informationresearch

VOL. 14 NO. 1, MARCH, 2009

Evaluating shared workspace performance using human information processing models

Antonio Ferreira and Pedro Antunes
Department of Informatics, University of Lisbon, Campo Grande, 1749-016 Lisbon, Portugal
José A. Pino
Department of Computer Science, Avenida Blanco Encalada 2120, Tercer Piso, Santiago, CP 837-0459, Chile

Abstract

Introduction. Shared workspace evaluation can be expensive and time consuming. It is also usually oriented towards high-level qualitative perspectives of the collaboration among users. A quantitative method is presented, giving emphasis to the low-level details of critical scenarios of shared workspace interaction, and allowing for comparisons of predicted execution time.
Method. Models of human information processing are used to approximate human behaviour while working through a shared workspace. Generic groupware input/output devices and information flows are categorised.
Analysis. Three cases of shared workspace activity are analysed. For each case, two or more design scenarios are evaluated and their performance compared.
Results. The method contributes to formative evaluation regarding the manipulation of coupling mechanisms and the timing and availability of group awareness information, and it offers indications about the potential performance of users working with shared workspaces.
Conclusions. The proposed method is aligned with the century-old need to measure before improving. It is aimed at providing the groupware designer with a tool to make quick calculations, enabling several design iterations, without requiring users or functional prototypes.

Introduction

Shared workspaces are becoming ubiquitous groupware tools, allowing co-workers and interest groups to share information and organize their activities in very flexible and dynamic ways, usually relying on simple graphical metaphors. An important feature of shared workspaces is that they conceal much technical functionality from users. This functionality is needed to manage data distribution,

synchronization and replication, security, persistence, access management, connected and disconnected modes, and other aspects.

This concealment challenges groupware designers in multiple ways: on the one hand, they have to design graphical metaphors that are at the same time compelling, innovative, user-friendly and useful; on the other hand, shared workspaces must adequately mediate the interaction between the users' mental models and the underlying groupware functionality; and finally, there is always the problem of optimizing the performance of shared workspace usage. Assuming that shared workspaces may support many users working through the computer, such an optimization may produce important usability gains to users and it may, as well, become an important factor to the success of groupware technology.

In this paper, we address the final design problem: optimizing shared workspace performance. This is a challenging endeavour for practitioners and researchers because existing methods have considerable trade-offs and impose significant constraints:

- Measuring performance is expensive in terms of resource consumption (time, users, experts, apparatus). This applies especially, but not exclusively, to controlled laboratory experiments and it is inherent to the specific constraints of the collaborative context. In most common situations, the development projects do not afford having multiple laboratory experiments with different shared workspace designs, which, by themselves, are more complex to develop, just to find out that the results are equivocal because of the complexity of the collaborative setting.
- Of course, several researchers have recognized the difficulties with performing laboratory experiments in the collaborative context and have developed *discount* methods. These methods avoid measurements in the laboratory and are focused on identifying qualitative issues and contrasting them against prescriptive measures. Therefore, because of their problem orientation, these methods provide little support for comparing design options and, especially, for measuring the performance of shared workspaces.

We argue that, in spite of these constraints (further detailed in Related work, below), groupware designers should be able to make quick measures and calculations about shared workspace performance. Our motivation is based upon the century-old need to measure before improving, as well as on the evidence that fast evaluation enables several design iterations.

In this paper, we present a method to quantitatively *predict* and *compare* the performance of shared workspaces. We define a shared workspace as a computer mediated workspace with a shared data model, visualization and control policy. The method is not based on functional prototypes, thus avoiding laboratory experiments and it is not based on a discount approach, thus also avoiding qualitative evaluations. The alternative method we adopt is to analytically model the functionality of the shared workspace and, from that model, apply known human information processing models to measure the shared workspace performance and to draw conclusions about design options.

This alternative approach has already been used in other contexts, namely in the human-computer interaction field, ever since the early 1980s. It actually predates many of the discount methods, particularly those for groupware evaluation, which were developed partly to counter the difficulties of measuring computer-mediated group work. The *keystroke-level model* (Card *et al*. 1980) is one example of a quantitative method that has received much attention from human-computer interaction researchers. This model represents the user as having perceptual, motor and cognitive processors, each with its own performance parameters, which approximate single-user interaction at a low level of detail. In this paper, we discuss the framing of this model in light of the specific characteristics of groupware, so that the performance estimates posited by the model may help predict shared workspace performance.

