Designing a Context-sensitive Dashboard for an Adaptive Knowledge Worker Assistant

Ralf Biedert[1], Sven Schwarz[1], and Thomas R. Roth-Berghofer[1,2]

[1] Knowledge Management Department,
German Research Center for Artificial Intelligence DFKI GmbH
Trippstadter Straße 122, 67663 Kaiserslautern, Germany

[2] Knowledge-Based Systems Group, Department of Computer Science,
University of Kaiserslautern, P.O. Box 3049, 67653 Kaiserslautern

{ralf.biedert,sven.schwarz,thomas.roth-berghofer}@dfki.de

Abstract. We develop a context-sensitive dashboard to assist knowledge workers. The term context in our scenario defines explicit categories used to group observed native operations (NOPs) performed, and these NOPs correspond to actions undertaken during work. The dashboard consists of several small applications, widgets, that permit quick access to such entities as files, bookmarks, persons, and notes as well as to a visual backtrack on a per-context basis. The dashboard can be activated on demand. Meanwhile, in background, the user is constantly monitored: a user observation hub records high level user actions, system hooks take screenshots and keystrokes. These can be used by the visual backtrack widget to give a replay and help users to find a way back into the context again after some interruption through evaluating similar situations in the past.

1 Scenario: A Knowledge Workers Daily Work

Thursday is a typical work day in Joe's life. It is 11:30 am and he works on his paper for a context workshop: he wants to write an introductory usage story. On his desktop a word processor is open to edit a paper, a browser window shows a translation website, and his mail client contains a half finished mail to his supervisor. It is very likely that Joe will have to interrupt his work every quarter of an hour: He will be asked to supervise a student's master's thesis as he did the days before, fix a minor but long-lasting bug in some old code and, as every day, he has to decide which cantina offers the better food for lunch.

Studies [1, 2] show that knowledge workers using computers are confronted with many different tasks every day. Some of these tasks are closely related, others are not, and especially switching among unrelated tasks induces a large manual and cognitive overhead [3]. Applications have to be switched and other entities have to be brought to front. Also users have to recall the state before the previous interruption in order to avoid duplicate efforts.

Other frequently occurring problems are interleaved and cascaded interruptions, and returns to different contexts. While knowledge workers process tasks, their work is halted because other duties in between need their attention. During the execution of a new task, other interruptions may occur, so that the actual task is lost within their mental stack.

It would be helpful to have a tool which supports the user in addressing these issues. It should be able to provide help depending on the current context or worksphere, which may substantially differ from the help needed within other contexts. Further such an assistant should easily provide context-sensitive aid, reducing the required interaction to a minimum. Finally it should reduce the user's mental workload while he switches among contexts and help him remember previously done actions and tasks.

The terms "help" or "assistant" in our scenario are about speeding up the user's work. As the user will be able to perform his work without our application we provide no functional benefit out of his perspective. However the non-functional requirement we design our system for is that the users may complete their tasks with less interaction and less time. If, for example, a frequently observed operation takes a number of key- and mouse-strokes to perform and a certain time, we want to be able to present a shortcut with lower costs for reproducing this operation.

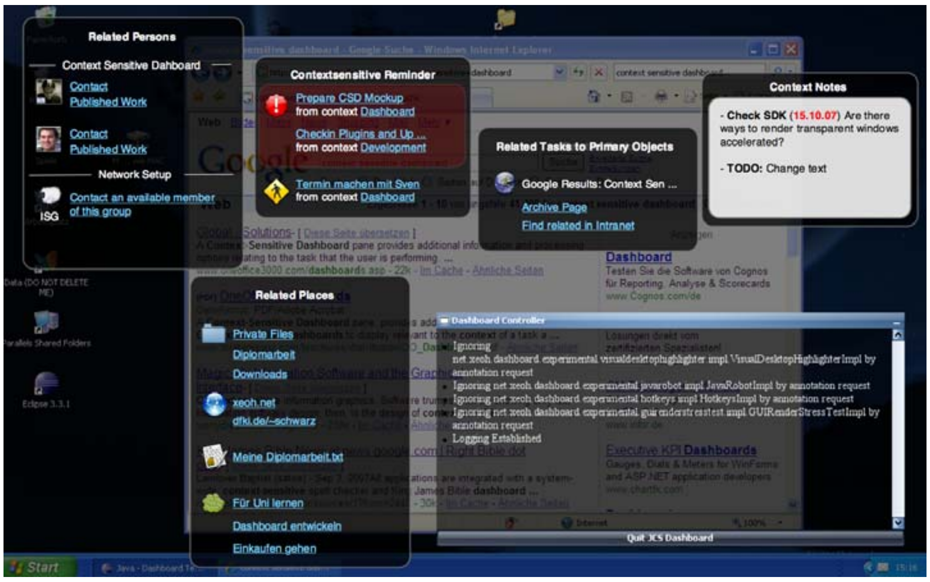
Our principal idea is to accelerate access to contextual elements (CE) occurring as entities in observed operations. These elements include files, documents, bookmarks, persons, notes, and reminders. Also the user's cognitive representation of the current context is taken into account, i.e., we want to help him to activate his memory regarding the status of the selected task more quickly. We think the methods presented can especially be of use for further multi-tasking, multi-purpose operating systems which extend the analogy *a context corresponds a task corresponds a desktop*.

