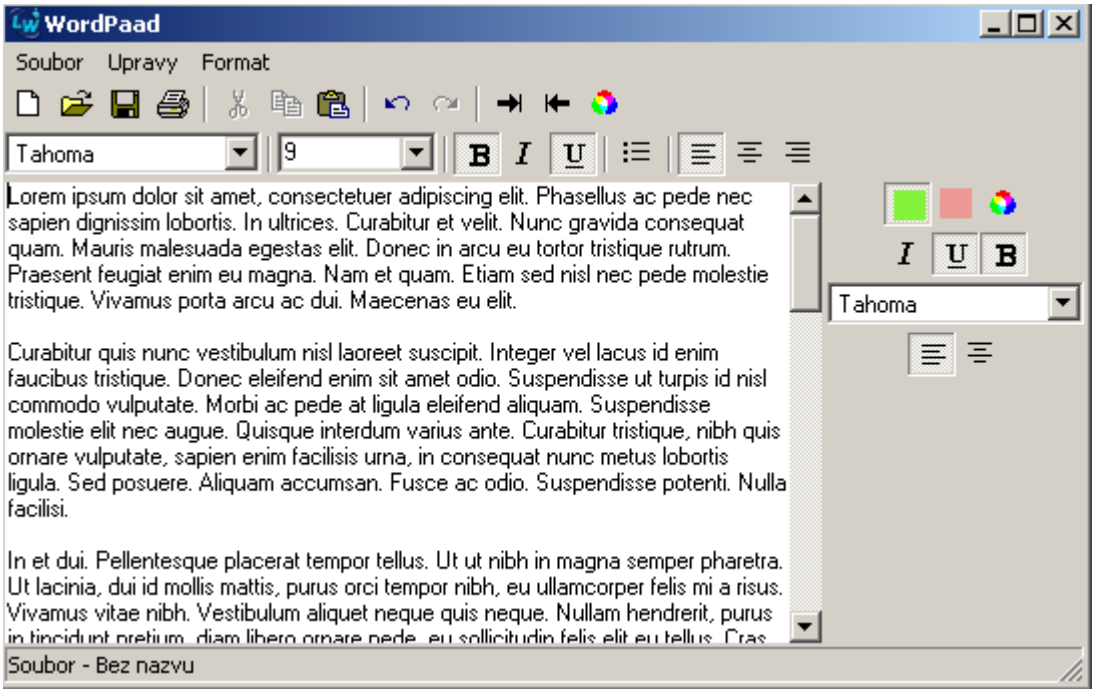


Figure 8: The first Boulevard prototype

3 Boulevard

Boulevard is a panel containing buttons, icons, sliders and many other well-known user interface elements. The content and layout of Boulevard are changing automatically without direct user a intervention and for the sake of better user's satisfaction. Boulevard does not contain all possible user commands provided by the application, that is why Boulevard is an additive to menus and toolbars, in contrast to a replacement of such. Menus and toolbars serve the user as a safety zone, where all functions are always available at the same place, position and visual representation.

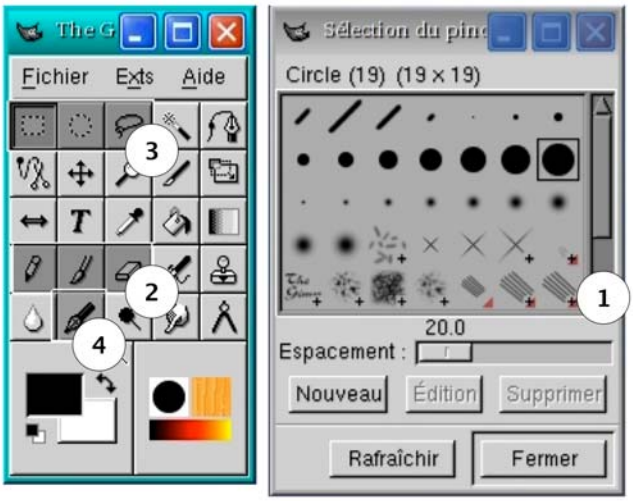


Figure 9: Façades Source Windows

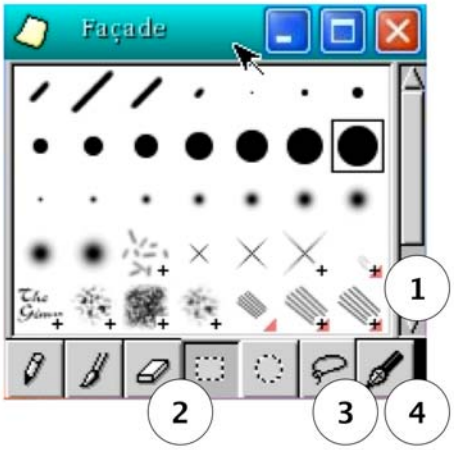


Figure 10: Façades Window

Boulevard was inspired by Façades [51], which is an adaptable interface. Façades is an add-on window where the user can place items from various places of screen (Figures 9 and 10) and then use the Façades window as a personalized area. Façades implementation is based on a Metisse server[2] [21] , where Façades window contains "pointers" to other areas of the screen. From the implementation come some advantages and limitations. A great advantage is that we can use Façades with any software under the X server, and from only a visual principle we have also synchronized status of commands (for explanation of context synchronization see chapter 9.3.1). The direct outcome is that Façades cannot contain an item (command) which is not currently displayed on the screen. In our opinion, we can avoid this limitation by using different screens; at least one for displaying tools and one as a area with Façades. Other screens serve only as visual sources for tools. So from here came the idea to make an add-on window inspired by Façades, but auto-adaptive. Boulevard got its name, since as Martin Dostál said:

> *"Today's cities are vast and complex, just like today's applications. Boulevard is a metaphor of a main street (boulevard), where the users can find everything they need."*

Boulevard orders items (user commands) in according to ranks. Each Boulevard item (user command) has assigned a rank — value between 0 and 1. Rank considers two factors: recency and frequency. The frequency factor holds the information about how frequently the user uses the command. The recency factor indicates how often a command has been used recently. Recency acts as an accelerator, which helps a rarely-used user command (from the frequency point of view) to get faster to a more prominent position if it has been used more often recently. But if such an accelerated command is used no more, it loses the recency accelerator and goes back to a less prominent position rapidly. Following formula represents the rank computation for item x:

$$rank(x) = w\frac{|x|}{|T|} + (1-w)\frac{\sum_{p_i \in P_x}(q - p_i + 1)}{\sum_{i=1}^{q} i} \tag{1}$$

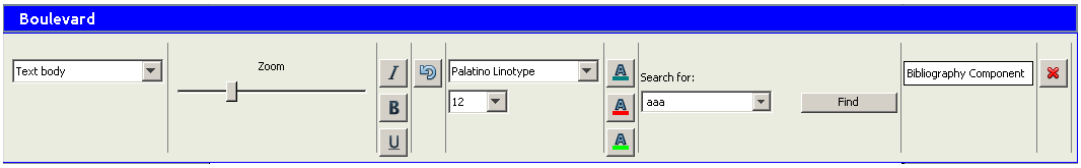
- w – (weight) is a value between 0 and 1 and indicates how much we care about the recency factor (0) or the frequency factor (1) (a balance between recency and frequency).
- $|x|$ – the number of activations of item (command) x
- $|T|$ – the total count of all activations of all items (commands)

[2]Metisse is a X Window system designed to make it easy for HCI researchers to implement new rendering techniques as well as novel window management techniques.

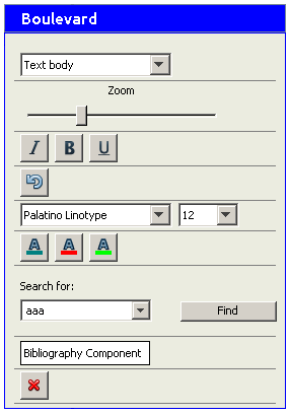
- q – the length of the recency queue (FIFO)
- P_x – a set which contains the positions of occurrence of the item (command) x in the queue (the front position of the queue is represented by value 1)

This formula come into existence by modification of previous version of formula, which was introduced in the Bsc. thesis. Formulas are not much different, but the current version is more balanced (in the frequency/recency point of view) than the previous one.

Boulevard provides two basic layouts: a vertical and a horizontal (Figure 11). One the horizontal layout sorts prominent places from left to right, the most prominent place is on the left (a toolbar style). The vertical layout sorts prominent places from top to bottom, the most prominent place is at the top (a menu style).



(a) Boulevard horizontal layout

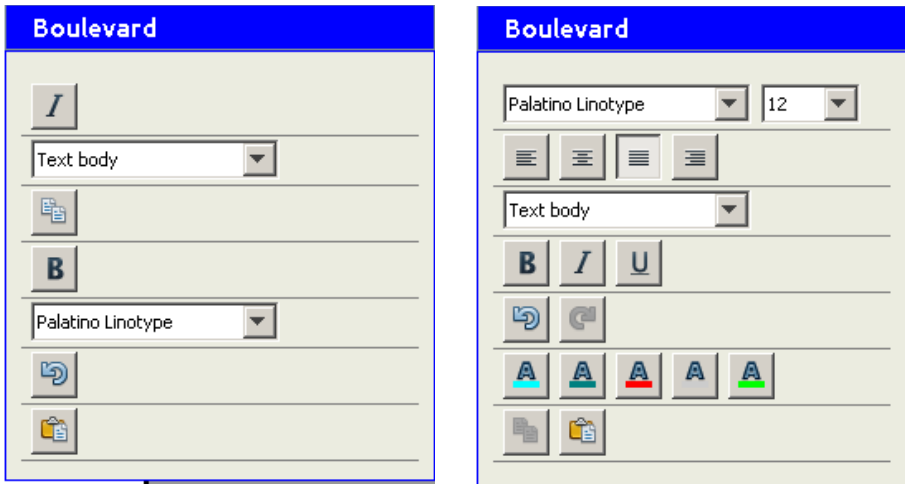


(b) Boulevard vertical layout

Figure 11: Possible layouts of the Boulevard

Boulevard also keeps similar commands together to help the user to perceive Boulevard. Boulevard has predefined groups of such commands. In one group there are usually semantically similar user commands. We call this feature *Sweeping back*. The Figure 12 depict Boulevard with disabled sweeping-back feature and with enabled sweeping-back feature.

We wanted to make Boulevard more usable, so we added another dimension — similar to Ribbon. A position of a user command in Boulevard is denoted by the primary and secondary position. Every primary position is subdivided to



(a) Sweeping back disabled (b) Sweeping back enabled

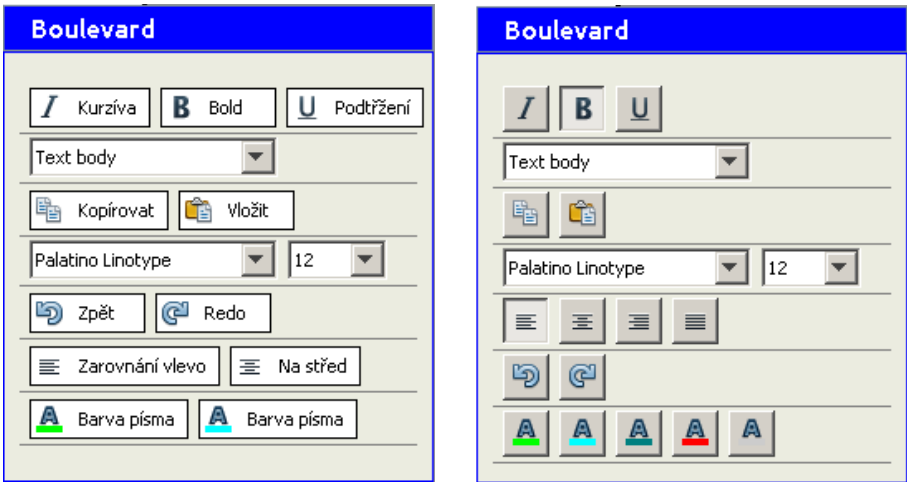
Figure 12: Sweeping back feature

the secondary-positions. The secondary-positions are currently used for sweeping-back feature — keeping semantically similar user commands together. These user commands are on the same primary position and are sorted by ranks on secondary positions. A list of such user commands and more details are in Chapter 5.2.4.

Boulevard also considers how user commands are used (from the user interface point of view),e.g., if to invoke the command a toolbar or a menu were used (further interaction style). This is called an adaptive representation (Figure 13). We use such information for choosing visual representation of an item (command) in Boulevard, which is familiar for the user. If a toolbar is preferred, the command is represented a toolbar button. If a menu is preferred, the command is represented as a menu-like button. A command activated by a keystroke is not displayed in Boulevard, since such a command is not required in Boulevard. Adaptive representation considers also the parameters of command. The example of such is the "Find" command. On the Figure 14 is depicted the original OpenOffice.org dialog. The Boulevard possible adaptations of the "Find" user command are in Figure 15.

There is a study on how users use the interaction styles performed by Martin Dostál [28]. In the study, menu is found as prevalent interaction style, toolbars and keystrokes were particularly used to frequently used user commands. Toolbars were found used in document-content context and keystrokes in application control context.

A user would probably appreciate a way to personalize Boulevard manually — for example, to fix positions for some commands, add or remove commands and choose the visual representation style. Unfortunately, such behavior has not been implemented yet, except one: changing number of items (commands) and a layout of Boulevard by direct manipulation with the Boulevard (adaptable



(a) Menu-like only representation (b) Toolbar-like only representation

Figure 13: Adaptive representation

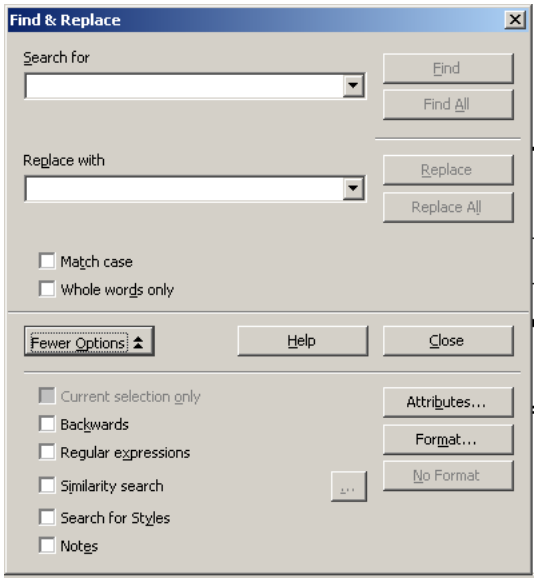
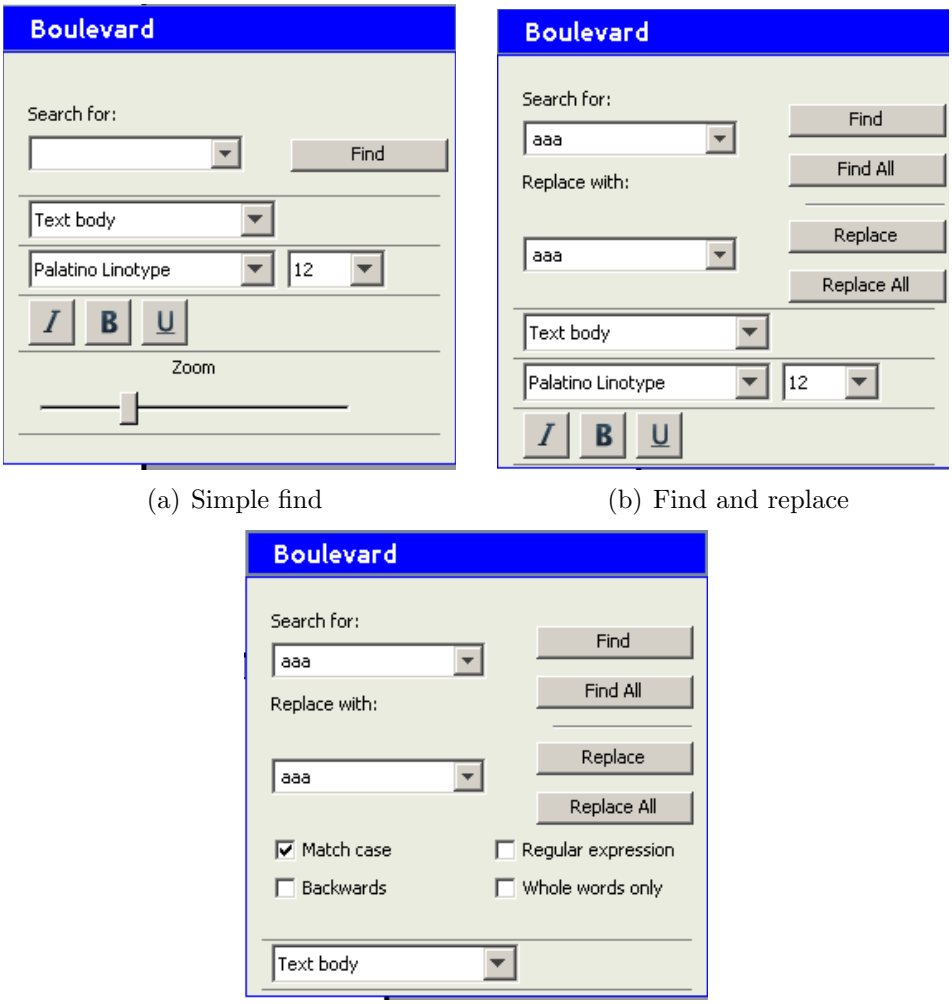


Figure 14: The original find dialog in OpenOffice.org



(a) Simple find

(b) Find and replace

(c) Advanced find

Figure 15: Various Boulevard representations of the find command

approach).