The paper is organized as follows. The Related work and Theoretical background sections provide the antecedents for this research. The Method to evaluate shared workspace performance section describes the proposed method for evaluating the performance of users working together in a shared workspace. Three cases of workspace activity, each one with several alternative designs, are evaluated in Using the

method section to demonstrate the value of this method. The Discussion and implications for design and Conclusions sections end the paper with a discussion on the benefits and limitations of the method, as well as providing some implications for design.

Related work

Groupware evaluation and, in particular, shared workspace evaluation, is difficult to perform because the trade-offs inherent to different evaluation methods are further constrained by the complex multidisciplinary nature of groupware systems. Traditionally accepted methods of evaluation, such as laboratory experiments (Fjermestad and Hiltz 1999) and field studies (Hughes *et al*. 1994), are becoming increasingly unmanageable because they involve multiple people, who can be hard to find with the required skills, may be geographically distributed, or simply unavailable for the considerable time necessary to accomplish collaborative tasks. Furthermore, the evaluation process requires significant time and expertise to prepare and execute, while the time to design prototypes runs out. These limitations led to the recent emergence of a collection of discount methods (most of them derived from *singleware* methods) with the purpose of reducing the complexity and cost of groupware evaluation.

Groupware task analysis (van der Veer and Welie 2000) is a method that combines high-level hierarchical task analysis and field observations for addressing all stages of groupware design. It is based upon a conceptual framework that includes agents, group work and situations, in a manner similar to the work models defined by the *contextual design* approach (Beyer and Holtzblatt 1998), well known in the human-computer interaction area.

The next three discount methods for groupware evaluation are based upon a common descriptive framework called *mechanics of collaboration* (Pinelle *et al*. 2003), whereas each method applies its own evaluation perspective. The mechanics are formalizations of high-level group work primitives (e.g., communicating and coordinating), which help the designer focus on how the shared workspace supports the required collaboration. Starting with *collaboration usability analysis* (Pinelle *et al*. 2003), this method couples field observations and a version of hierarchical task analysis that allows variation, iteration and parallel work, for representing group work. The *groupware walkthrough* method (Pinelle and Gutwin 2002) uses step-by-step written narratives or task diagrams of collaboration scenarios and aims at gathering the opinions of expert inspectors while they use the workspace. Finally, *groupware heuristic evaluation* (Baker *et al*. 2002) is based upon a number of experts evaluating the compliance of a shared workspace with a list of heuristics.

The previous discount methods provide various views on groupware evaluation, but they all lack the capability to quantitatively predict human performance, thus hindering the comparison and measurement of prototype designs. These shortcomings are addressed by the application of models of human information processing, to simulate human behaviour while executing tasks with a computer.

One of the most successful models of human information processing is provided by the *goals, operators, methods and selection rules* (GOMS) method (Card *et al*. 1983). Its validity is asserted by a family of task analysis and modelling techniques (including the keystroke-level model) and by a significant number of studies on diverse applications, such as call centres, information browsers, industrial schedulers, text editors and others (John and Kieras 1996). Although traditional GOMS evaluations focus on an individual user working with *singleware*, as it was originally developed, recent research shows that it is possible to model multiple users interacting with groupware.

Distributed GOMS or DGOMS (Min *et al*. 1999) applies hierarchical task analysis and human information processing models to represent group activity and to predict execution time, distribution of workload and other performance variables. This method successively decomposes group work in group tasks until individual subtasks can be identified. At this level of detail the subtasks are defined in terms of perceptual, motor and cognitive operators, as well as with a new communication operator that is used to coordinate individual tasks executed in parallel. The limitation, however, is that such a coordination

mechanism is more appropriate to groups where users react to predefined events and not sufficiently rich to describe the type of interdependency established by users working through shared workspaces (Malone and Crowston 1994).

Another application of human information processing models to groupware considers *teams of models* to analyse a complex task executed by a group of users (Kieras and Santoro 2004). The task involved several users with individual roles monitoring a display and executing actions in a coordinated way, by means of a shared radio communication channel. This approach assumes that several individual models are necessary to explain collaborative work, but the study does not address workspace collaboration and instead focuses on coordinated work.