Our idea is based on the design related to Apple's Dashboard[1]: A context-sensitive *overlay* is started in the background. When pressing a hotkey, a fullscreen window will be put topmost, while its semi-transparency assures the user is still able to see the original content of the screen.

The advantage of the this approach is the reduction of wasted screen estate [4]. A permanently visible window would also induce a permanent penalty regarding the space available to other applications. In contrast a dashboard takes no screen space while inactive, and, at the same time, allows the presentation of information using the whole screen. Further the transparency allows the user to see through to the original content shown before activation, which enables him to keep a mental reference to his work, as can be seen in Figure 1. This means assisting him in his current context, so that he is not forced to *loose* himself in the assistance itself.

The rest of this document is structured as follows: Section 3 explains the fundamental concepts in more detail, the way we define concepts mathematically and how help can be generated. Section 4 presents a technical overview of our

[1] Apple's Dashboard: http://www.apple.com/de/pro/photo/dashboard.html



- Check SDK (15.10.07) Are there ways to render transparent windows accelerated?

- TODO: Change text

```
• Ignoring
  net.xeoh.dashboard.experimental.visualdesktophighlighter.impl.VisualDesktopHighlighterImpl by
  annotation request
• Ignoring net.xeoh.dashboard.experimental.javarobot.impl.JavaRobotImpl by annotation request
• Ignoring net.xeoh.dashboard.experimental.hotkeys.impl.HotkeysImpl by annotation request
• Ignoring net.xeoh.dashboard.experimental.guirenderstresstest.impl.GUIRenderStressTestImpl by
  annotation request
• Logging Established
```

Fig. 1. A transparent overlay allows quick access to important contextual elements Please note that the screenshot is a partial mockup. Some of the widgets are currently being developed and not yet connected to the observation system.

architecture, subsystems, and widgets. This is followed by Section 5, a short discussion on evaluation issues we anticipate to encounter. Introductionary stories at the beginning of each section sketch the concepts from a user's point of view.

2 Related Work

A very similar idea of a dashboard like ours is used in the project *The Dashboard*[2]. They define context as a state derived from the currently focussed applications and their content, and it is used to allow the system to continuously search for related information in its backends. Those backends include documents, the personal mail database as well as various web services.

Mark et al. [1] studied the extent to which worksphere fragmentation occurs during office work. Their definition of work fragmentation *as a break in continuous work activity* is the kind of interruption we want to assist the user in overcoming with our approach. Also [2] points out the extent to which work activities are nestedly interrupted.

Stuff I've Seen [5] provides search centered access to previously seen items of the past. Documents are indexed and can be retrieved by keywords. The search results are embedded in their temporal context, global landmark events and personal events can be used as cognitive retrieval cues to aid the selection of a

[2] The Dashboard: http://www.nat.org/dashboard/

right document. However, no explicit context is modeled and also no automatic recommendations for the current work are given.

The idea to organize entities by their time for an more intuitive access is not new. Already in 1996 the Lifestreams project [6] implemented a sort of visual history to show previously used documents and allows a quick overview about the temporal context a document was used in.

Malibu [7] provides an always-on widget to aggregate resources in order to improve the access speed. Tasks, activities, bookmarks, feeds and people can be displayed. However, no visually assisted backtrack is provided.