3.1 Implementation Issues

Implementing Boulevard in to the existing software is tediously difficult. The first issue is absence of a good suitable logger for building adaptive interfaces. Without a logger, there is not an adaptive interface. We have several requirements to such a logger: (1) we need a logger which logs user command, parameters and even the interaction style. (2) Such a logger must also have some kind of API, as we need to know about the performed action immediately, not after some time required to parse the logger data. Implementing such a logger is very difficult and we suppose this is the main reason, why there are so few adaptive interfaces prototype for the existing software.

Of course, more than a logger is necessary. There are also many prerequisites, e.g., context synchronization of items (commands) in Boulevard (e.g., which font is currently selected, more detail in Chapter 9.3.1), access to the application icons and also the access to the windows of an application, in which we can draw Boulevard.

4 Expert Systems

Boulevard is driven by an expert system, which provides the above-described functionality. There were several reasons why we decided to use an expert system as a core of Boulevard:

- A relatively easy transfer of ideas to the expert system rules.
- Non-sequential declarative programing is easy to extend and modify, which is important for prototyping and testing.

The history of expert systems begins in 1950's, when the first languages for symbolic manipulation were introduced. Symbolic manipulation appeared as a promising way for Artificial Intelligence (AI) since a higher abstraction level of programming. The most prominent and well-know a language of this kind is LISP (LISt Processing). A lot of AI programs were written in LISP, for example SHRUDLU [52] (program for understanding a natural language), because of its higher abstraction level, which leads to a more elegant and "intelligent" solution. Expert systems are mostly very dynamic, for example rules can be added or removed at the run-time — like in LISP, so the implementation of an expert system in LISP is easier than in static language (e.g., C).

In 1984 at NASA's Johnson Space Center, the AI department invented many prototypes of expert systems based on the most modern hardware and software. These expert systems appeared as promising, but there was some trouble to integrate them to other NASA projects. Usage of LISP as a core language turned out as ambivalent. The benefits of LISP were important, but at that time there were some complications with LISP, such as low availability of LISP computers, high cost of LISP software tools and hardware and poor interaction of LISP with other languages, which is important for complex systems. Just reminding the FLEXCEL, which was reportedly slow because of poor interaction between LISP and Microsoft Excel.

In 1985, CLIPS (C Language Integrated Production System) was invented as a response to eliminate the disadvantages of LISP, but to kept the advantages of expert systems. CLIPS was released as public domain software and later became the most widely used expert system, a de-facto standard for expert systems. CLIPS is more than a language for writing expert systems, today it is also a complete environment for developing expert systems (IDE).

An expert system is a computer program which is intended to help or simulate behavior of a human expert. There is a funny, but partly accurate definition of a human expert in [3]: *"One definition of an expert is someone more than 50 miles from home and carrying a briefcase."* We consider an expert to be someone with long experience, skills and advanced knowledge in some branch. Someone whose opinion is widely respected in some area of expertise. To simulate an expert, an expert system must have the same or similar knowledge like one. There are three basic types of knowledge in expert systems:

Fact — basic high-level information holder, a basic unit of data used by rules. They can be simple — non structured, like "a rabbit" or "a turtle", or more complex — structured, like "A rabbit, which likes carrot". The complex facts are structured into slots. Facts represent objects and their structure.

Rule — represent experience knowledge. Rules use IF-THEN form. For example: "If rabbit has eaten all food, then give him another". Every rule has a Left Hand Side (LHS) and a Right Hand Side (RHS). The LHS acts as "IF" predicate and the RHS acts as "THEN" part. By other words the LHS represents a list of premises and the RHS conclusions. The LHS contains fact "templates" which must be satisfied — associated (unified) with real facts.

Procedural knowledge — holds knowledge how to do something, in a sequential way. Such knowledge is represented by functions. For example, a procedure how to feed the rabbit: "open the cage, insert food, close the cage".

The core of an expert system is the inference engine, which is responsible for which and when rules will be executed. Inference is based on the Modus Ponens rule, which stands for $A \rightarrow B$, $A \vdash B$. This means, from fact A and rule $A \rightarrow B$ infer B (add B to the fact base). Strictly in expert systems, B does not have to be only a fact, B can be a sequence of commands, which can affect facts, rules or even the inference engine. The inference engine creates an inference chain, which is a sequence of executed (fired) rules with their corresponding unified facts. There are two basic types of chaining:

Backward chaining – has defined goals and tries to resolve goals by determining whether they are provable and at which circumstances. The result (if provable) is goals with variables unified by facts. This approach is used by PROLOG [22].

Forward Chaining –which does not have any defined goals, simply starts inferring (deducing) from the starting facts using defined rules, until there is nothing to deduce. This approach is used in most expert systems, even in CLIPS.

Although these appear as very different approaches, they are not so different. Backward chaining is more a deduction calculus, forward chaining is more suitable for dynamic systems, where the change of situation's conditions is frequent. Forward chaining is particularly more useful for adaptive user interfaces, since the user's action triggers a new inference.

The forward chaining inference engine starts by creating a list of all possible rules that satisfy LHS. Such a list we call an agenda. This means trying to unify

the fact templates in LHS with particular facts from the fact list. A rule with associated facts which satisfies LHS is one item in the agenda. There can be one rule contained more times, but with different associated facts. An agenda is a set of rules with unified variables which satisfy LHS and so can be executed (fired).

It is common that there are more than one item at one time in agenda, so that is why there are strategies. Strategies act as a conflict-resolution mechanism for such cases. It is possible to write expert systems without the need of strategies, such a program always has only one possible rule for execution. But then such a program becomes completely sequential, thus writing such a program as an expert system is meaning less and more complicated than under imperative paradigm. So, in true expert systems conflict-resolution strategies are needed. Of course, there are many ways how to resolve a conflict, for example a random choice or other more sophisticated approaches. The default CLIPS's conflict-resolution strategy is the depth-first strategy: newer activated rules (recently added to the agenda) are placed before the older activated rules. Every rule can have a defined salience, which acts as a priority of a rule. Salience is respected by the resolution strategy as more important than other aspects.

Expert systems have a important mechanism to avoid cycling. Activation is a pair of a rule and corresponding unified facts. If a rule is activated, this activation is remembered since the same activation is denied to avoid cycling. We should remember that an activation pair contains an internal representation of a fact, so if a fact is modified to the same value, its internal representation ID changes so that such a pair can be activated again.

A naive implementation of an expert system can be relatively easy, but very slow in terms of performance. Checking and unifying LHS of every rule on every fact is a very computationally exhausting task. There is a much more clever algorithm which reduces the complexity of task. It is called the RETE algorithm [36].

5 Boulevard Expert System

There are several reasons to use an expert system in Boulevard. The main reason is a relatively easy transfer of an idea of Boulevard behavior to corresponding rules. In most cases, one Boulevard feature is ensured by one corresponding rule. Such rules are easy to read and write, and therefore easy to modify. It is also unimportant where such a rule should be placed in the program, in contrast to procedural programs. It is easy to enable or disable some Boulevard features by simply enabling or disabling the corresponding rules. This is very useful for Boulevard testing on users.

Use of an expert system has also some disadvantages, like higher requirements to the quality of code since a poorly implemented expert system often loses its advantages. Such an expert system is then hard to understand and modify. Another disadvantage of expert systems is related to difficult debugging, the reason is obvious: we do not know in which sequence rules are fired. A good expert system is non-sequential at all, do not use any additive synchronization facts and is independent of the used strategy (in the point of view of correctness, not performance).

5.1 Facts

The most important fact type in the expert system is `action`, which is a fundamental further inseparable item, mostly one user command, for example "Bold" or "AddTable". The name `action` originated before before we changed the used terminology — from user action to user command. This fact type is a fundamental part of the expert system, so it is described in detail with all slots:

- name — an internal name of an command, mostly same as UNO command in OpenOffice.org
- label — a user-friendly name of a command, used for item visualization. There is problem with localization of Boulevard in current CLIPS version since a label cannot contain for example Czech characters.
- in-boulevard — this Boolean indicates if the item should be in Boulevard or not.
- rank — this number indicates the rank of an item, used for determine prominence of the item. For how rank is computed see formula 1.
- total — this number indicates the total activation of an item
- representation — contains determined representation style of an item
- boulevard — the count of activations of an item from Boulevard

- toolbar — the count of activations of an item from a toolbar
- menu — the count of activations of an item from a menu
- context-menu — the count of activations of an item from a context-menu
- keystroke — the count of activations of by a keystroke
- toolbar-dialog – the count of activations using a toolbar-dialog interaction style, which means a toolbar for initiating a user command and a dialog for executing the user command
- menu-dialog — the count of activations using a menu-dialog interaction style
- context-menu-dialog — the count of activation using a context-menu-dialog interaction style
- keystroke-dialog — the count of activation using a keystroke-dialog interaction style
- boulevard-primary-position — an arrangement of items in Boulevard is ordered by prominence of positions. Every item in Boulevard has its own unique position, determined by the primary and secondary position. The primary position represents primary prominence of an item. More items can have the same primary position, which indicates that these items are in the same sweeping back group.
- boulevard-secondary-position — indicates the secondary position of an item.
- parameters — this slot holds a list of several recently used parameters to an command, like the particular size of the used font-height.
- ui-element — an exact user interface element used to visualize an item in Boulevard, like `toolbar-button`, `menu-button`, `find-simple`. This is one of the outputs from an expert system, so the visualization layer must understand these values.

The fact called `boulevard-parameters` holds the parameters of Boulevard, such as frequency-recency weight, the count of total used commands, the queue of recent commands, the number of commands in recency queue, the minimal rank, the rank difference-tolerance and the count of recent parameters.

The fact `sweeping-back-group` just defines which commands are attended to sweping-back.

5.1.1 Item Uniqueness

A very important fact is that mostly every user command is considered unique by its name. But there are commands which are unique as a pair of a name and a parameter. These commands are currently only color-related commands: "FontColor", "FontBackground" and "FontHighlight". This is because we wanted such command to be presented as a separate item with its own parameters, rank and so on. That also explains why these commands are in strange one-element-only sweeping-back groups. For good understanding of the color handling see Figure 13.

Another important fact type is `new-action`, which is used for non-processed commands by the expert system, this fact holds information about which user command was used (with corresponding information like parameters and interaction style). An expert system processes this fact by updating an existing `action` fact or creating a new `action` fact and then retracts this `new-action` fact (in dependence of an item uniqueness).

5.2 Rules

There are twenty-six rules in our expert system and each of them has its own responsibility and ensures a different feature. Some rules seem to be very similar and contain a few of the same assumptions. It is a compromise between a non-sequential expert system and code reuse. Expert system rules can be divided into several categories:

- process new action
- sort actions
- rank computation
- sweeping back
- visual style - representation
- minimal rank
- find parameters

5.2.1 New Action

The Interceptor gives information about performed user command (with all important details) on the input of the expert system. The Boulevard manager ensures the assertion of this information as a `new-action` fact to the expert system. There are three possible cases: the very first command, the first command and an update command. Every case is ensured by a different rule.

The `first-action` rule ensures a case, when there is no item (command) in Boulevard at all, so we assert the `action` fact at primary-position 1 and secondary-position 1. This rule seems redundant, but it simplifies other rules and there is no need to hold another extra fact. The `total-number-of-activations` is updated every time (holds the total number activations used for computing ranks). This is not necessary and can be computed as the sum of all items, so this is only optimization.

```
(defrule first-action
?f1 <- (new-action (name ?name) (initiated ?initiated)
             (executed ?executed) (parameter ?parameter))
(not (action))
?boulevard-parameters <- (boulevard-parameters)
=>
(process-action (assert (action (name ?name) (label ?name)
        (boulevard-primary-position 1))) ?initiated ?executed ?f1
        ?boulevard-parameters ?parameter))
```

The `create-action rule` ensures a case, when there are items in Boulevard, but not exactly an item with the same name as the `new-action` — item in Boulevard is unique (see section 5.1.1 about item uniqueness). So a new `action` fact with the corresponding attributes is asserted..

```
(defrule create-action
?f1 <- (new-action (name ?name) (initiated ?initiated)
             (executed? executed) (parameter ?parameter))
(not (action (name ?name)))
(action (boulevard-primary-position ?pp1))
(not (action (boulevard-primary-position ?pp2 &:(> ?pp2 ?pp1))))
             ?boulevard-parameters <- (boulevard-parameters)
(Unique-actions (actions $?actions))
(test (not (member$ ?name ?actions)))
=>
(process-action (assert (action (name ?name) (label ?name)
       (boulevard-primary-position (+ ?pp1 1)))) ?initiated ?executed ?f1
        ?boulevard-parameters ?parameter))
```

And an `update-action` rule ensures case a when in Boulevard is an item with the same name as the `new-action` — item is not unique in Boulevard, so it is important to update an `action` fact with new information, such as new usage of action by the user and parameters.

```
(defrule update-action
?f1 <- (new-action (name ?name) (initiated ?initiated)
```

```
         (executed ?executed) (parameter ?parameter))
?action <-  (action (name ?name))
?boulevard-parameters <- (boulevard-parameters)
        (Unique-actions (actions $?actions))
(test (not (member$ ?name ?actions)))
=>
(process-action ?action ?initiated ?executed ?f1 ?boulevard-parameters
        ?parameter))
```

5.2.2 Compute rank

The computation of rank is ensured by only one rule . The rule for each fact `action` calculates the rank and if the newly calculated rank is different from the original one, the `action` fact is modified with a new calculated rank. For rank calculation there is a function which represents procedural knowledge about how to calculate rank from the given values. The function is called `compute-rank` and takes the following parameters: the item name, the items's total activations, total activations of all functions, the recency actions queue and the frequency-recency ratio. For more information about rank, see Formula 1.