In summary, existing applications of human information processing models to the groupware context are targeted at predicting performance in coordinated work scenarios where users react to predefined events, requiring neither shared workspaces nor group awareness. The method we describe in this paper complements current research by predicting performance in scenarios of collaboration through shared workspaces.

Theoretical background

In general, human information processing models have been associated with the *model human processor* (Card *et al*. 1983), which represents human information processing capabilities using perceptual, motor and cognitive processors. Nevertheless, several differences can be identified when considering higher-level models that were built up from the model human processor: for instance, the keystroke-level model uses a serial-stage processing model, whereas *cognitive, perceptual and motor GOMS* (CPM-GOMS) addresses multi-modal and parallel human activities (e.g., recognizing an object on the display while moving the hand to the keyboard) (John and Kieras 1996). In spite of these differences, a common characteristic of existing human information processing models is that they are *singleware*: they assume that *one* user interacts with the computer interface. Figure 1 is a representation of the model human processor and also illustrates that there are conventional information flows from the user's cognitive processor to the motor processors, from the input to the output devices of the computer interface (e.g., the keyboard and the display) and back to the user's perceptual and cognitive processors.

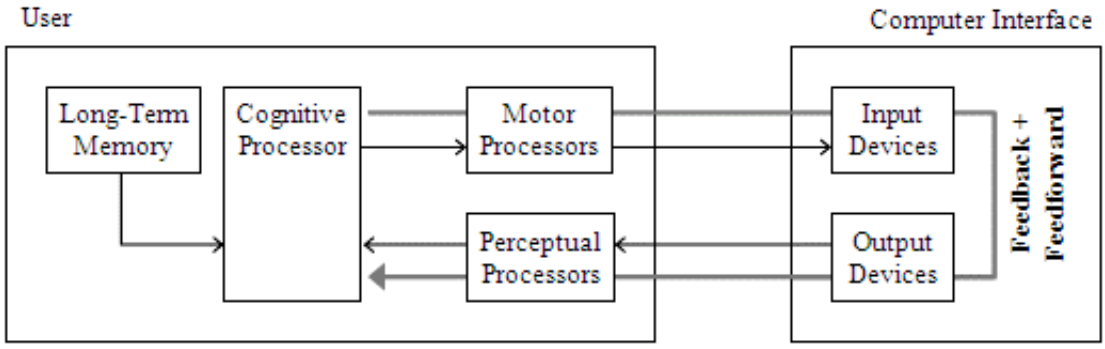

Figure 1: *singleware* information flows

According to some authors, the information processors and flows depicted in Figure 1 apply directly to groupware (Kieras and Santoro 2004). To model a group of users, one can have individual models of the interaction between each user and the computer interface; one can also assume that the interface is shared by multiple users and that the users will deploy procedures and strategies to communicate and coordinate their individual actions. Thus, according to this view, groupware usage is reflected in some conventional information flows, spanning multiple users.

The problem, however, is that this view does not consider two fundamental groupware features: first, the conventional information flows are considerably changed to reflect collaboration, mutual awareness and interdependence; and second, the focus should not remain on the interactions between the user and the

computer interface but should significantly change to reflect the interactions between users, mediated by the groupware interface. We deal with these two issues in the next section.

Groupware conventional information flows

Let us start with an explanation of the *singleware* conventional information flows in Figure 1: the first flow corresponds to information initiated by the user, for which the computer interface conveys *feedback* information to make the user aware of the executed operations (Douglas and Kirkpatrick 1999, Wensveen *et al.* 2004); the second flow concerns the delivery of *feed-forward* information, initiated by the computer interface, to make the user aware of the available action possibilities (Wensveen *et al.* 2004).

Now, when we regard groupware, some additional categories have to be considered. In this paper, we consider explicit communication, feed-through, and back-channel feedback.

Explicit communication addresses information produced by one user and explicitly intended to be received by other users (Pinelle *et al.* 2003). For example, a user may express a request for an object to another user. This situation can be modelled as a computer interface capable of multiplexing information from input devices to several output devices. The immediate impact on the model in Figure 1 is that we now have explicitly to consider additional users connected to the interface, as shown in Figure 2.