During the MyMory[3] project, conducted at the DFKI, an application was created to employ the commonly used multi-desktop paradigm and extend it towards a multi-context working and switching environment. There one desktop is a technical representative and platform supporting one user's context (among many contexts). The interface allows quick creation of new desktops as well as easy switching between virtual desktops, which in turn leads to new or changed contexts. With such a context switching framework windows and parts of their content can be restored along with their previous screen positions, thus a first step to reduce the users mental load is performed.

3 Our Approach on Helping Users: From Native Operations to User Assistance

> Around 11:30 a.m. Joe processes Task A: "Paper-Writing". The system observes a sequence of so called native operations "open paper.tex", "edit paper.tex", "print paper.pdf", "send_mail joe boss". Shortly before noon Joe interrupts his work and switches the context to "Student Supervision". Because the system has previously observed this context a number of times, it can offer a quickly accessible *mail-to* button and shortcuts to documents related to the student. After he restores the desktop to the "Paper-Writing" state the system also provides him with the last activities he has done before leaving.

3.1 Principal structure of our workspace scenario

This section further describes the basic idea of our framework, how our application is embedded into the system, and how assistance is generated. Our goal is to support knowledge workers. Thus the principal entities we consider are shown in Figure 2. We assume our users work on one task at a time, but do however change them frequently.

There is no explicit task model in our domain. We rather think of them as hypothetical constructs which manifest themselves through native operations (NOPs). These native operations are recordable by system- and application-sensors and include, amongst others, file- and browser-operations such as *open*

[3] MyMory: http://www.dfki.uni-kl.de/mymory/

URL, communication-operations as *contact person* or cognitive operations such as *read passage of document* where the corresponding objects are substituted by concrete instances at runtime. An observable NOP therefore consists of an operation and a variable number of entities to which the operation is related. In contrast to observable NOPs, there are also unobservable interactions, for which no sensors are available yet. Further research needs to be done to decide which NOP-sensors are required for an appropriate assistance.

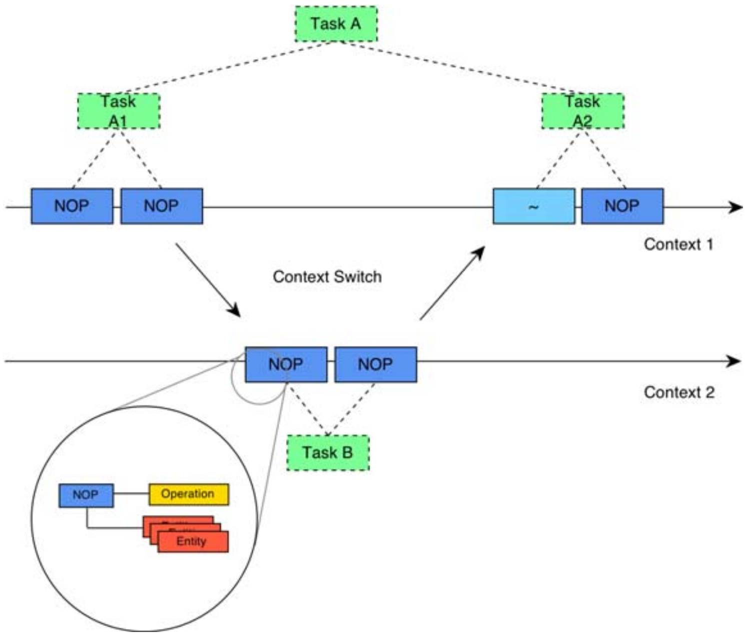


Fig. 2. Overview of the dependencies between contexts, tasks, native operations (NOPs) and entities. The $\sim$-mark indicates unobservable interaction *noise*.

The contexts in our domain are nominal categories of the workplace. Our system can be explicitly set by the user into a context in which it remains until told otherwise. This is the basic functionality which will be gradually enhanced by an automatic context detection and switch proposal based on learned context data from the MyMory project. It was shown that persons' definition of what should be contained in a task or context differs [8], so we do not define any a-priori context property except the ones just given and leave it, during the training phase, to the user to decide when a context switch is appropriate and sensible.