5.2.3 Sorting Actions

These rules serve to reorganize Boulevard, especially by rank. For this purpose, there are two rules: `swap-primary-positions` and `swap-secondary-positions`. The first is used to sort by primary positions and the second one for sorting by secondary positions. Sorting by primary position is quite simple. The rule "search" (in our context means it that there are facts which satisfy condition) for two items, where the first one has a higher rank than the second one and the first one has a higher primary position than the second one, which means that the first one is on a less prominent position than the second. If this is satisfied, the primary positions of both items are swapped. To be more exact, swapped are not only these items, but all items with the same primary positions. Swapping entire rows on a specific primary position is handled by function `swap-primary-positions-function`. Only items with secondary position 1 are considered, since they are the most prominent items on primary positions — with the highest rank. To avoid frequent changes in Boulevard caused by meaningless rank changes, we use a `rank-difference` constant. Difference in ranks of items which we want to swap must be greater than the `rank-difference`.

```
(defrule swap-primary-positions
?action1 <- (action (rank ?rank1) (boulevard-primary-position ?pp1)
     (boulevard-secondary-position 1))
?action2 <- (action (rank ?rank2) (boulevard-primary-position ?pp2)
```

```
     (boulevard-secondary-position 1))
(test (and (> ?rank1 ?rank2) (> ?pp1 ?pp2)))
(boulevard-parameters   (rank-difference ?rank-difference)
                                        (minimal-rank ?minimal-rank))
(test (or   (> (abs(- ?rank1 ?rank2)) ?rank-difference)
                          (< ?rank2 ?minimal-rank)))
=>
(swap-primary-possitions-function ?pp1 ?pp2))
```

Sorting by the secondary position rule is very similar to sorting by the primary position rule. This rule is only for actions that are "swept-back", so there are more than one item on one primary position. Consider two items, the first one and the second one, both on the same primary position. If the first one has a higher rank than the second one and the first one is on a higher secondary position than the second one, then the secondary positions of both items are swapped. Unlike sorting by primary position, only considered items are swapped, not the entire row or column. And also in this case the `rank-difference` is used for avoiding frequent insignificant changes in Boulevard.

```
(defrule swap-secondary-positions
?action1 <- (action (rank ?rank1) (boulevard-primary-position ?pp1)
          (boulevard-secondary-position ?sp1))
?action2 <- (action (rank ?rank2) (boulevard-primary-position ?pp1)
         (boulevard-secondary-position ?sp2))
(test (and (> ?rank1 ?rank2) (> ?sp1 ?sp2)))
(boulevard-parameters   (rank-difference ?rank-difference)
                                         (minimal-rank ?minimal-rank))
(test (or   (> (abs(- ?rank1 ?rank2)) ?rank-difference)
                    (> ?rank1 ?minimal-rank)))
=>
(modify ?action1 (boulevard-secondary-position ?sp2))
(modify ?action2 (boulevard-secondary-position ?sp1)))
```

Positions of items are changed also because we do not want to have gaps (empty positions) in Boulevard. A gap on the primary position appears after sweeping-back since two or more actions are given on the same primary position, so empty positions may appear. In the previous versions of the expert system also avoiding gaps on secondary positions were ensured. Gaps on secondary positions appeared when there were several items on the same primary position and rank of one of them fell below minimal rank. Such an item is not included in the final Boulevard, but may occur that another item with a higher secondary position has a higher rank and thus is included in the final Boulevard, but these items are not swapped since the difference of ranks is below the `rank-difference`. Now it is not necessary, since we have modified the sorting rules so that they prevent the

ussie of minimal rank. The rule is simple: If there is an item that has no other item above, except the item with primary-position 1, such an item is moved "up" by one position.

```
(defrule avoid-gap-primary
?action <- (action (boulevard-primary-position ?pp1)
(boulevard-secondary-position 1))
(not (action (boulevard-primary-position ?pp2 &:(= (- ?pp1 1) ?pp2))
(boulevard-secondary-position 1)))
(test (not (= ?pp1 1)))
=>
(modify ?action (boulevard-primary-position (- ?pp1 1))))
```

5.2.4 Sweeping back

The `sweeping-back` rule keeps Boulevard organized. This rule uses predefined groups of items (commands) which we want to have together, no matter their ranks (except, of course, cases below minimal rank). The groups contain similar user commands, like Bold, Italic and Underline. The currently defined groups are:

- FontColor
- BackgroundColor
- BackColor
- CharFontName,FontHeight
- Bold, Italic, Underline
- Undo, Redo, Repeat
- LeftPara, RightPara, CenterPara, JustifyPara
- Cut, Copy, Paste, SelectAll
- Open, Save, SaveAs, SaveAll, Reload, NewDoc
- ExportTo, ExportToPDF
- DefaultNumbering, DefaultBullet
- ShowTrackedChanges, TrackChanges, ProtectTraceChangeMode, CommentChangeTracking, MergeDocuments
- DecrementIndent, IncrementIndent

- PrintPreview, Print Printer
- BrowseView, PrintLayout ,Zoom, FullScreen
- DeleteColumns, DeleteRows
- InsertColumns, InsertRows
- SplitCell, MergeCells
- InserPageHeader, InsertPageFooter, InsertFootNote, InsertBookmark, InsertAnnotation,
- InsertTable, InsertFrame, InsertGraphic
- InsertDateField, InsertTimeField, InsertPageNumberField, InsertPageCountField, InsertTopicField, InsertTitleField, InsertAuthorField
- InsertSection, SetHyperlink
- SpellingAndGrammarDialog, Therauzus, SpellOnline
- SetAnchorToPage, SetAnchorToPara, SetAnchorAtChar, SetAnchorToChar
- BringToFront, SendToBack, ObjectForwardOne, ObjectBackOne
- FlipVertical, FlipHorizontal
- LineSpacing, LeftRightMargin, TopBottomMargin, TabStops

Please note the special groups FontColor, BackgroundColor and BackColor, which consist of only one action. This certainly looks strange, but it is all right and it is because these actions are special. Their uniqueness is given not only by a name, but also a parameter. See Chapter 5.1.1. This feature is provided by a single rule. Let us consider two items. The first one keeps its position and the second one is moved. The first one and the second one are together in one sweeping-back group. The first one has a lower primary-position than the second one. If this is satisfied, then the second one is moved to the same primary-position as the first one and to the secondary-position of one greater than the current greatest secondary position.

```
(defrule sweeping-back
(sweeping-enable)
(action (name ?name1) (boulevard-primary-position ?pp1)
          (boulevard-secondary-position ?sp1)) ;ke ktere
?action2 <- (action (name ?name2) (boulevard-primary-position ?pp2))
(not (action  (boulevard-primary-position ?pp1) (
```

```
        boulevard-secondary-position ?sp4 &:(> ?sp4 ?sp1))))
(sweeping-back-group (group $?group))
(test (and (member$ ?name1 ?group) (member$ ?name2 ?group)))
      (test (< ?pp1 ?pp2)) ; 6.4.201
=>
(modify ?action2 (boulevard-primary-position ?pp1)
         (boulevard-secondary-position (+ 1 ?sp1))))
```

There is also a rule for disabling the sweeping-back feature. This is simply since we wanted to be able to enable or disable this feature at run time. This is useful for testing, better understanding and presenting the sweeping-back feature. The rule is not important, so it is not presented in detail. The rule simply takes items on secondary position greater than 1 and puts them to the bottom of Boulevard. Such items are later sorted to the right position.

5.2.5 Visual Representation

Boulevard has some kind of simple meta-GUI built-in, because of an adaptive representation. We must describe the item's interaction model which is used for generation of proper user interface visualization and behavior. This ensures what we call adaptive representation, which means adapting visual representation of items (commands) contained in Boulevard to the users preferred interaction style.

This task consists of two parts. In the first part, the most frequently used interaction style is selected and in the second one a specific user interface element is assigned. The first part consists of the calculation of the most frequently used interaction style based on values stored in `action` slots, such as: toolbar, menu, context-menu, keystroke and Boulevard. For this purpose there is the `select-representation` rule. The main computation is performed by function which returns the most used interaction style. The computed value is stored in the `representation` slot.

The second part uses the above computed value stored in the item's `representation` slot, special facts `actions-interaction-group` and `interaction-representation-ui-element`. For this purpose, there are these rules: `select-ui-element`, `select-default-ui-element` and `select-ui-element-exception`.

```
(defrule select-default-ui-element
?action <- (action (representation ?representation) (name ?name)
(ui-element ?ui-element))
(interaction-representation-ui-element (representation ?representation)
(interaction no-selection) (ui-element ?element))
(not (actions-interaction-group  (actions $?actions &:(member$ ?name
?actions))))
     (test (not (eq ?ui-element ?element)))
```

```
(not (ui-element-exception (action ?name)))
         (test (not (eq ?name ExecuteSearch)))
=>
(modify ?action (ui-element ?element)))
```

The rule `select-default-ui-element` considers item and its representation and using defined `interaction-representation-ui-element` fact assignes a default user interface element used for non-selection interaction. The Rule uses `interaction-representation-ui-element` groups, which defines the user interface element for representation and interaction style. The groups are in Table 1. We consider three possible interactions for group of items (exactly, there

Preferred interaction style	Command interaction	UI element
toolbar	single-selection	ToolbarToggleButton
toolbar	no-selection	ToolbarButton
toolbar	multiple-selection	ToolbarDropDownList
menu	single-selection	MenuToggleButton
menu	no-selection	MenuButton
menu	multiple-selection	ToolbarDropDownList
context-menu	single-selection	MenuToggleButton
context-menu	no-selection	MenuButton
context-menu	multiple-selection	ToolbarDropDownList

Table 1: Adaptive Representation Groups

can be only single item in a group). The possible interactions are: no-selection, single-selection and multiple-selection. No-selection is the default interaction, used for cases when an item cannot be active or inactive, like undo or redo. Please be careful in distinguishing between active and enabled, inactive and disabled. "Command is active" means that the user command is currently used, like boldface or bullets, but "command is enabled" means that the command is possible to use. Single-selection is interaction when item can be active or inactive. For simplicity, such behavior you can imagine like check box. And multiple-selection interaction is for actions, which can have more than two values (on/off), for example font size or font name.

You can see that `ToolbarDropDownList` is used for all possible representations since this kind of interaction is used by a small group of commands and for all of them and for all possible representations the `ToolbarDropDownList` is intended. Also for the context-menu and the menu representation there is always the same ui-element assigned. The reason is we consider context-menu and menu visual style as very similar for the user. But the system is ready for a simple change.

Single selection items are: Bold, Italic, Underline, Strikethrough, LeftPara, RightPara, CenterPara, JustifyPara, ControlCodes, SpellOnline, TableBound-

aries. Possibly, there are some commands missing here, but some missing commands are intended, like bullets since in current OpenOffice.org version 3.2 bullets in toolbar use surprisingly no-selection interaction style, so Boulevard uses the same style as toolbar.

Multiple-selection items are: FontHeight, CharFontName and StyleApply. No-selection items are not defined since it is the most frequently used interaction style, so it is the default style. Only for the special items above listed we use different style. This approach has also the advantage that new items that are not defined in Boulevard expert system are displayed correctly with big probability and the definition of all items is not necessary. This is ensured by `select-default-ui-element` rule. The rule `select-ui-element` assigns ui-element for single-selection and multiple-selection interaction.

```
(defrule select-ui-element
?action <- (action (representation ?representation)
(name ?name) (ui-element ?ui-element))
(interaction-representation-ui-element (representation ?representation)
(interaction ?interaction)
(ui-element ?element))
(actions-interaction-group  (interaction ?interaction)
(actions $?actions))
(test (member$ ?name ?actions))
     (test (not (eq ?ui-element ?element)))
(not (ui-element-exception (action ?name)))
=>
(modify ?action (ui-element ?element)))
```

There is also a rule used for exceptions, which is currently only one: Zoom command. In this case the `ToolbarSlider` ui-element is assigned to the Zoom-Slider action, no matter on the representation. This rule is intended as a relatively easy way to adjust a small set of items.

```
(defrule select-ui-element-exception
?action <- (action (name ?name) (ui-element ?element-orig))
(ui-element-exception (action ?name) (ui-element ?element))
         (test (not (eq ?element ?element-orig)))
=>
(modify ?action (ui-element ?element)))
```

5.2.6 Minimal Rank

We have these simple rules mainly due to sweeping-back, because this feature places items on the higher primary position than they should be strictly

by rank. So without this rule only a once used item (command) with the rank close to zero never disappears from Boulevard since this item is in the same sweeping-back group with a frequently used item. This is an unwanted behavior. So this rule uses a `minimal-rank` constant, which is minimal rank of item to be contained in Boulevard. The rule `add-action-to-boulevard` is attended for adding items to Boulevard by modifying the slot value `in-boulevard` to `true`, if the item has a higher or equal rank as defined `minimal-rank`. The rule `remove-action-from-boulevard` removes an item from Boulevard if the item's rank is lower than the `minimal-rank`.

```
(defrule add-action-to-boulevard
?action <- (action (rank ?rank) (in-boulevard false)
(representation ?repre))
(boulevard-parameters (minimal-rank ?minimal-rank))
(test (>= ?rank ?minimal-rank))
=>
(modify ?action (in-boulevard true)))

(defrule remove-action-from-boulevard
?action <- (action (rank ?rank) (in-boulevard true))
(boulevard-parameters (minimal-rank ?minimal-rank))
(test (< ?rank ?minimal-rank))
=>
(modify ?action (in-boulevard false)))
```

5.2.7 "Find" Parameters

These special rules are used only for the find item (command). This command is special since it has complex parameters which are considered for assigning ui-element. The first rule is for processing a newly-asserted `find-parameters` fact, similar to `new-action`. We want to store only a limited number of last-used find parameters, for this purpose an index is assigned to every `find-parameters`. This value is stored in the slot `index`. The first rule `process-new-find-parameters` finds the currently highest value of index, and stores this value incremented by one to the new one. The second rule is attended for removing old `find-parameters`. The required quantity of stored `find-parameters` is stored in the `parameters-quantity` slot in the `boulevard-parameters` fact. So this rule retracts `find-parameters` facts with index lower than the currently maximum index minus `parameters-quantity`. The rest rules are for assigning the proper ui-element. The rules are: `select-find-simple`, `select-find-replace`, `select-find-simple-parameters` and `select-find-replace-parameters`. This rules choose that ui-element, which contains all interaction and possible parameters used by the user. The corresponding representations of the "Find" user command are in Figure 15.

```
(defrule select-find-simple
        ?find <-(action (name ExecuteSearch) (ui-element ?ui-element))
        (not (find-parameters (find-type findAll)))
        (not (find-parameters (find-type replace)))
        (not (find-parameters (find-type replaceAll)))
        (not (or    (find-parameters (match-case true))
                    (find-parameters (backwards true))
                    (find-parameters (reg-exp true))
                    (find-parameters (whole-words true))))
        (test (not (eq ?ui-element FindSimple)))
        =>
        (modify ?find (ui-element FindSimple)))

(defrule select-find-replace
        ?find <-(action (name ExecuteSearch)(ui-element ?ui-element))
        (or (find-parameters (find-type findAll))
            (find-parameters (find-type replace))
            (find-parameters (find-type replaceAll)))
        (not (or    (find-parameters (match-case true))
                    (find-parameters (backwards true))
                    (find-parameters (reg-exp true))
                    (find-parameters (whole-words true))))
        (test (not (eq ?ui-element FindReplace)))
        =>
        (modify ?find (ui-element FindReplace)))

(defrule select-find-simple-parameters
        ?find <-(action (name ExecuteSearch) (ui-element ?ui-element))
        (not (find-parameters (find-type findAll)))
        (not (find-parameters (find-type replace)))
        (not (find-parameters (find-type replaceAll)))
        (or (find-parameters (match-case true))
            (find-parameters (backwards true))
            (find-parameters (reg-exp true))
            (find-parameters (whole-words true)))
        (test (not (eq ?ui-element FindSimpleParameters)))
        =>
        (modify ?find (ui-element FindSimpleParameters)))

(defrule select-find-replace-parameters
        ?find <-(action (name ExecuteSearch)(ui-element ?ui-element))
         (or (find-parameters (find-type findAll))
            (find-parameters (find-type replace))
```

```
    (find-parameters (find-type replaceAll)))
(or (find-parameters (match-case true))
    (find-parameters (backwards true))
    (find-parameters (reg-exp true))
    (find-parameters (whole-words true)))
(test (not (eq ?ui-element FindReplaceParameters)))
=>
(modify ?find (ui-element FindReplaceParameters)))
```

6 Why OpenOffice.org

At the beginning we performed a feasibility study and we have chosen OpenOffice.org for several reasons. OpenOffice.org is very similar to Microsoft Office 2003, which is probably the most widely used office suite. But in contrast to Microsoft Office, OpenOffice.org is an open source, which brings better ways for extending and customizing. OpenOffice.org itself is very popular office suite, version 3 has been downloaded more than 100 million times and has been translated to more than 100 languages [12]. We consider OpenOffice.org as a very good platform for HCI research. For such purpose our logger OpenOffice.org Interceptor can be very useful.

6.1 Programming under OpenOffice.org

OpenOffice.org extensions have been introduced in version 2.0.4. Extensions use an approach similar to the one used in Mozilla Firefox. Is is intended as an easy way to add new functionality and behavior to the existing OpenOffice.org instance without modifying the source and reinstalling the OpenOffice.org. The user just double-clicks on extension, Extension Manager opens and user confirms the plan to install extension. That is all. Extension is one file which is in fact a ZIP archive containing special directory and file structure. There are XML files describing details of extensions, such as used toolbars and their icons, menus, and so on. Creation of OpenOffice.org extension in our case ensures the Netbeans OpenOffice.org plug-in [9].

6.1.1 UNO

UNO (Universal Network Objects) is a concept used for interoperability between various programing languages, object models and even hardware architectures. For every UNO supported language must exist a corresponding so-called language binding. Currently supported languages are Python, C++ and JAVA for developing new UNO objects, and OpenOffice.org BASIC, OLE automation and .NET CLI for using UNO objects. UNO RunTime is a virtual environment, where UNO objects run and interact. There is a meta-language for describing UNO objects called UNOIDL (Universal Network Object Interface Definition Language), which is used for specifying UNO Components. It is similar to C header files. UNOIDL definition file contains information about object, especially about used interfaces, methods, structures, services or other entities with its corresponding parameters and parameters types.

7 OpenOffice.org Interceptor

The most difficult part of implementation of any adaptive user interface to existing software is a logger. For our purposes we didn't find any suitable logger that would satisfy all our needs, so we developed our own logger called OpenOffice.org Interceptor. It uses advanced techniques and it is a combination of two logging approaches: it utilizes a build-in macro recording facility (underlying function call) and also considers user interface events.

Macro recording approach is used for detecting user commands and corresponding parameters. Example of such logger is OWL [41], which, however, do not log the commands parameters. The advantage of this approach is the absence of need of interpretation of the user interface events as user commands, which is a difficult task. However, from its nature comes it's limitation: since it fundamentally apart form the user interface events, it is impossible to log user interface event-based information, e.g., the interaction style.