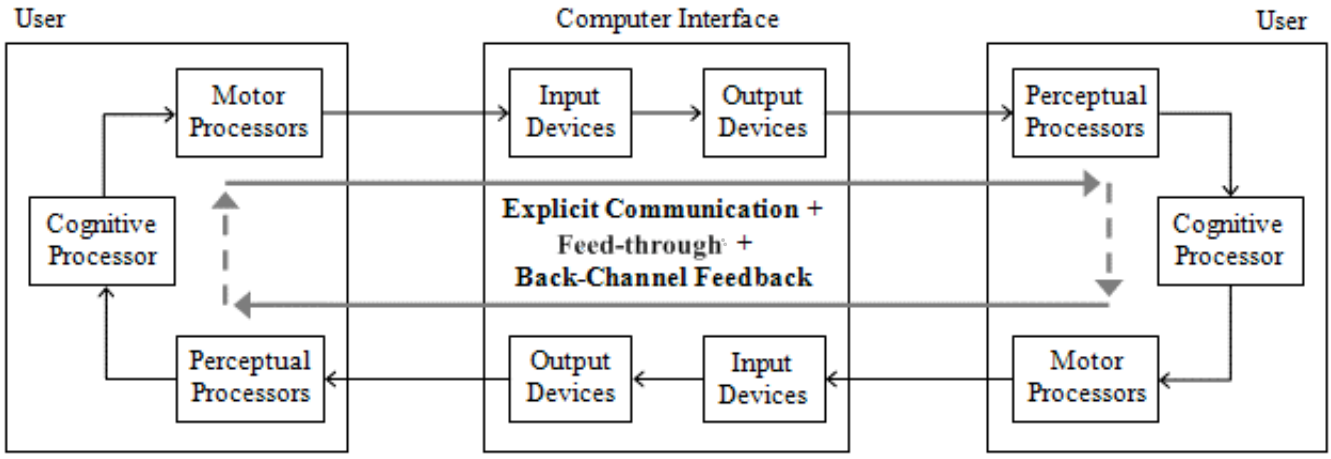

Figure 2: Groupware information flows

Feed-through concerns implicit information delivered to several users reporting actions executed by one user (Hill and Gutwin 2003). feed-through is essential to provide group awareness and to construct meaningful contexts for collaboration. For example, the shared workspace may show currently selected menus for each user who is manipulating objects. This information is automatically generated by the computer interface as a consequence of the user's inputs and is directed towards the other users. A very simple way to generate feed-through consists of multiplexing feedback information to several users. Sophisticated schemes may consider delivering less information by manipulating the granularity and timing associated with the operations executed by the groupware (Gutwin and Greenberg 1999).

Finally, *back-channel feedback* concerns unintentional information initiated by one user and directed towards another user to facilitate communication, indicating, in particular, that the *listener* is following the *speaker* (Rajan *et al.* 2001). No meaningful content is delivered through back-channel feedback, because it does not reflect cogitation of the user. Back-channel feedback may be automatically captured and produced by the computer interface based upon the users' body gestures and vocal activities.

Groupware specializations of the computer interface

All groupware information flows are naturally processed by the user's perceptual, motor and cognitive

processors and the corresponding computer input and output devices. However, we regard the separate processing of explicit communication, feed-through and back-channel feedback in specialized input and output devices to show the distinction between collaborative and non-collaborative interactions. We define the *awareness input and output devices* as devices specialized in processing information about who, what, when, how and where the other users are operating in the shared workspace.

Another specific feature of the awareness input and output devices is that they not only afford users to construct a perceptual image of the collaborative context, but they also allow users to perceive the role and limitations of the computer interface as a mediator. This is particularly relevant when the Internet is used to convey feed-through information, where feed-through delays are less predictable and significantly longer than feedback delays (Gutwin *et al*. 2004) and the available bandwidth and network availability may be limiting factors (Cosquer *et al*. 1996).

A further reason for proposing the awareness input and output devices is related to another particular characteristic of groupware: it lets users lose the link between executed operations and group awareness, a situation called *loosely coupled* (Dewan and Choudhary 1995). Two types of coupling control may be considered: first, users may control coupling at the origin to specify what and when private information should become public; second, coupling can be controlled at the destination to restrict the amount of awareness information, e.g., by specifying filters on objects and types of events. In all cases the user needs some cognitive activities to discriminate and control awareness information delivery and we model this situation with the *coupling input device*. We illustrate the resulting groupware interface in Figure 3.