After this informal description of the concepts used we now want to describe our ideas in a more systematic way: Let s be a system of observation and T be the time domain. For simplicity reasons assume T is countable and totally ordered, the number of millisecond elapsed since a defined start date would be an example. Let E' be the countable number of concrete entities, best imagine this as the set of all possible URIs, and V an unlimited number of variables. Let $E = E' \cup V$ be the number of all entities. $C \subset E$ defines a countable number of contexts, this set also includes $undefined \in C$, an unknown state. Let $context_s : T \to C$ be a function which returns the context of the system at a given time, and let $O \subset E$ be a set of possible operations.

Let $N \subset E$ be the set of all possible NOPs. It is important to note that the individual $n \in N$ may occur several times at different $t \in T$. This is expressed by the function $occurrence : N \to 2^T$ which returns the set of all times the corresponding NOP has been observed. It is safe to assume that the resolution of T is in such a way, that at a given time $t \in T$ no more than one NOP appears. For convenience we define the inverse as $occurrence^{-1} : T \to N$, a function returning the corresponding NOP to a given time, and we will refer to it simply as *occurrence*, too. The functions $operation : N \to O$ and $entities : N \to E^{\mathbb{N}}$ define a NOP's operation and entities. If $n \in N$, $operation(n) \in E'$ and $entities(n) \subset E'$ we say n it is a concrete NOP, otherwise an abstract NOP. Further, uninterrupted NOPs within one context are referred to as a *context slice*. For example the NOPs of Task A1, A2 and B of Figure 2 form a context slice each.

3.2 Providing Context-Sensitive Assistance Through Previous Observations

Using the previously recorded NOPs we are now able to give selection shortcuts. But before we begin we introduce the term *current local context* (CLC). Due to the ordering T implies, with the help of *occurrence* on the elements N, we can say an (application) context $c \in C$ manifests itself retrospectively as an ordered observation sequence of NOPs, i.e., $c \cong (n^{(t_1)}, n^{(t_2)}, ...n^{(t_{latest})})$, where the superscript variables denote the time. As the CLC of a group of entities $g = (n^{(g_1)}, ..., n^{(g_{max})})$ we specify a window $w = (n^{(w_1)}, ..., n^{(w_{max})}) \subseteq c$ of temporally connected elements of one context slice, so that $w_1 \leqslant g_1$ and $w_{max} \geqslant g_{max}$. Let W be the set of all windows. In other words, a window of a group is a continuous part of a context slice that covers at least all elements of that group; $|w|$ denotes the window size.

The idea using the window based approach is related to the case-based reasoning (CBR) principle that "similar problems have similar solutions" [9]. The combination of CBR and context-sensitivity is not new [10], and in our scenario a context trace c serves as a continuously updated case base, where every possible window equals a case. A window surrounding the last observed NOPs g resembles the current case, and a similarity measure $sim : W \times W \to [0; 1]$ serves as a tool to compare how related two windows are. Finally adaptions of previously observed windows relating the current one produce hints of what entities to provide next.

Because our frames have different sizes we have to choose a measure respecting this. One of the simplest ones would be to normalize the NOPs in a sensible

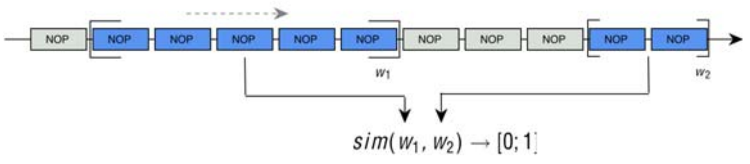


Fig. 3. Principal similarity calculation. The last NOPs of the current context (items right) are compared to a sliding window of the context's pasts. The window with the highest similarity is chosen as a foundation for further adaption.

way and calculate the coverage of the observation frame in the sliding window frame. The following formula

$$sim_{simple}(w_{current}, w_{sliding}) := \frac{|w^*_{current} \bigcap w^*_{sliding}|}{|w_{sliding}|} \quad (1)$$

formalizes this idea, where w^* represents the normalized set of w. Normalization can also be done in several ways and describes a means to convert concrete NOPs to a more general representation. As a NOP consists of an action part and an entities part, we can distinguish, for example, the methods 'None' which keeps the NOP as it is, 'Entity Extraction' (ER) that extracts a single entity of a NOP thus ignoring the operation, and 'Entity Extraction and Reduction' (ERR), a concatenation of ER and some entity reduction scheme.