User interface events-based logger gathers information about low-level (e.g., key presses and mouse move) and high-level (e.g., window move) user interface events. Such information could be useful for certain types of studies, e.g., examine the user interface layout and user interface controls position. However, for studies on users activity from user commands point of view such loggers are unsuitable. Some user interface events loggers can produce more than 100 lines of log for one simple user command. Example of such loggers are RUI [40] or AppMonitor [17]. There is also a problem to detect only command really performed by the user. By really performed command we consider command which was really executed, not only initiated and then canceled. For example, if the user opens the font dialog, selects font and font size, and then clicks on the cancel button. This command is only initiated, but not performed, so then we do not care about this action. Such a behavior would be difficult to implement using only user interface events approach.

An important part of our logger is API. Since Interceptor is implemented as a standard UNO object, it is possible to use it from various programming languages, which are supported by UNO. Such API is important for adaptive user interfaces, because of accurate and immediate response from the logger. This feature is also useful (and actually used) for user testing, such as conditioned tests (an interactive test design, where future tests depend on the previous user's behavior) and test control. Using UNO, our logger can be used from OpenOffice.org BASIC, which is an easy-to-use programming language for programming OpenOffice.org. Using OpenOffice.org BASIC even a less experienced user can experiment with adaptive user interfaces. We have written a 78-lines-long program that utilizes OpenOffice.org Interceptor to create adaptive menu on OpenOffice.org Writer using BASIC (Appendix A). OpenOffice.org with Interceptor is a framework for easy user interface experiments, like prototiping adaptive interfaces and user testing.

OOI provides the functionality of common loggers, like exporting of a log to a file. The information provided by the Interceptor is indicated by the log structure, which follows:

- User command — a string identifying the name of a performed command, e.g. "Bold", "SaveAs" or "Paste".
- Interaction style used to initiate a command
- Interaction style used to execute a command
- Time and date — time and date of the command in `YYYY-MM-DD hh:mm:ss:mss` format. This information represents time of command accomplishment, except a few cases, where the time of command invoke start is used.
- User — a string identifying the user. This parameter can be set in the OOI settings.
- App — particular application of the OpenOffice.org Suite (currently Writer or Calc).
- Command parameters — parameters that have been applied to command.

The low-level user interface events are not logged, since such type of information is only required for a certain kind of user studies. However, Interceptor can be easily extended to provide such information, eventually. Key press events (except keystrokes) are not logged in order to preserve privacy.

We present a log sample in Table 2. The table is divided into two separate parts, the second part contains "Parameters" column with corresponding parameters and values.

Action	Initiated	Executed	Date&Time	User	App
Bold	toolbar	toolbar	2009-04-20 18:12:55	Dostal	Writer
FontColor	toolbar	toolbar	2009-04-20 18:13:02	Dostal	Writer
InsertTable	menu	dialog	2009-04-20 18:13:29	Dostal	Writer
Zoom	toolbar	dialog	2009-04-20 18:13:41	Dostal	Writer

Parameters
Bold: true;
FontColor: 65535;
TableName: Sallary;Columns: 2;Rows: 4;Flags: 9;
Zoom.Value: 110;Zoom.ValueSet: 28703;Zoom.Type: 0;

Table 2: Log Sample

Interceptor user interface is divided into three tabs — the first tab contains the above discussed log, the second tab contains a brief statistics (aggregated to particular user commands). The third tab is attended to configuring Interceptor's settings.

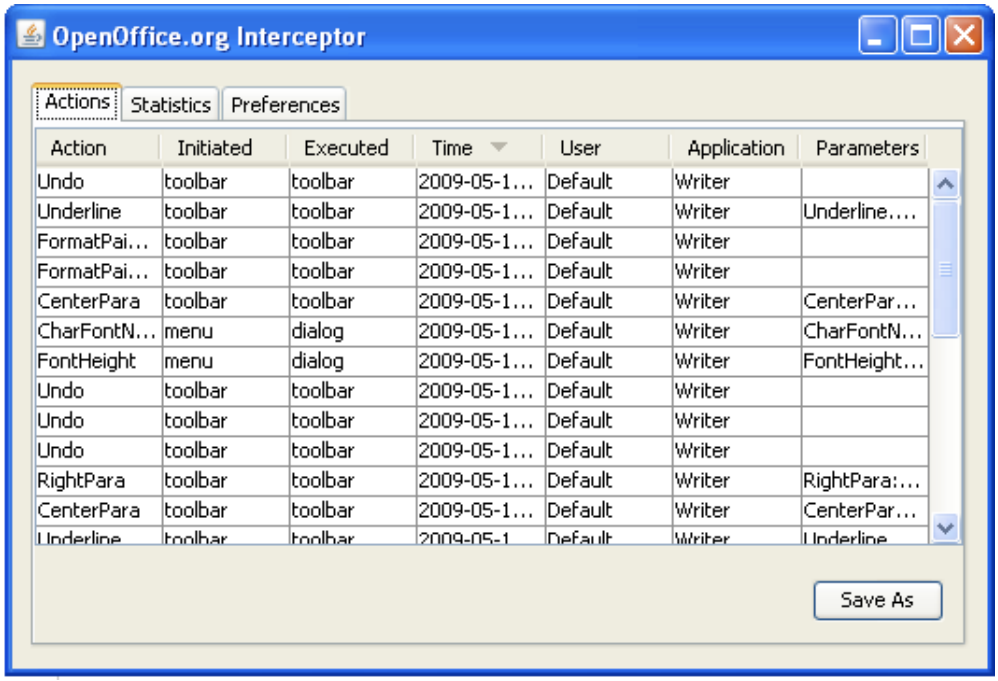


Figure 16: OpenOffice.org Interceptor window

8 Interceptor Architecture and Implementation

This section describes how user commands and user interface events are internally handled in OO (OpenOffice.org) in order to understand the logging algorithm described in Section 8.1. This chapter is adopted from an unused part of OOI (OpenOffice.org Interceptor) article, which we have written together with Martin Dostál. The term `action` originated before before we changed the used terminology — from user action to user command. Let us discuss user interface events first. Events (such as mouse clicks, key presses, window activations or document events) which happen during OO operation are handled by objects called Listeners. A component that handles an event must create and register a corresponding listener to the object that generates the handled event. A listener code is executed when such an event happens.

Another important part of OO internals is a Dispatch framework that is intended for executing commands provided by OO, or more precisely by a particular OO application (i.e. Writer naturally provides different functionality than Calc). Every user interface command has its own unique name, called URL that is handled by a corresponding dispatcher. Dispatchers are obtained from DispatchProvider objects. If an object requests a dispatcher, it should call DispatchProvider that for given URL provides a corresponding Dispatcher object. For instance, when the user clicks on the "boldface" button in toolbar, the button calls dispatcher (obtained from DispatchProvider) that will perform the corresponding action — setting a boldface. Another important fact is that it is possible to change the dispatcher provided by DispatchProvider at runtime using DispatchProviderInterceptor object that is responsible for managing dispatchers. In other words, Dispatch framework is useful for unified communication between user interface and OO. It completely separates commands from their user interface representation. This separation is clearly an advantage in the sense of program architecture but also a huge complication for implementation of logging. The consequence is that it is unable to determine interaction style used to perform user action straightforwardly. Now we will shortly describe the main components used to retrieve information about user activity:

- A Top Window listener — listens for topwindow events. A topwindow is considered as a rectangular container area in an OO window, represented by menus, popup menus, toolbars or dialogs, except the system dialogs that are not topwindows and thus must be handled separately. Topwindow events are used to inform about activation, deactivation or closing a topwindow. This information is required for determining the interaction style.

- A Key listener — listens for key press events. We are concerned only about keystrokes — key combinations that include a "Control" key modifier. This listener is required for determining the keystroke interaction style, such as pressing "CTRL + S" to perform the "Save" action.

- A Macrorecorder — is an object which records user activity and produces a macro source code. For instance, the default OO macrorecorder produces macros in OpenOffice.org Basic syntax. Macrorecorder is informed about most of the performed actions and their parameters. Such actions we call *macrorecordable*. However, there are some actions which do not inform macrorecorder about their activation and thus must be handled separately, namely "RecentFileList", "NewDoc" and "InsertSection".

- A Document event listener — listens for *document events*. A document event is a high-level action related to entire document, such as "Print", "Save", "Close", "NewDoc", etc. We handle only "NewDoc" document event, because this action is not macrorecordable thus can not be logged using the macrorecording facility.

- A DispatchProviderInterceptor — is intended for replacing a dispatcher by another one. We replace the original dispatchers for such actions that may take unpredictable time to perform or such actions that are not macrorecordable. In the case of this kind of action, we need to be informed about action before starts its execution, or respectively, before it ends. Please note that our dispatchers also calls the original (replaced) dispatchers to preserve original functionality of a dispatcher.

- Accessibility — we use UNO Accessibility API (UAA) [10] to determine a type of a topwindow. Furthermore, if the topwindow type is a dialog, then the name of the dialog provided by the Accessibility API is also considered. The topwindow type is used to determine a menu, a popup menu or a dialog as an interaction style.

8.1 Action Logging Implementation

This subsection introduces logging actions implementation. Generally, there are three ways we use to log an action:

- Using macrorecorder — as we stated above, most actions are logged using this approach. OOI implements own macrorecorder that is responsible for logging macrorecordable actions. Due to internal OO architecture, the default OO macrorecorder cannot be used with OOI simultaneously. The consequence is that OOI disables recording macros.

- Using DispatchProviderInterceptor — this approach is used to log actions that are not macrorecordable or may take unpredictable time to complete. Such actions are: "Paste", "Save", "Export", "Copy", "InsertFormula", "WebHtml", "SaveAsTemplate", "RecentFileList", "NewDoc", "InsertSection". Fortunately, the corresponding dispatcher is provided with action

parameters in the same way as macrorecorder is, so that actions are logged correctly including all parameters.

- Using document event listener — this approach is used to log only "NewDoc" action that is neither macrorecordable nor can be logged using DispatchProviderInterceptor approach so it must be handled using a Document event listener.

8.2 Determining Interaction Style of Performed Action

In this section we introduce how an interaction style used to initiate or execute an action is being determined. This part of logging is a much more complex task than action logging and uses similar techniques as event based loggers. We have mentioned the main four problems of determining interaction style in OO. The main pitfall lies in the fact that there is no straight way to determine initiation and execution style correctly due to separation of actions from user interface representation. Another complication arises from the fact that a dispatcher of a performed action is executed after closing of the topwindow (a dialog, a menu or a popup menu) where the action has originated. Nevertheless, the performed action must be associated with an originating topwindow that was used to initiate or execute the action in order to detect used interaction style. The third limitation is that toolbars can not be directly identified as an interaction style such as a menu, a dialog or a popup menu. The only possible solution is to consider a toolbar as used interaction style, whenever a keystroke, a menu, a dialog or a popup menu has not been determined. The fourth problem is that the dispatcher of a performed action informs the macrorecorder after such an action was completed. Such a solution may seem cumbersome, but it also yields an important advantage. OO toolbars contains many different and complex user interface elements with a a complex behavior (a formula editor, a media browser, drop-down listboxes, additional toolbars, etc.), whose activity can not be determined using Accessibility API nor any other source of user interface events available in the OO API. Now we will describe our logging algorithm in detail. The algorithm uses following variables to keep track on recent user interface events related to a recent action:

- `init` — contains the interaction style used to initiate an action. Possible values are a keystroke, a menu or a popup menu.
- `exec` — contains the interaction style used to execute an action. Possible values are a keystroke, a menu, a dialog or a popup menu.
- `timestamp` — contains time in milliseconds. This variable is used to validate values of the `init` and `exec` variables.
- `indialog` — a boolean variable which indicates that a dialog has focus.

Setting values of the above variables is not entirely straightforward and it is described at Figures 17, 18, 19 and 20. Now we continue the algorithm description with focus on topwindow events and dialogs. When a topwindow event is triggered (for instance, clicking "Paste" menu item in the "Edit" menu category), `init`, `exec` and `timestamp` variables are set. The `timestamp` variable is set to the current time. Therefore, when an action is performed, the `timestamp` and current time values are compared. If the difference between `timestamp` value and current time is less than 500 ms, then `init` and `exec` variables are proclaimed as *valid* and therefore, their values are used to identify the initiation and execution style of action. In other case, the variable values are considered as *invalid* and set to "toolbar", except a few special cases that require special handling as depicted on Figure 17.

For dialogs, `init` and `exec` variables must be handled more specifically. The `exec` variable is set to "dialog", when an action is executed from a dialog. The `init` variable value is observed when a dialog is opened and gets focus. It is also possible to have more than one dialog open when some of the opened dialogs are not modal. The `init` value is stored in the list of currently opened dialogs including the dialog name provided by the Accessibility API. This processing ensures correct behavior of the logging algorithm for non-modal dialogs. For instance, when the "Find and Replace" dialog is being opened, any possible action can be performed due to non-modality of the dialog.

Special handling is required for standard system dialogs that are provided by operating system, namely "Open" and "Save". The reason is that standard dialogs are not handled by a topwindow listener, and thus must be processed individually as seen on Figure 18. Figures 17, 18, 19 and 20 depict the logging process in appropriate detail. A few comments on Figure 17:

- The algorithm always starts in Figure 17; Figures 18, 19 and 20 describe corresponding blocks of the algorithm in more detail.
- Most actions imply that the logging algorithm must be performed more than once to log an action including all logged parameters. Therefore, the algorithm variables are not initialized at the start of the next iteration of the logging algorithm.
- Record action process: adds a new log entry according to the current values of `iniit`, `exec` variables of the recently performed action.
- Record Action Using Timestamp process: the process consists of two parts. The first part is described by "setting init and exec using timestamp" (see Figure 20) and the second part is described in a bullet above.
- Intercepted URL means an URL which is handled by DispatchProviderInterceptor in order to determine `init` and `exec` values correctly. See Section 8.1 for enumeration of intercepted URLs.

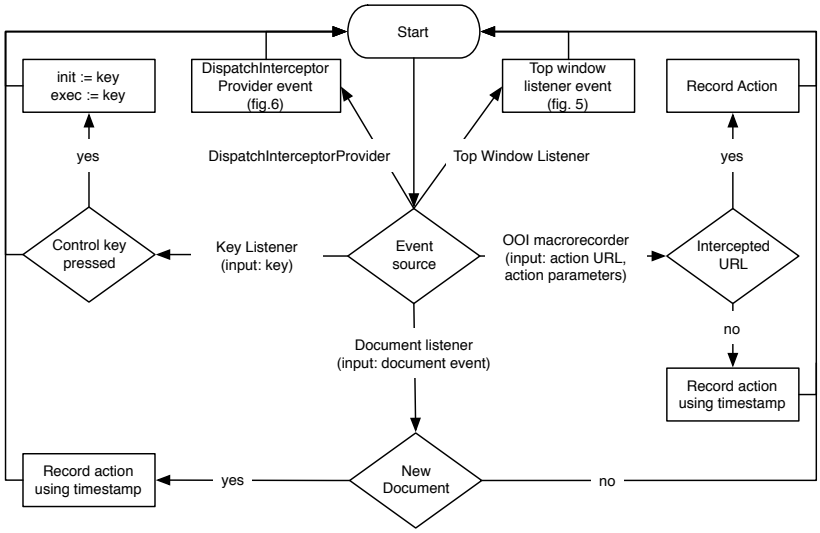


Figure 17: Logging Algorithm – The Main Part

8.3 Example of Performing Logging Algorithm

In order to clearly understand the logging algorithm we provide an example of action processing in logging algorithm. We will describe what happens when a user opens a font dialog using the menu. Then he selects "boldface" in the dialog and after that clicks the "OK" button. Then the user selects "boldface" again, but using a toolbar. In this case logging algorithm must be processed six times to process these two actions correctly:

1. The first incoming event is a topwindowlistener event "closed". The topwindow type is "menu", so that `init` and `exec` are set to "menu", `timestamp` is set to the current time (user selected "Character..." menu item in "Format" menu category).