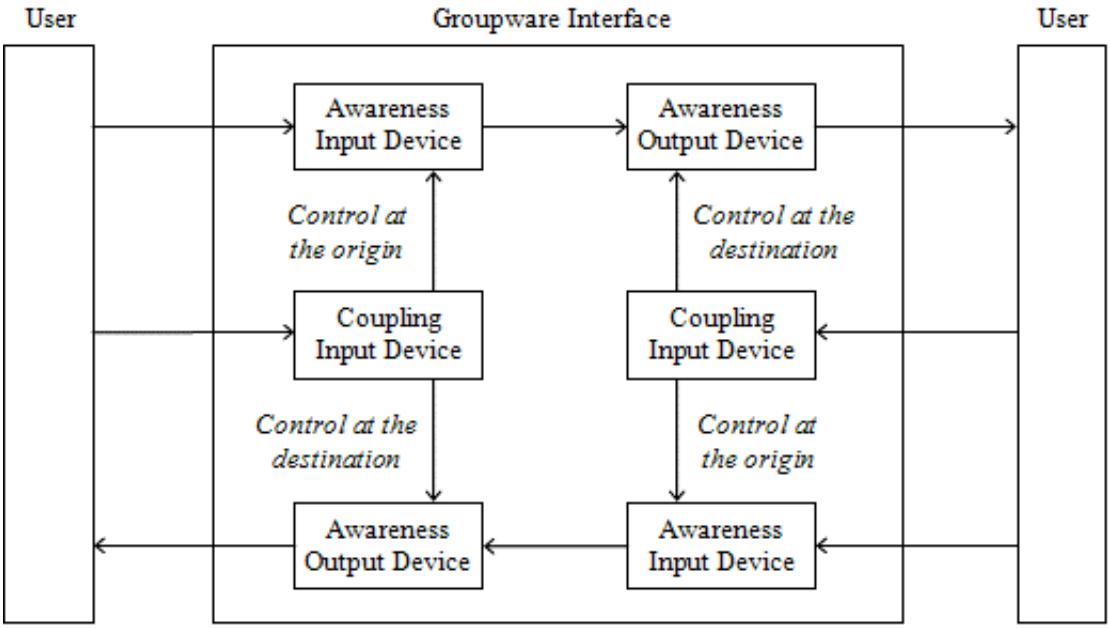

Figure 3: Groupware interface with specialized awareness and coupling devices

In summary, our interpretation of the *model human processor* takes the groupware context in consideration and essentially emphasizes the cognitive activities related to the awareness and coupling features supported by the groupware interface.

Method to evaluate shared workspace performance

Step 1: Groupware interface. The method starts by defining the generic elements of the groupware interface. We propose that the interface should be broken down into one or more shared workspaces. Such decomposition simplifies the modelling of complex groupware tools, which often organize collaborative activities in multiple intertwined spaces, usually humanly recognizable, supporting various purposes, objects and functionality.

Using the groupware interface in Figure 3 as a reference, we define a shared workspace as a distinctive

combination of awareness and coupling devices. We exclude from the groupware model any workspaces not having, at least, one awareness or coupling device, since they would not involve collaboration.

The outcome of this step is then: (1) a list of shared workspaces; (2) a definition of supported explicit communication, feed-through and back-channel feedback information flows; and (3) a characterization of supported coupling mechanisms. In this step, alternative design scenarios may also be defined, considering different combinations of shared workspaces, awareness information and coupling mechanisms.

Step 2: Critical scenarios. The second step describes the functionality associated with the shared workspaces defined in the previous step, with a special focus on critical scenarios. *Critical scenarios* are collaborative actions that have a potentially important effect on individual and group performance. The functionality may be decomposed into sub-actions, using a top-down strategy, but attention should be paid so that the descriptions remain generic. As in the previous step, alternative design scenarios may be defined, considering several combinations of users' actions.

Step 3: Boundary selection. The third step is a focusing step, where the (possibly infinite) configurations of each shared workspace, including its objects and users, are abstracted according to the designer's intuition, expertise and goals.

In this step, several characteristics of the shared workspaces may be controlled by assumptions concerning aspects such as: the position and size of graphical elements on the computer display, the mechanisms that provide awareness information; the coupling mechanisms of group work; the number of users in the group; the probabilities of user actions; the placement of objects in the workspace; and others that the designer may find relevant to workspace performance.