The 'None' normalization requires a perfect match of previously observed situations, we expect this mode to deliver the highest precision. The ER-mode does not take into account what was previously done with observed entities and calculates similarities based solely on the objects themselves. ERR-mode also ignores details of the exact location or nature of objects and adds a certain degree of unsharpness. For example, given the NOPs $n_1 = (open, http : //www.spiegel.de/politik/ausland/542470.html)$ and $n_2 = (print, http : //www.spiegel.de/politik/ausland/542365.html)$, they would be different according to the 'None' and ER function, but similar according to an ERR function ignoring the filename. The ERR function is likely to give a better recall, which should be increasing with its permissiveness.

Because we want to provide shortcuts based on previous observations, we need a way to select appropriate candidates. Any *sim* function induces an total order on all windows and thus also multiple ratings on the enclosed NOPs. A *rating strategy* decides how the similarity of different windows impacts the overall rating of a NOP for assistance purposes. Let $rating_{sim,w} := N \rightarrow \mathbb{R}^*$ be a rating agglomeration function for a given NOP based on a similarity measure and a query window. For our purposes we define the strategy function to be

$$strategy(n) := \max(rating_{sim,w}(n)) \quad (2)$$

that simply selects the maximal element of the set of all possible ratings. Using *strategy* we eventually have an indicator for the relevance of a NOP. The combination of Equation 1 and 2 means: Compare the current scenario with similar scenarios in the past that occurred in the current context line. See which entities (e.g. files, folder, URLs) happened to be accessed at the same time. If, during equivalent situations in the past, entities were relevant it is arguable the same entities might be relevant now again.

After we introduced our principal concepts, depicted a notion of context and explained a way of how to generate information for assistance, the next section will describes the technical details of how the assistance items will be displayed and how the services are being integrated.

4 The Dashboard Architecture

> Interaction with the dashboard is fairly simple. After Joe decides to switch the context to “Student Supervision” he might want to activate the dashboard by pressing *F12*. Instantly a semi-transparent overlay appears to display some previously configured widgets. Joe looks at the “Bookmarks Widget” for the current context. Among the top entries a link to the student’s blog is displayed, as it was the most frequently accessed URL of that context. This behavior is based on the assumption, that frequently observed NOPs will very likely be observed again, and therefore the cost of generating them should be reduced. Further, if Joe returns from lunch later on and switches back to “Paper-Writing” he might activate the dashboard, too. Then the ’Visual Backtracking Widget’ would show a short summary of the events that happened during the context’s last *context slice*, including a visual aid of past activities in form of screenshots and text entered or entities accessed.

Figure 4 shows how the dashboard extends the operating system. Directly connected to the operating system are some low level services for keyboard listening and application observation. A screenshot recorder takes snapshots every minute. Based on the low-level operating system services we build some higher level facilities. An input summarizer filters human readable text from the raw keyboard input, stripping special characters and searching for most frequently used words. The context services store information about observed NOPs and allows their time slice based retrieval. A statistics recorder can be used to calculate usage numbers for NOPs and entities, which the widgets uses to display probably relevant information and accept input. Eventually the system interface service will be invoked to execute a command such as opening a file. The NOPs are recorded through the User Observation Hub (UOH) to which the dashboard connects to. It collects information from various sources and application-plugins. Amongst them are Firefox- and Thunderbird-plugins and a connector to the User Activity Logger[4].

[4] User Activity Logger: http://pas.kbs.uni-hannover.de/

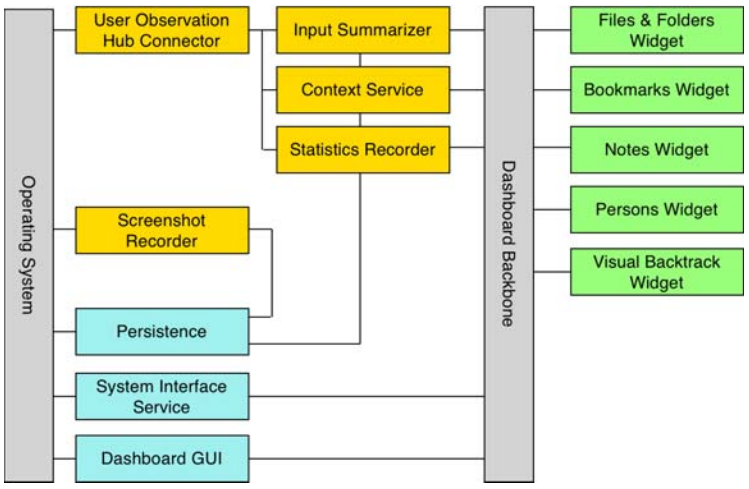


Fig. 4. Dashboard Architecture. The services monitor the operating system for user actions and access basic technical facilites. The dashboard backbone connects the services amongst each other and also allows the widgets to lookup requested modules.