2. The second incoming event is a topwindowlistener event "Activated": the topwindow type is "dialog". Since "Character" dialog is not opened yet, we continue with the "set init and exec" process: `timestamp` is valid, so that no changes to variables are required. Then the dialog is added to the list of currently opened dialogs including `init` actual value and dialog name provided by accessibility, so that the list contain the following values: `((''Character'' ''menu''))`. Variable `indialog` is set to true. (Character dialog has been displayed)

3. The third incoming event is a topwindowlistener event "Deactivated": topwindow type is "dialog" and `indialog` is set to false. (User selected boldface and clicked on OK)

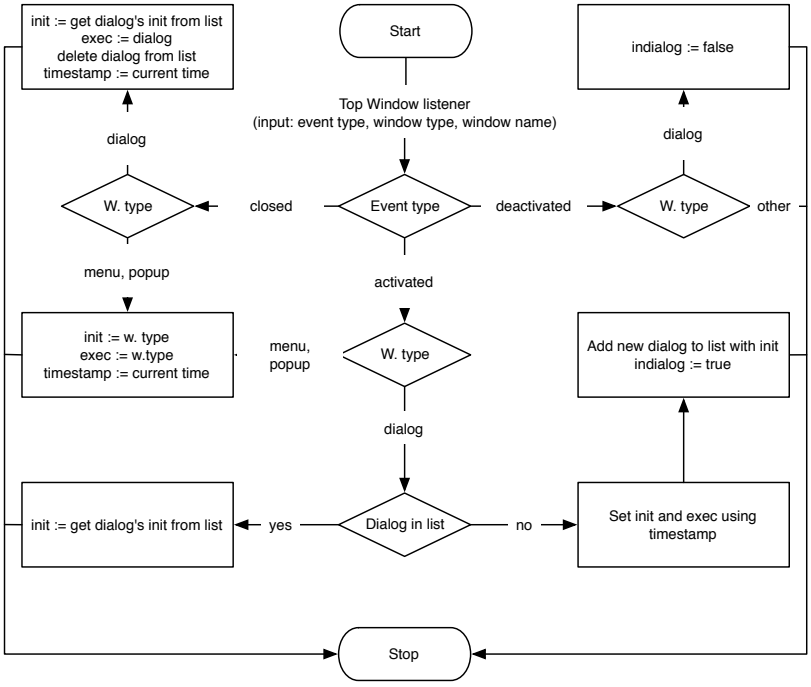


Figure 18: Logging Algorithm – The Top Window Listener Event

4. The fourth incoming event is topwindowlistener event "Closed": topwindow type is "dialog", then `init` is set to "menu" (obtained from list of currently opened dialogs), `exec` is set to "dialog" and `timestamp` is set to the current time. The dialog is deleted from the list. (Dialog has been closed)

5. Next event is processed by macrorecorder. Since "Bold" is not an intercepted URL, "Record Action Using Timestamp" is performed: `timestamp` value is valid, so that no variable value is changed. The resultant log entry:

   ```
   Bold, menu, dialog, 2009-05-08 16:14:01, Dostal, Writer,
   "Bold: true;"
   ```

6. Now we continue with the second action: the user clicks on "Bold" toolbar button. The incoming event is handled by the macrorecorder. URL is "Bold", which is not an intercepted URL, so "Record Action Using Timestamp" process is performed: `timestamp` value is proclaimed as invalid and `indialog` is false, thus `init` and `exec` variables are thought as invalid and then both rewritten to "toolbar". Afterwards "Record action" is performed and the following log entry is added:

   ```
   Bold, toolbar, toolbar, 2009-05-08 16:14:15, Dostal, Writer,
   "Bold: false;"
   ```

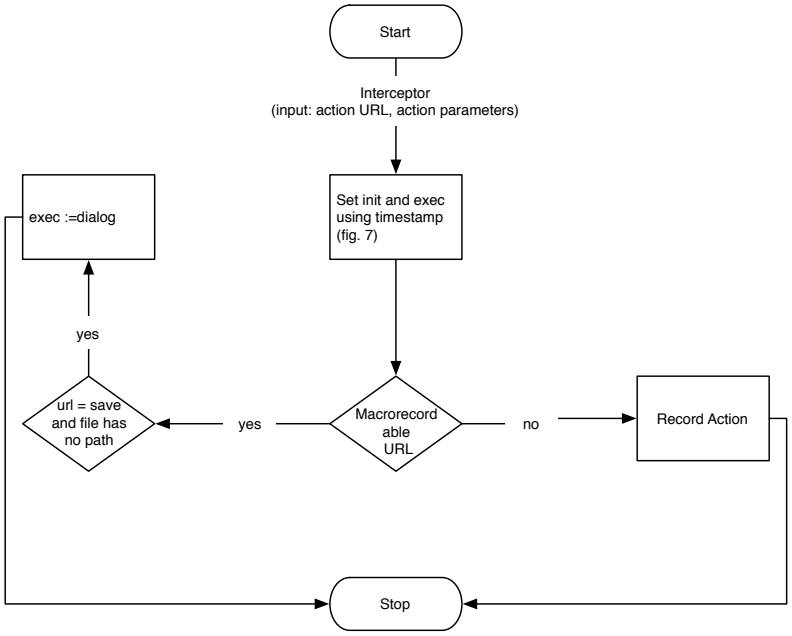


Figure 19: Logging Algorithm – The Dispatch Provider Interceptor Event

8.4 OOI Limitations

This section briefly discusses OOI limitations. Note that they are mainly caused by bugs in OO version and UNO limitations.

- Although OOI is written as platform independent, it works under Windows and Linux but not under Mac OS X. This limitation is caused by OO and Mac OS X compatibility problems. We believe that this problem will be solved in a future version of OO for Mac OS X.

- OOI is limited to work with Writer and Calc only. Another OO applications are disabled at the present due to their insufficient stability of the macrorecording interface. We assume that the reason is that macrorecording support under Impress, Draw, Base and Math applications has been added recently, and thus must be improved in order to provide enough compatibility and stability of OO API.

- It is not possible to record macros when OOI logging is enabled. This limitation is caused by the internal architecture of OO which does not enable using more than one macrorecorder at a time. In other words, there can be more than one macrorecorder created at a time but switching between macrorecorders — a "logging" macro recorder used by OOI and original OO macrorecorder is not possible. Nevertheless, this issue may be solved in future OO versions if developers make appropriate changes to OO API.

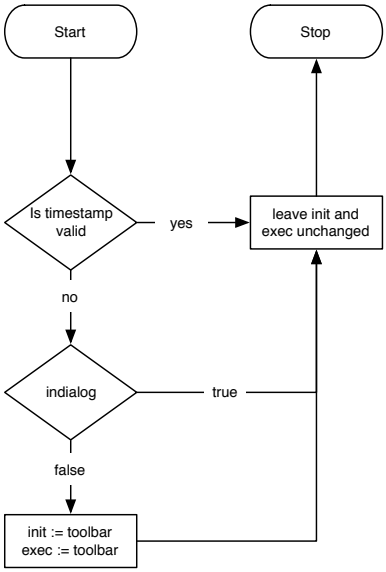


Figure 20: The Logging Algorithm – Setting `init` and `exec` values using `timestamp`

8.5 Rapid Prototyping of Intelligent Interfaces

OpenOffice.org Interceptor offers one more feature important for HCI researchers — a program interface. It can be used as a logging framework for developing intelligent user interfaces under OO. Most present loggers do not provide any logging API, do not provide information about parameters of performed actions. Also logged data must be processed further to understand user activity which limits their applicability as logging framework for developing intelligent user interfaces. OOI overcomes those limitations. In order to provide a standard program interface, OOI is implemented as an UNO component, so user actions may be logged in any programming language with UNO bindings. The OOI program interface is easy to use, see Figure 21 that presents simple OO Basic program which displays logged information about recently performed action.

How does the program depicted in Figure 21 work? Using `createUnoService` and `createUnoListener` an OOI instance is created. The `addListener` installs a listener object responsive for logging. When an action has been performed, the listener calls `ooi_listen` with following parameters: “action”, “initiated”, “executed”, “time”, “user”, “application” and “parameters”.

8.6 OpenOffice.org Interceptor Installation

Current version works with OpenOffice.org 3.2.1 (default installation) under Microsoft Windows XP Service Pack 3 and JAVA 1.6.20. Please notice the In-

```
Option Explicit
Global ooi As Object
Global ooi_listener As Object

Sub RegisterLogHandler
ooi = createUnoService("org.openoffice.oointerceptor.XOOInterceptor")
ooi_listener = createUnoListener("ooi_", _
   "org.openoffice.oointerceptor.XOOInterceptorListener")
ooi.addOOInterceptorListener(ooi_listener)
End Sub


Sub ooi_listen(action, initiated, executed, time, user, application, _
parameters)
MsgBox "action: " &  action & ", initiated: " & initiated & _
", executed: " & executed & ", time: " & time & ", user: " & _
user & ", application:" & application & ",  parameters: " & parameters
End Sub

Sub Main
RegisterLogHandler
End Sub
```

Figure 21: An Example of using OOI program interface

terceptor limitations described in Chapter 8.4.

1. Open the file `Interceptor.oxt` by double clicking or with OpenOffice.org Extension Manager.
2. Confirm installation.
3. After installation, relaunch OpenOffice.org.

9 Boulevard Manager and Visualization Layer

This controls the expert system, connection to instance of OpenOffice.org, Interceptor and also ensures the visualization of Boulevard. The project is written in JAVA language using Netbeans IDE and its OpenOffice.org plug-in [9] for easy extending of OpenOffice.org. We used this plug-in to create a so-called extension, (in other words plug-in), which can be used for extending OpenOffice.org without modifying the source of OpenOffice.org. For boulevard visualization JAVA [4] and SWING library [15] are used, which is a newer and more sophisticated library than earlier AWT(Abstract Window Toolkit).

9.1 Boulevard Manager

Boulevard manager is an important part of Boulevard, it utilizes and controls Interceptor, CLIPS expert system and Boulevard Window. We will describe the process (Figure 22) more closely from the user command to redrawing Boulevard (if necessary). The process is initiated by an intercepted user command by OpenOffice.org Interceptor. The intercepted user command is asserted to the expert system as a `new-action` fact and inference is started. When inference ends, the `action` facts are read from the expert system memory and these facts are then used for visualization of Boulevard.

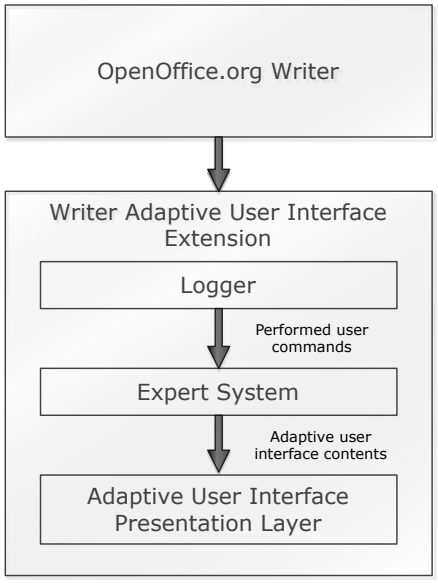


Figure 22: Boulevard internals

9.2 CLIPSJNI

CLIPSJNI [2] stands for CLIPS Java Native Interface, which is an interface for using CLIPS from Java environment. For Windows, it uses CLIPS DLL library[3], which is used as CLIPS run time and environment. CLIPSJNI provides almost full control of CLIPS as in standard CLIPS IDE. We can obtain facts from CLIPS environment as JAVA objects, which is very useful. We cannot use the precompiled CLIPSJNI jar, because there is a condition to have CLIPS DLL file contained in WIN32 system directory. This was an unacceptable limitation for our Boulevard implementation as an easy to install OpenOffice.org extension. Such installation would require Administrator access to write CLIPS DLL file to WIN32 system directory. So this is why we have compiled our own version of CLIPSJNI, where it is possible to select path for CLIPS DLL, which we have contained in our JAR file. Except this, there are not any other changes.

Inference is started by CLIPSJNI, but we do not use normal `RUN` command starting inference, but `RUN` with parameter of maximum number of inferences instead. In the current version, it is 1000. That was useful for unstable versions of the expert system, where cycling of expert system was possible, so inference never ended and thus OpenOffice.org stopped responding. With this limitation, the user just received a message about cycling of the expert system and then Boulevard was visualized in state where the inference stopped after 1000 steps. Also it is possible to see the current and previous status of Boulevard, so the problem can be easily debugged.

There is also a mechanism for easier debugging. In Boulevard options window (in Figure 23), we can enable Boulevard debug mode, and then we can step expert system inference. After every step we can see the current status of all stored facts, and also visualization in real Boulevard window.

We also always measure the number of inferences after every used user command. We can generally say by simple observation that even when there are many items, about 100, and the minimal-rank value is not 0, but a greater number like 0.01, the number of inferences is relatively small, mostly between 10 to 20 after every used user command.

9.3 Boulevard Visualization Framework

This layer takes output from the expert system as input, which describes what Boulevard should look like now. After the inference finishes, we read using CLIPSJNI all facts of type of `action`. Then we use all important slot values like the primary and secondary position, if an item should be placed in boulevard, ui-element and parameters. This is all the information we need to visualize Boulevard. We also remember current Boulevard state and when we receive a

[3]Dynamic Link Library – shared library concept used in Microsoft Windows, file has usually DLL extension

new version from the expert system, we compare these versions and determine, which items are newly added, which items changed its position and which items are removed. This information we use mainly for animations. See more about animations in Chapter 9.5.

9.3.1 User Interface Elements Context Synchronization

In this section we will explain the issue, which lies in synchronizing Boulevard items status with currently opened OpenOffice.org window and its cursor position in document which indicates context. Simply put, for example, whether the user click in the text to where boldface is on or off, the boldface icon in toolbar must also be enabled or disabled by the current cursor position, which we call in this case context. This problem makes implementation of an alternative user interface in the existing software very difficult. Information from loggers is not enough since the context can change without performing any user command and without redrawing any icon, which could be logged by some user interface logger. Luckily, OpenOffice.org provides UNO objects called Dispatchers, which are used internally for performing user commands, and also for determining the status of such commands. Every command in OpenOffice.org has its own Dispatcher, which has its own unique UNO Command URL. By this URL, we can obtain an appropriate Dispatcher, and register our own listener. Listener is an object, which generally has a method called Listen. We create such an object and implement Listen method on our own. Then we register our listener into the Dispatcher, by calling method of the dispatcher "register listener", with our listener as parameter. Then the dispatcher call ours listener method listen, when status of dispatched command changes. This is how we handle this problem in OpenOffice.org.

Every user command can be enabled or disabled, for example, we cannot delete a table column, if we do not have the cursor in the table. However, it is more complex issue since many user commands have more possible statuses than active or inactive and enabled or disabled. Commands with complex parameters, like the currently used font and its size are problematic. Such commands must be handled specially, since such information is not presented as a basic type like Boolean or integer, but as a standard UNO object like FontHeight. In such cases, we must have predefined how to handle these objects and how to obtain the desired value. This is why such a issue cannot be solved by a general solution, we must take special care of every command with more complex parameters than Boolean. Currently, such commands are: Font, FontHeight, Style, Zoom, BackGroundColor, FontColor, FontHeighlight and Find.

After the information from the listener about changing the user command status is received, change of the status of our visualized action must be performed. For example, select the appropriate align icon, and deselect the other currently unused align icons.

A relatively special case is colors, as we have to generate the appropriate icon with the right color. In OpenOffice, there is only one icon for all colors. Precisely, color is expressed in the font color icon as a rectangle at the bottom. We have to repaint this rectangle by a particular color and generate a new icon, which is then used as a icon for color command with a special parameter. There is another specialty about color, and that is that they are not unique commands in Boulevard, for more see Section 5.1.1. There is also an interesting problem with color icon design in OpenOffice.org: some users interpreted color icons as underline command (color icons are on Figure 13).

9.4 Boulevard Layout Manager

The layout manager is responsible for what Boulevard looks like. We have implemented our own layout manager, because we didn't find any standard SWING layout manager, which served our purpose well. The layout manager is called every time, when Boulevard content or window size are changed. It calculates the position for every item. The layout computation depends on the currently used layout style — horizontal or vertical. Computation follows the same direction as item prominence in Boulevard. Simply speaking, the difference between the horizontal and the vertical layout lies in swapping the computed x coordinates with the computed y coordinates.

The proper icons for representing user commands were needed, so we obtained them from OpenOffice.org source and included in our extension. We did not find the way to make the Boulevard window a native window of the OpenOffice.org, so we created a new ordinary window for our purposes. But this window could not be really ordinary since such a window contains a big system title bar, which is undesirable for Boulevard. We wanted the window to look as a part of OpenOffice.org as much as possible, so we used a so-called undecorated window and implemented our own small title bar, moving and resizing features. Other specialty of our window is a always-on-top behavior, since the Boulevard window can then be in the OpenOffice.org window and never goes under another window.

Such implementation has some limitations, except the different look of the Boulevard window from the OpenOffice windows, there is also a focus issue. When the user uses Boulevard, OpenOffice loses focus and vice versa, when the user uses OpenOffice, Boulevard loses focus. This causes a issue with tooltips, which is an important part of the user interface.

The layout manager also initiates painting of items. Not just Boulevard items, but also auxiliary items for better orientation in Boulevard, such as line separators. The layout manager also checks if every item can be painted to Boulevard properly in the sense of item size. If an item does not fit in the Boulevard window properly, it is not painted at all. This is useful for the user, since he can set the intended number of items in Boulevard by a simple change of Boulevard window size, and not only on the primary position, but also on the secondary position.