Step 4: Shared workspace performance. The final step is dedicated to comparing the alternative design scenarios that were defined in the previous steps. These comparisons require common criteria, for which we selected the *predicted execution time* in critical scenarios.

We use the keystroke-level model (Card *et al*. 1980, Card *et al*. 1983) to predict execution times because it is relatively simple to use and has been successfully applied to evaluate single-user designs (John and Kieras 1996). In this model, each user action is converted into a sequence of mental and motor operators (see Table 1), whose individual execution times have been empirically established and validated by psychological experiments (Kieras 2003, Olson and Olson 1990, Card *et al*. 1983). Therefore, the designer may find out which sequence of operators minimizes the execution time of a particular user action.

Operator	Time	Description
H	0.4	Home hand(s) on keyboard or other device
K	0.1	Press or release mouse button
M	1.2	Mentally prepare
P	1.1	Point with mouse to target on a display

Table 1: Keystroke-level model operators (Card *et al.* 1980) and predicted execution times, in seconds (Kieras 2003)

Naturally, the application of the keystroke-level model must be adapted to groupware, considering that the execution time we want to evaluate affects several users who work through shared workspaces. Our approach consists of focusing the evaluation on critical scenarios having selected sequences of operators concerning frequent manipulations of the shared workspaces, possibly involving more than one user at

the same time.

For instance, suppose we want to evaluate the performance of several design options for managing access to objects in a shared workspace. A critical scenario occurs when a user accesses the object, immediately followed by another user trying to access the object but finding it locked. We may use the keystroke-level model to estimate the execution times of these combined operations for each design option and thus finding out which one minimizes the overall execution time. This will be discussed in one of the cases presented in the next section.

Using the method

In this section, we apply the proposed method to evaluate the performance in three cases of shared workspace activity: locating updated objects, reserving objects, and negotiating requirements.

Locating updated objects

The first case considers a graphical shared workspace where several objects may be updated in parallel by a group of users. An object can be a text document, a drawing, or any other type of information that is relevant to the activity of the group. In collaborative scenarios such as this it is important that users are aware of the updates that are being applied to the objects, otherwise group performance may degrade because of, for example, wrong decisions based upon obsolete mental images of objects, or duplicate work due to the object being created elsewhere in the meanwhile.

In this case, users can play two roles: the first occurs when they update one or more objects; the second role is characterized by the need to be aware of and locate objects that have changed. We will assume that the first role has already been fulfilled (an object was recently updated) and so we will analyse the second one. The design challenge is that there are many ways to convey updated information from one user to others and some of these ways may be preferable.

We note that collaboration among users in this case is somewhat indirect, in the sense that we focus on information flowing from the shared workspace to individual users, although such flows are a consequence of updates made by other users. However, these information flows ease the construction of more sophisticated collaborative scenarios and, thus, their importance should be acknowledged.

Step 1: Groupware interface. The shared workspace is capable of storing a large number of objects. However, since computer displays have limited screen resolution, access to the objects is provided by a *viewport*, which shows only a small portion of the shared workspace. The *viewport* can be moved and so the whole shared workspace is effectively viewable, albeit at a cost, measured in extra execution time, which depends on the design options.

Our first design uses a list of object names on the right side of the screen to provide awareness to users on objects that have recently been updated (see top left side of Figure 4). Because the list takes up space, the viewport is smaller than the entire computer display, which lowers the probability of an object being shown at an arbitrary time on the viewport. This is design scenario A.