4.1 Context-sensitive Widgets Deliver Context Related Material

The main interface are the widgets displaying current context information. The decision which widgets for which entities should be chosen is fairly open, but also dictated by the observable NOPs and their connected entity types.

Currently we have the following panels in mind: A people widget to show persons important to the current task as well as buttons to contact them. A file and folder widget with quick links to frequently accessed elements. A bookmark widget for context-sensitive frequently accessed URLs, context-sensitive notes to store free text information regarding the current task and a visualization of usage statistics.

In addition we have a visual backtracking widget that allows a peek into the past regarding the current context. We implemented this functionality through capturing screenshots every 60 seconds and logging all keyboard input. At the highest temporal resolution this information may serve as a remembering aid to replay what the user has performed. It may as well be transformed into a denser summary of past events, if looked at from some more distant time.

The visual backtracking widget also includes a current local context summary. While the more general widgets provide access to the most important elements for the whole context-time, the visual backtrack separates the context into time-slices of current context activity. To each time slice the most important elements are shown directly besides them. The advantage herein is the access given to the user regarding items he used while working on the corresponding slice.

All these widgets listen to a context service and also to the usage statistics facility, that monitors additional events to the keystrokes mentioned before. The context changes sent by this source are used to load and compute the appropriate assistance of the corresponding widgets, such as the most relevant persons in the past of a context that was just switched to.

5 Evaluation Outline: Method and Issues

> A glance at the visual backtracking widget helped Joe remembering some of the previously opened documents and also part of his remaining work activities. By using the provided link to one of his files, he was able to access it in approximately two seconds requiring three mouse- and keystrokes. Without the dashboard he had to employ other means to remind himself of previous activities and had to locate the items manually. If he knew the exact location of the file, this might have taken five to ten seconds and required at least five double clicks assuming a nested item depth of five on a Windows system.

According to ISO 9241-11 quantitative software usability can be divided into the sections *efficiency*, *effectiveness*, and *satisfaction*; this paper covers the first two. Even though raw performance improvements do not always [11] induce a higher likelihood of users actually using a software, the main concern of the dashboard is to effectively speed up a users work. Satisfaction-ratings are equally relevant, but they are not our main worry: We assume that after efficiency and effectiveness have been established, it is a matter of implementation and design that mostly affects this variable.

To generate utilizable propositions about the dashboard regarding its performance these statements have to be comparable to other systems in order to show one of them *works better* or is an improvement regarding some criteria.

This yields two questions: "better than what other system?" and "better in which way?". A third problem is how such an advantage could be measured.

In our view any controlled laboratory experiment will not be suitable for this task, as it is an artificially constructed environment wherein long term studies simply are not reasonably performable. Also such a laboratory study is likely to give an advantage to our application, as a scenario will effectively be modeled with context switching tasks in mind, where it is currently not clear if and how such an assistance is accepted by users in reality.

This is closely related to questions regarding the alternative system under observation. A common approach is to split the subject group into one part using the context-sensitive dashboard and another part using no explicit support system. While this method surely gives basic tendencies of which system is preferred, we are not exactly sure what more general statement is supported by this observation: In fact, one concrete multi-context dashboard implementation with its own widgets, symbols, and layout is compared to an abundance of alternative work concepts performed.

The factors under observation are more easily to determine. As the goal of our system is 'increased work throughput' in the means of less required interaction for task execution, and a decreased cognitive load, these should be the compared variables. As pointed out previously, we believe the quantitative usage and speedup costs can rather easily be measured. Interaction costs with the system can be expressed as the number of clicks, key presses and mouse meters performed while the dashboard was active. The overall time spent in the dashboard also equals the time the user *lost*. Those numbers must be compared against the benefits gained by the usage. If the time spent was less than the time saved, we can prove the statement that the dashboard was at least not obstructive.