The window size, particularly the ratio between height and width of window also induces the layout — horizontal or vertical.

The layout manager also initiates animations, and not only the entire animation, but also every animation step. We use the standard SWING timer for such purpose. The layout manager holds instance of timer, which calls all interested listeners.

The layout manager also cares about cursor position. Boulevard cannot change its content when the cursor is in the Boulevard window, because such behavior could be confusing and annoying for user. Imagine that the user once uses a command by clicking on it and after that, without moving the cursor, under the cursor there now appears a different command. Such behavior would increase the number of errors made by the user and it would also be very annoying.

9.5 Animations

The reason why we implemented animations into Boulevard is simple — they helps the user to better understand and perceive Boulevard. For example, which item was moved and where, which item was removed or added. If the Boulevard window be just repainted in a second, the user would not even notice the change, not even knowing exactly which item was changed and how. We implemented three types of animations: The first and most important animation is used for item, whose location is changed, but in both states the item was painted into Boulevard. We animate smooth motion of an item in Boulevard. The animation consist of ten steps and every step has the same duration, which is 50ms. The number of pixels by which the item moves is variable during the animation — less at the beginning and the end of the animation than in the middle. That is because we wanted a smoother movement.

Another animation is used for adding a new item into Boulevard. This is because we want to notify the user about a new item (command) which they can use. The animation looks a like "blow-up". The item is gradually magnified from the size of a few pixels to multiple size, and then downsized back to regular size. It all takes less than a second. This animation cannot be implemented by simply changing the size of the SWING object (container). Changing directly the size of items is a slow method, since it initializes the layout manager, which can also initiate another layout manager of the contained item. For these reasons, this animation is implemented in the following steps. At the beginning, we have an item which we want to animate. We "tell" this item to paint itself, but not into the Boulevard window, but into an image object. Then we use this image for animations. Changing size of an image is a much simpler task than changing the size of a complex SWING button object. But to keep an animated item in the right place during the animation, we have to change its position in every step, since the item is positioned in the left top corner, not the center.

The last type of animation is to remove an item from Boulevard. This animation is very similar to the above mentioned "blow-up" animation. This animation is almost inverted with a few exceptions. The item is not magnified but decreased. Also the position is not changed after every step of the animation. The animation looks like the item is collapsing to its left top corner. The animation is different, not just inverted for better recognition by the user.

9.6 Visualization Result and Future Work

This part of Boulevard is much important, since it indicates how the user sees and perceive Boulevard. The visualization has also the most implementation issues. There should be done much more in the future, namely:

- Animations are poorly implemented. Animation was not the goal of this work, but we could not resist even trying to play with them. So we implemented basic animations, but in the current state it is hard to implement more types of animations. Reimplementation of the layout manager with an appropriate animation framework would be great.
- The Boulevard window is not a part of OpenOffice, causing focus and tooltips issues. This induces reimplementation of the entire visualization part of Boulevard.

9.7 Boulevard Installation

Disclaimer: Boulevard is definitely not software for end-users, it is a prototype, which serves as a tool for evaluation of an adaptive user interface design. There is absolutely no warranty. However, if someone wants to experience Boulevard, here are simple installation instructions: Current version works with OpenOffice.org 3.2.1 Writer (default installation) under Microsoft Windows XP Service Pack 3 and JAVA 1.6.20 (however, Boulevard was reported as working also under the Windows 7). Please notice the Interceptor limitations described in chapter 7.1.4, which are the same as for the Boulevard.

- Open the file `Boulevard.oxt` by double clicking or with OpenOffice.org Extension Manager.
- Confirm installation.
- After installation, relaunch OpenOffice.org. Please be careful with the OpenOffice.org speed luncher, which may cause the incomplete relaunch of the OpenOffice.org.

Boulevard has an administrative window (Figure 23) implemented, which can be invoked from the OpenOffice.org Writer's menu. In the administration window it

is possible to change Boulevard's parameters, such as: Recency-Frequency ratio, minimal rank, rank tolerance. We can also enable or disable some Boulevard features, like animations, sweeping-back and adaptive representation. The administration window also helps to debug the expert system by stepping inference and offers a function to save and load Boulevard from a file. Other tabs contain just some debug output.

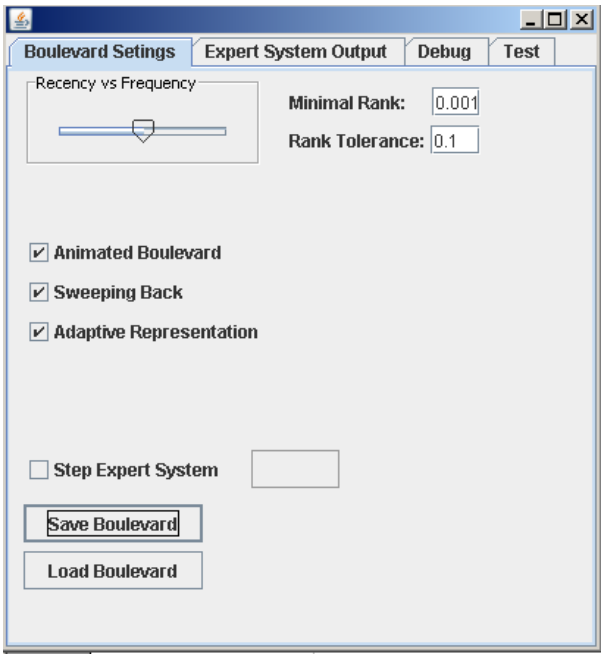


Figure 23: Boulevard administration window

10 Boulevard Usability Test

An important part of work was a usability test. We were very curious about the test results. User experience consists of two basic aspects: quantitative (e.g., task time cost and error rate) and qualitative (user's satisfaction). We can measure the quantitative aspect by a logger, however, from a log analysis we cannot extract real user's satisfaction, which is also important. For measuring user's satisfaction questionnaires are used. There are many types of usability tests, each is suitable for a different stage of software development:

- Early prototype testing — may be done by discussion with the user after prototype presentation.
- Prototype testing — may be done by direct observation of the user's behavior and discussion or by other more sophisticated method, like the SUS [18].
- Almost complete prototypes — usually done by performing benchmarks; measuring the time taken by the tester to complete a given task, may be combined with questionnaires.
- Complete prototypes, software for end users — long-term studies, collecting volumious data from a high number of users, analyzing and planning changes for improvements for future versions.

10.1 Previous Test

In the previous work, Jakub Černek performed only simple tests of Boulevard usability by direct observation of user's behavior by the screen recording software. In his thesis [20] there are four videos (the videos are also available on Dostál's www page [25]) taken by screen recording software. There is also some kind of conclusion, which is more a commentary to the videos than an evaluation.

10.2 The Test

Unfortunately, because of time limitations we performed only a pilot study — a preliminary evaluation of the implemented prototype. There should be more tests in the future. Our test consist of a simple benchmark, we measured the time needed to complete a given task and an error rate of three different interaction styles: toolbar, menu and Boulevard interaction style. After that the testers filled in questionnaires.

The test was structured as follows:

1. Briefing
2. Meet the OpenOffice.org Writer

3. The test
4. Toolbar test
5. Menu test
6. Boulevard test
7. Questionnaire fill in

The test began with a simple briefing about user experience and information needed for the test, like what a toolbar, a menu and Boulevard are. The briefing took about ten minutes accompanied by five simple slides. There were some general and particular problems about which we also talked about in the briefing:

- We are not testing users, we are testing the software.
- The test is not about document content, but about performing user commands (activating / deactivating). However, we created an initial document for the test (see Appendix B).
- However, some actions need the correct context (cursor position) for activation, for example, issuing "Copy" command assume a selected part of the document.
- When users look for a tested user command, they should not activate commands which they do not want to. This has two main reasons: (1) we measured various aspects of every tested command, apart from duration of activation also the number of errors made by the user — the number of wrongly activated actions. (2) Due to Interceptor implementation, which causes that if the user opens a dialog at first, then makes no changes and closes the dialog by clicking OK, information about performing action named mostly as a dialog is logged. In this case we measure one error made by the user, since the user performed a different command than he should. But if the user clicks on CANCEL, then nothing is logged, so we do not measure any error made by the user. This was presented to the testers as important.
- There was the abort button on the test window for case when the user cold not complete the task.

The next part was designated mainly to users who have never worked with OpenOffice.org Writer. The users had about three minutes to meet the OpenOffice.org Writer. As an observer I have to say that most users did nothing. Then the main test began by the toolbar part — choosing user commands strictly from a toolbar (using only a toolbar interaction style). List of tested commands is in Table 3.

UNO Command	Description
Bold	Tučné písmo (Bold)
Italic	Kurzívu (Italic)
Underline	Podtržení (Underline)
FontHeight	Libovolnou velikost písma (Font Height)
CharFontName	Libovolný font písma (Font)
RightPara	Zarovnání doprava (Align Right)
CenterPara	Zarovnání nastřed (Align Center)
DefaultBullet	Odrážky (Bullets)
DefaultNumbering	Číslování (Numbering)
IncrementIndent	Zvětšit odsazení (Increment Indent)
FontColor	Libovolnou barvu písma (Font Color)
Undo	Zpět (Undo)
Copy	Označte první slovo a zkopírujte jej do schránky (Copy)
Paste	Vložte slovo ze schránky na libovolné místo (Paste)
InsertTable	Vložte tabulku na libovolné místo (Insert Table)

Table 3: Toolbar test commands

There were fifteen particular commands in the toolbar test and each command was randomly repeated four-times, and for all the participants in the same order. The total number of toolbar tests is sixty.

After that the menu test began, where users should use only menu interaction style. This part was the most difficult for several reasons: (1) menus contains all possible commands of OpenOffice.org Writer, instead of a toolbar and Boulevard. Some commands are accessible directly from a menu, however many commands are accessible from a tab in a dialog, invoked from a menu. (2) There was also a naming issue, reported by some testers: in OpenOffice.org Writer is the "font dialog" named in the menu as *Character...*, not *Font...* (like in the Microsoft Office). This was reported by some testers as confusing. There were twenty actions in the menu part and each was repeated three times — sixty tests. A list of commands tested in the menu part is in Table 4.

When the menu part finished, Boulevard was displayed on the user's screen. Boulevard was build on the user's behavior in the previous test parts (toolbar and menu). So every user's Boulevard was unique. There was a problem if the user did not find the command in the previous parts, then such a command was not contained in Boulevard, since the user did not used such a command. Such a behavior of Boulevard is correct, so we did not care about this issue in the test design, however, it complicated the evaluation of the test, especially the error rates.

Commands in the Boulevard test were selected as union of commands from the toolbar part and the menu part, namely: Ruler, WordCountDialog, Zoom, Table-

UNO Command	Desctiption
SelectAll	Vybrat vše (Select All)
Italic	Kurzívu (Italic)
Underline	Podtržení (Underline)
FontHeight	Libovolnou velikost písma (Font Height)
CharFontName	Libovolný font písma (Font)
ViewBounds	Zobrazit hranice textu (View Bounds)
LeftPara	Zarovnání vlevo (Align Left)
RightPara	Zarovnání vpravo (Align Right)
SplitCell	Rozdělit buňky (Split Cells)
DeleteRows	Odstranit ·řádky z tabulky (Delete Rows)
FontColor	Libovolnou barvu písma (Font Color)
Undo	Zpět (Undo)
Copy	Označte první slovo a okopírujte jej do schránky (Copy)
Paste	Vložte slovo ze schránky na libovolné místo (Paste)
InsertTable	Vložte tabulku na libovolné místo (Insert Table)
DeleteColumns	Odstraňte libovolný sloupec z tabulky (Delete Columns)
TableBoundaries	Hranice tabulky (Table Boundaries)
Zoom	Lupa (Zoom)
WordCountDialog	Počet slov (Word Count)
Ruler	Zobrazte pravítko (Ruler)

Table 4: Menu test commands

Boundaries, DeleteColumns, DeleteRows, SplitCells, LeftPara, RightPara, CenterPara, ViewBounds, SelectAll, Bold, Italic, Underline, FontHeight, CharFontName, DefaultBullet, DefaultNumbering, IncrementIndent, FontColor, Undo, Redo, Copy, Paste, IntertTable. There were twenty-six actions in the Boulevard part and each was repeated three times, which is seventy-eight tests of Boulevard interaction style.

10.3 Testing Facility

To conduct the test, we needed Interceptor to gain the information about the user's activity in OpenOffice.org Writer. We used such information not only for measuring quantitative aspects of the test, but also for an implementation of the testing software itself. The testing software shows the user instructions about the currently tested command and leads the user through the entire test. When the user successfully performs a given command, the testing software displays the next instructions automatically. Such behavior would be almost impossible to implement without the Interceptor API. The testing facility is implemented directly into Boulevard. The internal logic is simple:

1. Creates the test window and sets the intended size and position of the OpenOffice.org Writer window and the Boulevard window (to accomplish uniform conditions for all testers).

2. Downloads the test design file from the server using SSH File Transfer Protocol [4].

3. Loads the test design file — XML file containing tested commands with an appropriate interaction style and other test related data.

4. Runs test — show every particular test task to the user, measure time from the start and check for accomplishment criteria, like command and required interaction style to past the test successfully, if defined.

5. When the entire test finishes, creates the log of test and uploads the file to the server.

Since our participants speak Czech, we had to translate the tested actions into Czech.

The tested commands were selected with respect to the frequency of usage, given by [29], however, there are some exceptions given by the interaction style.

The quantitative aspect of test was measured using Interceptor, the log structure follows (enumerated in same order as in log):

1. Boolean — Did the user abort the test?

2. integer — the number of fails (number of wrong activated actions)

3. integer — the test number

4. long — accomplishment duration time in milliseconds

5. string — the tested command

6. string — the interaction style used to initiate command

7. string — the interaction style used to execute command

8. integer — the test phase (1 indicates the toolbar phase, 2 the menu and 3 the Boulevard)

9. string — the user's ID, which consist of the the hostname, the username, the date and the time of test start.

10. string — command parameters from Interceptor

[4] *Poznamka: Secure Shell protocol (SSH) extension to provide secure file transfer*

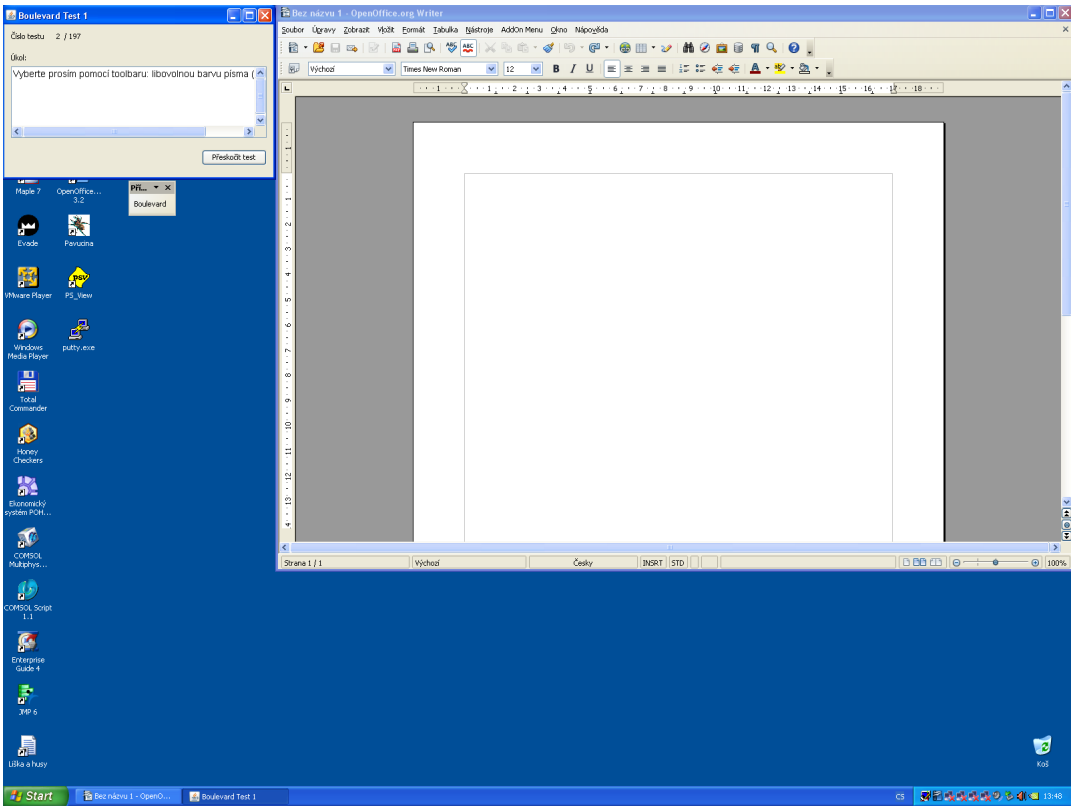


Figure 24: Tester's screen

Here is an example of a line from the log:

```
false,0,1,999,FontHeight,menu,dialog,2,U502-1-usr-2010-12-09-18-19,
 "FontHeight.Height:22.0;FontHeight.Prop:100;FontHeight.Diff:0.0;"
```

The qualitative aspects were measured using a questionnaire. We used the Lime Survey software [5] for an on-line survey, because of easier data collecting and easier evaluation. Lime Survey offers an export directly to "R" statistical software [13], which was used for statistic evaluation. Users completed the questionnaires just after they finished the test. The questionnaire consisted of sixteen questions, see Appendix C.