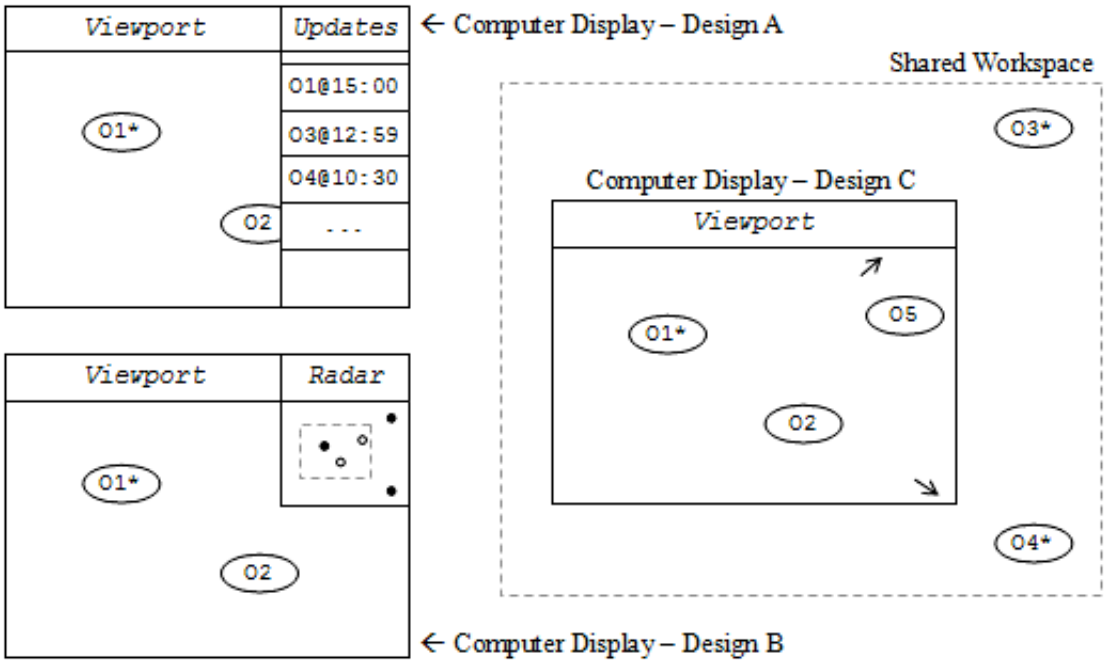

Figure 4: Design scenarios for locating updated objects in a graphical shared workspace

Design scenario B features the viewport and a miniature representation of the entire shared workspace, also called a radar view (Gutwin and Greenberg 1999). Whenever an object is updated a dark-filled dot replaces the normal hollow circle on the radar, thereby making the user aware of the update (see bottom left side of Figure 4). As in the previous design, the radar view takes up display space.

Finally, in design scenario C the entire computer display is devoted to the viewport. When objects are updated and if they are not already being shown on the computer display, then the border of the viewport is populated with awareness indicators that are little arrows pointing in the direction of the objects in the shared workspace (see right side of Figure 4).

We assume that human input is done using a mouse with a single button (for design scenarios A and B) and by the keyboard cursor keys (design scenario C).

At the end of this step, we can characterize the groupware interface in the following terms: (1) one shared workspace stores all objects; (2) awareness is provided in the form of feed-through information (no explicit communication or back-channel feedback is allowed); (3) awareness is supported by a viewport and by a list, or a radar view, or by pointing arrows, depending on the design scenario; (4) there is a loose coupling between the changes that are made to the objects and the awareness that is provided to the users (an update is simply represented by an asterisk); and (5) the viewport permits coupling control, showing some objects while omitting others.

Step 2: Critical scenarios. Regarding the critical scenario; i.e., how to locate an updated object, we now describe, for each of the three design scenarios, the actions that users have to perform.

In design scenario A, the user notices that an object (which is outside of the viewport) has been updated by looking at the list of recently updated objects. To locate it in the shared workspace s/he clicks the mouse button on the corresponding line in the list, causing the viewport to be positioned in such way that the object is shown on the computer display.

With design scenario B, a dark-filled dot appears on the radar view, the user points the mouse cursor and clicks the button somewhere in the vicinity of that dot to move the viewport to that location in the shared workspace, bringing the updated object into sight.

In design scenario C, a user can navigate through the shared workspace by pressing the keyboard cursor

keys. The appearance of a pointing arrow at the border of the viewport means that an object has been updated; s/he has to follow the arrow until the object appears on the computer display in order to know further details.

Step 3: Boundary selection. In this third step, we specify a practical and manageable model of the shared workspace, including the computer display and the viewport, upon which the performance comparison will later be based. To this end, we define the following assumptions:

1. the computer display has a typical 4:3 aspect ratio, with width W and height H;
2. the size of the shared workspace is a multiple, n, of the size of the computer display;
3. the shared workspace is filled with $nW \times nH$ objects;
4. every object has the same probability of being changed at any time.