Which leads to the next question: How can the benefits be measured? If the dashboard was opened, no exits in forms on provided elements were taken and no other visual backtracking interaction was performed, it might be reasonable for our scenario to assume, that the user had no benefit of the operation.

Gathering the data automatically during daily usage is advantageous, but it is not always clear if and when the dashboard could help the user. Without any user feedback, the system can not find out whether the data presented was not useful at all; the data presented helped the user to get a good overview of some matter; or even the data presented did not help the user for what he really wanted to do, but reminded him of another important matter.
If the dashboard is opened, no action in forms on provided elements are taken and no other visual backtracking interaction is performed, it might be reasonable for our scenario to assume the user has no benefit of the operation.

Using the individual data gathered on each workspace we can calculate general usage statistics and give an overview which features have been used how many times and what could be an estimate for the time saved using the context-sensitive dashboard.

Furthermore we have to take negative costs of the application into account: Every activation and the time spent in the interface reduces the benefit produced by usage of a shortcut. In the worst case the user spends more time activating and unsuccessfully searching than he actually gains by the shortcuts he uses.

6 Outlook and Further Work

The system described in this paper is currently under development. The dashboard backbone, the user observation hub, the recorder, the system services and some of the widgets are currently operational, but not yet provided with real data. Our next steps will be to integrate all components, complete the remaining widgets, test and deploy the application.

After that, the outlined usage statistics have to be collected to get an overview of how the users respond to and interact with the application.

Another open research question we want to address, is if we really need our explicit context representation and switch assistant when we employ the presented case-based reasoning approach. If we gathered enough usage data in form of NOPs we intend to do an offline evaluation comparing the precision / recall of an explicit context model versus a solely case-based approach.

Furthermore it is planned to open source the dashboard as well as its basic infrastructure in the near future. In standalone mode the general dashboard architecture can be used to develop new plugins and integrate with the hotkey and backtracking service, if used in conjunction with the MyMory infrastructure, also the other services will be usable, such as the context notification infrastructure, recording service of the user observation hubs messages, and the hereon based CE extraction. An integration into other context provider systems should also be possible with reasonable effort.

References

1. Mark, G., Gonzalez, V.M., Harris, J.: No task left behind?: examining the nature of fragmented work. In: CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems, New York, NY, USA, ACM Press (2005) 321–330
2. Smith, G., Baudisch, P., Robertson, G., Czerwinski, M., Meyers, B., Andrews, D.: Groupbar: The taskbar evolved. In: Proc. OZCHI 2003, University of Queensland (2003) 34–43
3. Rubinstein, J.S., Meyer, D.E., Evans, J.E.: Executive control of cognitive processes in task switching. In: Human Perception and Performance (2001)
4. Mullet, K., Sano, D.: Designing visual interfaces: Communication oriented techniques. SunSoft Press, Upper Saddle River, NJ 07458, USA (1995)
5. Dumais, S., Cutrell, E., Cadiz, J.J., Jancke, G., Sarin, R., Robbins, D.C.: Stuff i've seen: a system for personal information retrieval and re-use. In: SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval, ACM Press (2003) 72–79
6. Fertig, S., Freeman, E., Gelernter, D.: Lifestreams: An alternative to the desktop metaphor. In: SIGCHI Conference on Human Factors in Computing Systems Conference Companion (CHI '96), New York, ACM Press (1996) 410–411
7. Geyer, W., Brownholtz, B., Muller, M., Dugan, C., Wilcox, E., Millen, D.R.: Malibu personal productivity assistant. In: CHI '07: CHI '07 extended abstracts on Human factors in computing systems, New York, NY, USA, ACM (2007) 2375–2380
8. Czerwinski, M., Horvitz, E., Wilhite, S.: A diary study of task switching and interruptions. In: CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems, New York, NY, USA, ACM Press (2004) 175–182
9. Aamodt, A., Plaza, E.: Case-Based Reasoning: Foundational issues, methodological variations, and system approaches. AI Communications **7**(1) (1994) 39–59
10. Kofod-Petersen, A.: A Case-Based Approach to Realising Ambient Intelligence among Agents. PhD thesis, Norwegian University of Science an Technology, Trondheim (2007)
11. Nielsen, J., Levy, J.: Measuring usability: Preference vs. performance. Communications of the ACM **37**(4) (1994) 66–75