10.4 Test Result

Let us start with more detailed information about our participants. Our sample do not follow the normal population. The test were performed by eighteen subjects of various age (M= 30.5, Min.= 21, 1st Qu.= 23, Mdn= 25, 3rd Qu.= 32.25, Max= 55, Sd= 11.33). There were 14 (78 %) males and only 4 females (22 %). The highest completed education level of subjects: high school — 7 subjects (38.89 %), university degree — 11 subjects (61.11 %). In our sample

there were 8 students, 3 academics or teachers, 3 administrative workers, one M.D. and one unspecified. The participants have long-term experience with computers (M= 12.94, Min.= 5, 1st Qu.= 10, Mdn= 11.5, 3rd Qu.= 15.5, Max= 25, Sd= 4.74). The number of years of working with text editor is very similar to working time with computers. Mode of computer skills on scale from 1 from to 5, where 1 means no skills and 5 indicates pro user, is 4.

Unfortunately, not all participants completed the test successfully, however, all of them worked with Boulevard for long enough, so we evaluated at least their questionnaires. So we have only 12 complete logs from the test.

10.4.1 Quantitative Evaluation

At first, we filtered out the aborted tests and tests that took more than 30 seconds to complete. Out filtered data were evaluated later as error rates. After that the data were aggregate by the median for every user and every test phase (toolbar, menu and Boulevard) and also such times were expressed in percentage against the toolbar phase, which is here the 100 % (Table 5).

ID	Toolbar time (ms)	Menu time (ms)	Boulevard time (ms)	Toolbar %	Menu %	Boulevard %
1	2656	7125	2750	100	268	103
2	3109	5969	2984	100	191	95
3	4734	8758	4702	100	185	99
4	5625	7422	4516	100	131	80
5	5938	9516	4984	100	160	83
6	5296	7219	6071	100	136	114
7	5922	8531	3484	100	144	58
8	2726	6031	3235	100	221	118
9	4344	7859	6203	100	180	142
10	2312	5882	2640	100	254	114
11	7852	11757	10219	100	149	130
12	3500	6992	3797	100	199	108

Table 5: Aggregated data

Table 5 shows that the menu is the most time-consuming interaction style, however this was a presumable result since menu contain many more items than a toolbar and Boulevard. The arithmetic mean of percentage for a menu is about 185 % against a toolbar. Boulevard was slighly more time consuming than a toolbar. The mean of percentage related to user for Boulevard is 104 %. Toolbar time : 4.5 s mean and 4.5s median, menu time: 7.8s mean and 7.3 median, Boulevard time: 4.6s mean and 4.2 median. If we compare the median, Boulevard beats all other interaction styles.

We also performed a repeated-measures one-way ANOVA to find out, if there is statistical significance in the used interaction styles. Mauchly's test did not show a violation of sphericity against interaction style($W(2) = 0.77$, $p = 0.26$). With one-way repeated-measure ANOVA, we found a significant effect of interaction style on time ($F(2, 22) = 63.7$, $p < 0.01$, partial $\eta^2 = 0.97$). A post-hoc pairwise comparison, computed using the Bonferroni adjustment, revealed significant differences between toolbar and menu ($p < 0.01$) interaction style, and between Boulevard and menu ($p < 0.01$) interaction style.

Error rates correlate to the above evaluation. The number of aborted or timed-out tests (previously out-filtered) in the toolbar test is 19, in the menu test 67 and in the Boulevard test 24. The number of fails (activation of the wrong activated commands) in the toolbar test is 450, in the menu test 709 and in the Boulevard test 671.

10.4.2 Questionnaire Evaluation

The printable version of the questionnaire can be found in Appendix C. The questionnaire started with two ranking questions: The participants were asked to order the interaction styles (toolbar, menu and Boulevard) by subjective speed (time consumption). As subjectively fastest interaction style 12 participants ranked Boulevard, 6 toolbar and none menu. In the second place (second fastest): 11 toolbar, 6 Boulevard, 1 menu. As the slowest interaction style 17 participants voted for menu and one for toolbar.

The second ranking-question was about the satisfaction with particular interaction styles. Ten participants ranked Boulevard as the most satisfying interaction style and eight toolbar. On the second place 10 participants voted for toolbar, 8 for Boulevard. As the least satisfying interaction style was considered menu by all 18 participants. The participants were asked to evaluate the following on the five-point Likert scale [33], where 1 mean "I strongly disagree" and 5 mean "I strongly agree".

1. "I understand the basic principles and behavior of Boulevard" (Mode 4).
2. "Boulevard is intuitive and predictable" (Mode 4).
3. "Boulevard is interesting" (Mode 5).
4. "I like Boulevard" (Mode 5).
5. "I would use Boulevard" (Mode 5).

We were somewhat surprised by suspiciously so good users response to Boulevard.

10.5 Test Result and Future Tests

We were very pleased by the test result. The quantitative measurement did not showed statistically significant difference between the toolbar and the Boulevard interaction style. However, Boulevard is the best interaction style by the qualitative aspect, which is also important. We have shown that adaptive user interfaces can be well received by the users. In my opinion, there can be done more evaluation on the already collected data, such as:

1. To reveal a dependence between the measured speed of a particular interaction style and the perceived speed obtained from the questionnaire.

2. Find particular user commands with the significantly higher error rates depending on the interaction style. Some such commands are expected, like align-related commands, since they are visually similar and their positions are unstable in Boulevard in contrast to the toolbar or menu.

It is important to perform various other tests, particularly, test every feature of Boulevard, such as: user commands ranking (short-term vs long-term), adaptive representation, sweeping back, the general impact of animations on the acceptance of Boulevard by users and test many types of such animations. Also it is important to perform long-term studies. Long-term studies are substantially difficult, since such studies cannot be done on prototypes. However, such studies are important, because users can like certain behavior during the short test, but the same behavior can be annoying in a long-term use. Some long-term observations can be found in Chapter 11.

11 Writing This Thesis With Boulevard

This work has been written on OpenOffice.org Writer with Boulevard. We did not want to miss the opportunity to try Boulevard in real work, so we did not use the good old-fashioned LaTeX (typographic engine without GUI) at first, but the OpenOffice.org. This decision was not a victimless since OpenOffice.org is definitely not a good tool for writing a thesis (for example, does not support BibTeX). On the Figure 25 is the resulting, personalized, Boulevard as it was at the end of my work on this thesis. I did not have much time to perform a long-term study, so here are some of my observations:

- Boulevard is better a for less experienced user who uses the word processor as a typewriter (thus uses various formatting commands), but not so suitable for a user who uses styles, since they ensure most of functionality automatically. However, Dostál's observation [29] indicates that a lot of users not use styles.
- At the beginning, Boulevard was annoying and stressful for me, however, the reason can be that I was suspicious about it's proper functionality, which cold be caused by the fact that I am the author of implementation.

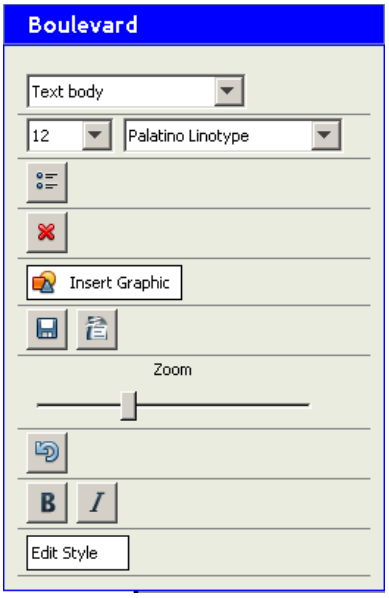


Figure 25: Resulting personalized Boulevard

However, the current version of my thesis is obviously not written in OpenOffice Writer. I have rewritten my almost-complete thesis to LaTeX, since I was very unsatisfied by OpenOffice.org.

12 What Have I Done and How?

Because of the wide aim of this work and due to a close cooperation with my thesis "leader" — supervisor, Martin Dostál, it is important to specify my role.

- Expert system design — This was the second version of the Boulevard expert system, I have done the first one in my Bsc. thesis. We wanted the second implementation to be almost perfect, so many rules are consulted with Martin Dostál.

- OpenOffice.org Interceptor — We assume Interceptor as critical part of any AUI and there is no such logger as Interceptor, which combines macro recorder approach and user interface events approach. The implementation was not easy and it contains some unconventional solutions, which is why the implementation took much more time than we expected. Sometimes we got into a dead end, so we both cooperated and studied OpenOffice API [11] to solve the problems. Here also lies the part of Dostál's work.

- JAVA implementation of Boulevard — This implementation was relatively easy since we learned how to interact with OpenOffice when working on Interceptor.

- Testing facility — The software used to perform test was designed by me.

- Benchmark test design and evaluation — The test design was done by me (of course, with some consultations), which is probably the reason why it is far away from perfect, but it helped us to better understanding of usability testing in practice.

During the work on Boulevard we also tried to publish some Boulevard-related papers. I am a co-author of the following papers: [30], [31] and [32].

Conclusions

The main objective of this thesis was to implement the Boulevard in the OpenOffice.org Writer and perform a usability study. We have presented the details of the Boulevard implementation and also of the by-product: the OpenOffice.org Interceptor — a hybrid approach logger.

The preliminary usability study of Boulevard fall out well, since Boulevard is comparable to toolbars by quantitative measurement (no statistically significant difference found). However, Boulevard was rated as better interaction style than toolbar by most of test participants (described in detail in Chapter 10.5).

The OpenOffice.org Interceptor — a hybrid approach logger, which originated as a by-product of Boulevard, turn out as an interesting and quite unique logger. The logger combines a user interface events logging approach and macro-recording logging approach at the level of underlying function calls. The combination of both the logging approaches makes possible to log the performed user commands with parameters and interaction style.

Future development of the Boulevard is challenging. It should be focused mainly on usability tests and more appropriate visual layer implementation (described in Chapter 9.6).

It would be great to implement Boulevard as an integral part of operation system. Such Boulevard would support various applications. Another possible direction in the development of Boulevard is a Boulevard for people with disabilities, which adapts to their particular disability. It could be interesting to implement a workflow analysis feature into Boulevard, which can automate often-performed tasks. A combination with a recommender system would be helpful as well.

The recommender system in Boulevard could be based on collaborative filtering — central collecting and processing of a lot of data from a high number of users. Such approach has two phases: finding usage patterns in collected data and then offer such found patterns to users.

Other future work should be aimed at developing loggers. Improvement of the Interceptor, and developing another Interceptor-like logger for other software, such as Microsoft Word 2003 and 2007. Such loggers can make the Boulevard implementation into other software than OpenOffice.org possible in the first place. However, also testing and measuring user's behavior on other platforms would be very useful, like comparison between WIMP-based Microsoft Word 2003 and Ribbon-based Microsoft Word 2007.

And, of course, continue in developing Boulevard, implementing all known missing features, such as adaptable features and delivering the public available version of the Boulevard. But most important is to test and verify all the already implemented Boulevard core behavior, such as adaptive representation, ranks (recency vs frequency) and sweeping back.

Závěr

Hlavním úkolem této diplomové práce bylo implementovat Boulevard do textového editoru OpenOffice.org Writer a provést testy jeho použitelnosti. Podrobně jsme ukázali detaily implementace Boulevardu a loggeru OpenOffice.org Interceptor — vedlejšího produktu Boulevardu.

Prvotní studie použitelnosti Boulevardu dopadla dobře, jelikož dle kvantitativního měření je Boulevard porovnatelný s toolbarem (nebyl nalezen statisticky významný rozdíl). A navíc, Boulevard byl většinou účastníků testu označen jako lepší interakční styl než toolbar (více v kapitole 10.5).

OpenOffice.org Interceptor — hybridní logger, který vznikl jako vedlejší produkt Boulevardu, se ukázal jako jedinečný. Interceptor kombinuje dva způsoby logovaní: zachycování událostí v uživatelském rozhraní a zachycování uživatelských akcí pomocí makrorekordéru na úrovni volání obslužných funkcí. Kombinace těchto dvou přístupů umožňuje logovat ne jen uživatelské akce a jejich parametry, ale také interakční styl.

Další vývoj Boulevardu by měl pokračovat, a to zejména uživatelské testy a reimplementace vizualizační vrstvy Boulevardu (více v kapitole 9.6).

Je mnoho dalších směrů, kterými Boulevard lze rozvíjet, například Boulevard jako součást operačního systému, adaptivní rozhraní pro lidi s postižením, analýza work-flow a následná automatizace často prováděných úkonů.

V Boulevardu by se mohl také uplatnit takzvaný recommender system, který by mohl být navíc založený na technologii collaborative filtering — centrální sběr a zpracování velkého možství dat od velkého množství uživatelů. Tento přístup má dvě fáze: nalezení uživatelských vzorů v datech a poté nabízení nalezených vzorů dalším uživatelům.

Další práce by se mohly zaměřit na implementaci loggerů do jiných platforem než OpenOffice.org. Takový logger by mohl umožnit implementaci Boulevardu pro jiné aplikace a také umožnit provádět některé uživatelské testy, jako například srovnání Microsoft office 2007 (Ribbon User Interface) s Microsoft Office 2003 (WIMP).

Nicméně je třeba nejdříve implementovat všechny možnosti a funkce Boulevardu a proměnit náš prototyp v aplikaci vhodnou pro koncové uživatele. Také provést všechny chybějící testy přímo zaměřené na určité vlastnosti Boulevardu, jako například adaptivní reprezentace, sweeping-back a ohodnocování prominence akcí v Boulevardu.

References

[1] *ADaptive User Interface System ADUS.* http://sid.cps.unizar.es/ANTARCTICA/ADUS/.

[2] *CLIPS Java Native Interface.* http://clipsrules.sourceforge.net/CLIPSJNIBeta.html.

[3] *CLIPS User's Guide.* http://clipsrules.sourceforge.net/documentation/v630/ug.pdf.

[4] *Java.* http://www.java.com.

[5] *LimeSurvey.* http://www.limesurvey.org/.

[6] *The Lisa project.* http://lisa.sourceforge.net/.

[7] *LISP Works CAPI.* http://www.lispworks.com/products/capi.html.

[8] *List of similar meta-languages for device-independent user interfaces.* http://www.usixml.org/index.phpmod=pages&id=58.

[9] *OpenOffice NetBeans Integration.* http://wiki.services.openoffice.org/wiki/OpenOffice_NetBeans_Integration.

[10] *OpenOffice.org Accesibility API.* http://ui.openoffice.org/accessibility/unoapi.html.

[11] *The OpenOffice.org API Project.* http://api.openoffice.org/.

[12] *OpenOffice.org Press Kit.* http://marketing.openoffice.org/press_kit.html.

[13] *The R Project for Statistical Computing.* http://www.r-project.org/.

[14] *SUPPLE: Automatic Generation of Personalizable User Interfaces.* http://www.cs.washington.edu/ai/supple/.

[15] *What is SWING?* http://download.oracle.com/javase/tutorial/ui/overview/intro.html.

[16] *Developing an Adaptive User Interface in Eclipse.* the Eclipse Technology eXchange Workshop at European Conference on Object Oriented Programming, 2006.

[17] Jason Alexander and Andy Cockburn. Appmonitor: a tool for recording user actions in unmodified windows applications. *Behavior Research Methods*, 40(2):413–421, May 2008.

[18] J. Brooke. SUS: a" quick and dirty" usability scale. *Usability evaluation in industry*, pages 189–194, 1996.

[19] C.S. Carr. Network subsystem for time sharing hosts. RFC 15, September 1969.

[20] Jakub Černek. Adaptivní uživatelská rozhraní. Bsc thesis, 2008.

[21] Olivier Chapuis and Nicolas Roussel. Metisse is not a 3d desktop! In *Proceedings of the 18th annual ACM symposium on User interface software and technology*, UIST '05, pages 13–22, New York, NY, USA, 2005. ACM.

[22] Alain Colmerauer and Philippe Roussel. History of programming languages—ii. chapter The birth of Prolog, pages 331–367. ACM, New York, NY, USA, 1996.

[23] Richard. Conniff. What's behind a smile?, 2007.

[24] J. A. Cote-Munoz. Aida: An adaptive system for interactive drafting and cad applications. In M. Schneider-Hufschmidt, T. Kühme, and U. Malinowski, editors, *Adaptive User Interfaces: Principles and Practice*, pages 225–240. North-Holland, Amsterdam, 1993.

[25] Martin Dostál. *Interceptor homepage*. http://dostal.inf.upol.cz/oo-interceptor.html.

[26] Martin Dostál. *Základy tvorby uživatelského rozhraní*. http://dostal.inf.upol.cz/data/0910/URO/uro-18-12-2010.pdf.

[27] Martin Dostál. User acceptance of the microsoft ribbon user interface. In *Proceedings of the 9th WSEAS international conference on Data networks, communications, computers*, DNCOCO'10, pages 143–149, Stevens Point, Wisconsin, USA, 2010. World Scientific and Engineering Academy and Society (WSEAS).

[28] Martin Dostál. An analysis of usage patterns in utilization of interaction styles. In *In Proceedings of the Human-Computer Interaction International Conference*, CCIS. Springer, 2011.

[29] Martin Dostál. On the differences in usage of word processing applications. In *In Proceedings of the Human-Computer Interaction International Conference*, CCIS. Springer, 2011.

[30] Martin Dostál and Zdenek Eichler. Fine-grained adaptive user interface for personalization of a word processor user interface: Principles and a preliminary study. In *In Proceedings of the Human-Computer Interaction International Conference*, CCIS. Springer, 2011.

[31] Martin Dostál and Zdenek Eichler. A hybrid approach to user activity instrumentation in software applications. In *In Proceedings of the Human-Computer Interaction International Conference*, CCIS. Springer, 2011.

[32] Martin Dostál and Zdenek Eichler. A research framework for performing user studies and rapid prototyping of intelligent user interfaces under the openoffice.org suite. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. ACM, 2011.

[33] Diane R. Edmondson. Likert scales: A history. In *CHARM - the Conference on Historical Analysis and Research in Marketing*, volume 12, page 127, 2005.

[34] Zdenek Eichler. Adaptivní uživatelská rozhraní. Bsc thesis, 2008.

[35] Leah Findlater and Joanna McGrenere. A comparison of static, adaptive, and adaptable menus. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '04, pages 89–96, New York, NY, USA, 2004. ACM.

[36] Charles Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17–37, 1982.

[37] François. Fournier. *Recommender Systems:Technical Report and Literature Review.* http://knol.google.com/k/françois-fournier/recommender-systems/2eyelehhior52/1.

[38] Krzysztof Z. Gajos, Katherine Everitt, Desney S. Tan, Mary Czerwinski, and Daniel S. Weld. Predictability and accuracy in adaptive user interfaces. In *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, CHI '08, pages 1271–1274, New York, NY, USA, 2008. ACM.

[39] Leah Kaufman and Brad Weed. Too much of a good thing?: identifying and resolving bloat in the user interface. *SIGCHI Bull.*, 30:46–47, October 1998.

[40] Kukreja, Urmila, Stevenson, E. William, Ritter, and E. Frank. Rui: Recording user input from interfaces under windows and mac os x. *Behavior Research Methods*, 38(4):656–659, November 2006.

[41] Frank Linton, Deborah Joy, Hans-Peter Schaefer, and Andrew Charron. Owl: A recommender system for organization-wide learning. *Educational Technology & Society*, 3(1), 2000.

[42] Wendy E. Mackay. Triggers and barriers to customizing software. In *Proceedings of the SIGCHI conference on Human factors in computing systems:*

Reaching through technology, CHI '91, pages 153–160, New York, NY, USA, 1991. ACM.

[43] Joanna McGrenere, Ronald M. Baecker, and Kellogg S. Booth. An evaluation of a multiple interface design solution for bloated software. In *Proceedings of the SIGCHI conference on Human factors in computing systems: Changing our world, changing ourselves*, CHI '02, pages 164–170, New York, NY, USA, 2002. ACM.

[44] Joanna McGrenere and Gale Moore. Are we all in the same bloat? In *Graphics Interface'00*, pages 187–196, 2000.

[45] Reinhard Oppermann, editor. *Adaptive user support: ergonomic design of manually and automatically adaptable software*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1994.

[46] Stanley R. Page, Todd J. Johnsgard, Uhl Albert, and C. Dennis Allen. User customization of a word processor. In *Proceedings of the SIGCHI conference on Human factors in computing systems: common ground*, CHI '96, pages 340–346, New York, NY, USA, 1996. ACM.

[47] M.B Rosson. The effects of experience on learning, using, and evaluating a text-editor. Unpublished manuscript, 1983.

[48] Andrew Sears and Ben Shneiderman. Split menus: effectively using selection frequency to organize menus. *ACM Trans. Comput.-Hum. Interact.*, 1:27–51, March 1994.

[49] David C. Smith, Frank E. Ludolph, and Charles H. Irby. The desktop metaphor as an approach to user interface design (panel discussion). In *Proceedings of the 1985 ACM annual conference on The range of computing : mid-80's perspective: mid-80's perspective*, ACM '85, pages 548–549, New York, NY, USA, 1985. ACM. Chairman-Johnson, Jeff A.

[50] Joel Spolsky. *Strategy Letter IV: Bloatware and the 80/20 Myth*. http://www.joelonsoftware.com/articles/fog0000000020.html.

[51] Wolfgang Stuerzlinger, Olivier Chapuis, Dusty Phillips, and Nicolas Roussel. User interface facades: towards fully adaptable user interfaces. In *Proceedings of the 19th annual ACM symposium on User interface software and technology*, UIST '06, pages 309–318, New York, NY, USA, 2006. ACM.

[52] Terry Winograd. *Procedures as a Representation for Data in a Computer Program for Understanding Natural Language*. PhD thesis, MIT, 1971.

A Adaptive Menu Implementation in OpenOffice.org

```
Global PopupMenuContainer As Object
Global MenuBar As String
Global MenuBarSettings As Object
Global ModuleCfgMgr As Object
Global ooi As Object
Global ooi_listener As Object

Sub RegisterLogHandler
InsertMenu("Favorites")
ooi = createUnoService(_
 "org.openoffice.oointerceptor.XOOInterceptor")
ooi_listener = createUnoListener("ooi_", _
 "org.openoffice.oointerceptor.XOOInterceptorListener")
ooi.addOOInterceptorListener(ooi_listener)
End Sub

Sub ooi_listen(action, initiated, executed, time,_
               user, application, parameters)
 For i = 0 to PopupMenuContainer.getCount() - 1
  item = PopupMenuContainer.getByIndex(i)
  If item(0).Value = ".uno:" + action Then
   PopupMenuContainer.removeByIndex(i)
   Exit For
  End If
 Next i
 MenuItem = CreateMenuItem(".uno:" + action, action)
 PopupMenuContainer.insertByIndex(0, MenuItem)
 ModuleCfgMgr.replaceSettings(MenuBar, MenuBarSettings)
 DisplayTextStatusBar("Action: " &  action & _
 ", initiated: " & initiated & ", executed: " & _
 executed & ", time: " & time &  ", user: " & _
 user & ", application:" & application & _
  ",  parameters: " & parameters)
End Sub

Function InsertMenu(MenuName as string)
    MenuBar = "private:resource/menubar/menubar"
    ModuleCfgMgrSupplier = createUnoService(_
    "com.sun.star.ui.ModuleUIConfigurationManagerSupplier")
```

```
   ModuleCfgMgr = ModuleCfgMgrSupplier.getUIConfigurationManager(_
   "com.sun.star.text.TextDocument")
   MenuBarSettings = ModuleCfgMgr.getSettings(MenuBar, true )
   PopupMenu = CreatePopupMenu("vnd.openoffice.org:FavoritesMenu",_
    MenuName, MenuBarSettings)
   PopupMenuContainer = PopupMenu(3).Value
   MenuBarSettings.insertByIndex(0, PopupMenu)
   ModuleCfgMgr.replaceSettings(MenuBar, MenuBarSettings)
End Function

Function CreatePopupMenu(Command, Label, Factory) as Variant
   Dim PopupMenu(3) as new com.sun.star.beans.PropertyValue
   PopupMenu(0).Name = "CommandURL"
   PopupMenu(0).Value = CommandId
   PopupMenu(1).Name = "Label"
   PopupMenu(1).Value = Label
   PopupMenu(2).Name = "Type"
   PopupMenu(2).Value = 0
   PopupMenu(3).Name = "ItemDescriptorContainer"
   PopupMenu(3).Value = Factory.createInstanceWithContext(_
    GetDefaultContext() )
   CreatePopupMenu = PopupMenu()
End Function

Function CreateMenuItem(Command as String, Label as String) as Variant
   Dim MenuItem(2) as new com.sun.star.beans.PropertyValue
   MenuItem(0).Name = "CommandURL"
   MenuItem(0).Value = Command
   MenuItem(1).Name = "Label"
   MenuItem(1).Value = Label
   MenuItem(2).Name = "Type"
   MenuItem(2).Value = 0
   CreateMenuItem = MenuItem()
End Function

Function DisplayTextStatusBar(DisplayedText as String)
 oFrame = ThisComponent.getCurrentController().getFrame()
  oBar = oFrame.createStatusIndicator()
  oBar.start(DisplayedText, 100)
  wait 3000
   oBar.end()
End Function
```

B Starting Document in the Usability Test

Lorem Ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin gravida ultrices accumsan. Mauris quam nisl, dapibus vitae venenatis at, tincidunt vel tortor. Curabitur arcu eros, porta vel vestibulum id, tincidunt at magna. Suspendisse potenti. Vestibulum vulputate turpis enim. Ut sapien urna, sagittis at consectetur non, fermentum ut augue. Ut eu lorem nunc, eleifend molestie orci. Nunc purus nisl, faucibus in tincidunt in, scelerisque id mi. Duis egestas, odio id interdum pulvinar, augue est malesuada felis, at imperdiet nulla lacus in tellus. Pellentesque a felis augue. Vivamus imperdiet leo ac tellus mattis tristique. Ut ultrices est sit amet augue tristique tristique. Aenean vel nisl purus, at dapibus dui. Integer imperdiet fringilla tortor, sed aliquet justo placerat eu. Maecenas porta facilisis quam quis faucibus. Donec ac velit risus. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed libero neque, feugiat id fringilla id, sollicitudin pulvinar mi.

Lorem	**Ipsum**	**pulvinar**
ante	Pellentesque	eleifend
tellus	tristique	purus

Duis semper, arcu in laoreet eleifend, nibh mi sagittis ante, ac luctus lectus velit nec velit. Duis rutrum tempor facilisis. Donec sed quam sem. In nec quam ut magna adipiscing molestie. Integer diam ante, dapibus et consequat id, gravida vel nibh. Duis pharetra urna odio, vitae ullamcorper nulla. Aliquam malesuada tincidunt urna in tincidunt. Donec et orci eros. Nam vitae tellus et sapien feugiat ullamcorper sed sed ante. Pellentesque quis elit nisl, ut iaculis elit. Duis laoreet, ligula ac ornare laoreet, justo libero rhoncus ipsum, ornare congue lorem tortor bibendum augue. Mauris risus nunc, blandit et accumsan id, tincidunt vel nibh. Duis varius eleifend ante, pretium laoreet purus feugiat tincidunt.

tristique	**pulvinar**	**tempus**
rutrum	mollis	ultricies
pulvinar	elementum	rutrum

Vestibulum elit sem, tempor a tristique sed, dapibus id turpis. Mauris tempus ipsum accumsan arcu cursus lobortis. Proin pulvinar iaculis nunc sed hendrerit. Pellentesque sem ante, pellentesque eu commodo at, egestas at leo. Sed ac massa nisi, fringilla sagittis ante. Nullam tempor, nulla ut venenatis accumsan, augue arcu euismod nulla, eget euismod lorem nunc quis neque. Pellentesque vulputate congue nunc non viverra. Ut quam nisl, elementum quis tempus iaculis, rutrum vulputate lectus. Aliquam iaculis commodo egestas. Fusce pulvinar ultricies viverra. Sed magna arcu, pharetra at ullamcorper eu, rutrum vitae velit. Cras mollis posuere dui, vitae tempus dolor feugiat at. Proin eu metus ut libero convallis rutrum.

iaculis	**malesuada**	**viverra**

C Questionnaire

Tento dotazník slouží jako doplněk k 1. testu Boulevardu.
Dobrý den,
na závěr Vás prosím o vyplnění tohoto dotazníku.

Osobní informace o testerovi
1 [name]Uveďte prosím jméno a příjmení.
Prosím napište svou odpověď zde:

2 [age]Uveďte prosím Váš věk. *
Prosím napište svou odpověď zde:

3 [sex]Uveďte prosím Vaše pohlaví. *
Prosím zvolte pouze jednu z následujících možností:
Muž
Žena

4 [education]Uveďte Vaše nejvyšší dosažené vzdělání.
Prosím zvolte pouze jednu z následujících možností:
Základní
Středoškolské
Vysokoškolské

5 [work]Uveďte obor Vašeho zaměstnání.
Prosím zvolte pouze jednu z následujících možností:
Administrativa
Odborná práce
Student
Lékař
Právník
Obchod
Řemeslo
Umění
Vzdělávání, nebo věda a výzkum
Jiné

6 [comp-years] Kolik let pracujete s počítačem?
Prosím napište svou odpověď zde:

7 [word-years]Kolik let pracujete s textovými editory?
Prosím napište svou odpověď zde:

8 [comp-skills] Ohodnoťte Vaši celkovou znalost počítačů.
Prosím zvolte pouze jednu z následujících možností:
1
2
3
4
5
1=nízká,5=vysoká

Boulevard
Část dotazníku týkající se Boulevardu - testovaného software
9 [boul-rank-phases]Seřaďte toolbar, menu a Boulevard subjektivně podle rychlosti použití. *
Prosím očíslujte každé okénko podle preferencí od 1 do 3

Toolbar

Menu

Boulevard

10[boul-rank-phases-sat]Seřaďte toolbar, menu a Boulevard podle toho, jak Vám jednotlivé způsoby vyhovovaly.
Prosím očíslujte každé okénko podle preferencí od 1 do 3

Toolbar

Menu

Boulevard

11 [boul-text] V této části dotazníku prosím ohodnoťte následující tvrzení, a to takto:
1=naprosto nesouhlasím, 2=nesouhlasím, 3=nevím, 4=souhlasím, 5=naprosto souhlasím
12 [boul-understand]Pochopil jsem základní principy (chování) Boulevardu. *
Prosím zvolte pouze jednu z následujících možností:
1
2
3
4

5
1=naprosto nesouhlasím, 2=nesouhlasím, 3=nevím, 4=souhlasím, 5=naprosto souhlasím

13 [boul-intuitive]Boulevard je intuitivní a předvídatelný. *
Prosím zvolte pouze jednu z následujících možností:
1
2
3
4
5
1=naprosto nesouhlasím, 2=nesouhlasím, 3=nevím, 4=souhlasím, 5=naprosto souhlasím

14 [boul-interest]Boulevard mě zaujal. *
Prosím zvolte pouze jednu z následujících možností:
1
2
3
4
5
1=naprosto nesouhlasím, 2=nesouhlasím, 3=nevím, 4=souhlasím, 5=naprosto souhlasím

15 [boul-like]Boulevard se mi líbí. *
Prosím zvolte pouze jednu z následujících možností:
1
2
3
4
5
1=naprosto nesouhlasím, 2=nesouhlasím, 3=nevím, 4=souhlasím, 5=naprosto souhlasím

16 [boul-use]Boulevard bych používal.*
Prosím zvolte pouze jednu z následujících možností:
1
2
3
4
5
1=naprosto nesouhlasím, 2=nesouhlasím, 3=nevím, 4=souhlasím, 5=naprosto souhlasím

17 [boul-comment]Budeme rádi, pokud nám napíšete co se Vám na testu či Boulevardu líbilo nebo nelíbilo.
Prosím napište svou odpověď zde:

Odeslat Váš průzkum.
Děkujeme Vám za vyplnění tohoto průzkumu.

D Content of the Appended DVD

The appended DVD is organized as follows:

`bin/`
: Installable `Boulevard.oxt`, `Boulevard-test.oxt` and `Interceptor.oxt` extension files.

`doc/`
: This thesis in PDF and LaTeX (with all sources).

`src/`
: Complete sources of the Interceptor and Boulevard.

`install/`
: Installation file of the OpenOffice.org 3.2.1 for Windows

`videos/`
: Some videos with Boulevard or Interceptor.

`tests/`
: Collected data in the usability test (the logged data — `concatenated-file.csv`, anonymized data from the questionnaires — `Questionnaire.csv`), the test-design XML file (`test.xml`), R source code for data evaluation.