The following two assumptions apply to design scenarios A and B, respectively:

5. the list of objects is H units high and one unit wide;
6. the radar view is square, $n \div 5$ units high, rounded up to the nearest integer;

Two more assumptions apply only to design scenario C:

7. by pressing a keyboard cursor key, the viewport is moved H units (up and down keys) or W units (left and right keys) in the respective direction;
8. the opposite borders of the shared workspace are linked together, making it possible to go, for example, from the left-most edge to the right-most edge directly.

In these circumstances, as the size of the shared workspace increases, so does the number of objects and also the radar view (for design scenario B), which make these assumptions reasonable. Assumption 4 is a convenient adaptation of reality, because usually some objects, e.g., text documents, are more frequently updated than others over a given period of time. Note that in design scenarios A and B the computer display is not entirely dedicated to the viewport, because the list of object names and the radar view take up space.

Step 4: Shared workspace performance. In this final step, we use keystroke-level model operators to characterize the actions that group members have to execute to locate updated objects in the shared workspace. The predicted execution time for this critical scenario will be obtained from the required sequence of operators, which depends on the design scenarios.

In all three design scenarios the estimated execution time is given by a weighted sum, $T = P_i T_i + P_o T_o$, considering two possible cases: (1) the updated object is already shown inside the viewport, with probability P_i and execution time T_i; or (2) the object is outside of the viewport, with P_o and T_o.

To calculate P_i we count the number of objects that can be seen on the viewport and divide it by the total number of objects in the shared workspace. To obtain P_o we simply subtract P_i from 1. For example, if we consider a computer display with $W = 4$ and $H = 3$ and a shared workspace with eight by six units ($n = 2$), then:

- in design scenario A, the list of object names takes up three units (by assumption 5), so the number of objects visible on the viewport is nine, and $P_i = 0.19$;
- in design scenario B, the radar view is a square, one unit high (by assumption 7), giving a total of 11 objects visible on the viewport, so $P_i = 0.23$;
- in design scenario C, the viewport uses the entire computer display, therefore $P_i = 0.25$.

After obtaining P_i and P_o we now describe the fine-grained details of how to locate an updated object by using sequences of keystroke-level model operators, which will ultimately provide the T_i and T_o

execution times. The keystroke-level model sequence for the first case—the updated object is visible on the viewport—is just a M operator, from performing a visual search on the viewport and finding the object in question. Since the M operator takes up 1.2 seconds (from Table 1), T_i = 1.2 seconds for all three design scenarios.

The calculation of T_o, for the case where the updated object is outside of the viewport, is done as follows: for design scenarios A and B, the user fails to find the object on the viewport, M, then searches the list of updated objects (or the radar view), another M, then points the mouse cursor to the list entry (or to the dark-filled dot on the radar), P and finally clicks the mouse button, KK (press and release), causing the updated object to appear on the viewport. The complete keystroke-level model sequence for scenarios A and B is MMPKK, with a predicted execution time of T_o = 3.7 seconds (see individual operator times in Table 1).

Regarding design scenario C, since the updated object is initially not visible on the viewport, the user has to navigate through the shared workspace using the keyboard cursor keys, guided by the pointing arrows at the border of the computer display (see Figure 4). In this case, the sequence of keystroke-level model operators depends on the sizes of the shared workspace and the viewport, because the larger the portion of the shared workspace that is outside of the viewport, the more cursor key presses are necessary to reach the updated object. To simplify, we consider the existence of an equation, $\overline{m}$, that calculates the *average* number of viewport moves to reach an updated object (for more details, see Appendix).

The sequence of keystroke-level model operators for locating an updated object that is outside of the viewport, in design scenario C, can finally be expressed as an M, the search for the object on the viewport (and not finding it), followed by KKM, which is a press and release of a keyboard cursor key plus a visual search, repeated $\overline{m}$ times. The corresponding predicted execution time is given by $T_o = 1.2 + 1.4 \times \overline{m}$ seconds (see individual operator times in Table 1. Also, since the user only presses the keyboard cursor keys while moving the viewport, and presumably leaves the hand on these keys, we assume that the predicted execution time to press or release one of these keys is the same as with a mouse button, that is, K = 0.1 seconds.

At this point, it is possible to compare the estimated execution time for locating an updated object, for all three design scenarios; the results for a computer display with size W = 4 and H = 3 and various sizes of the shared workspace are illustrated in Figure 5.

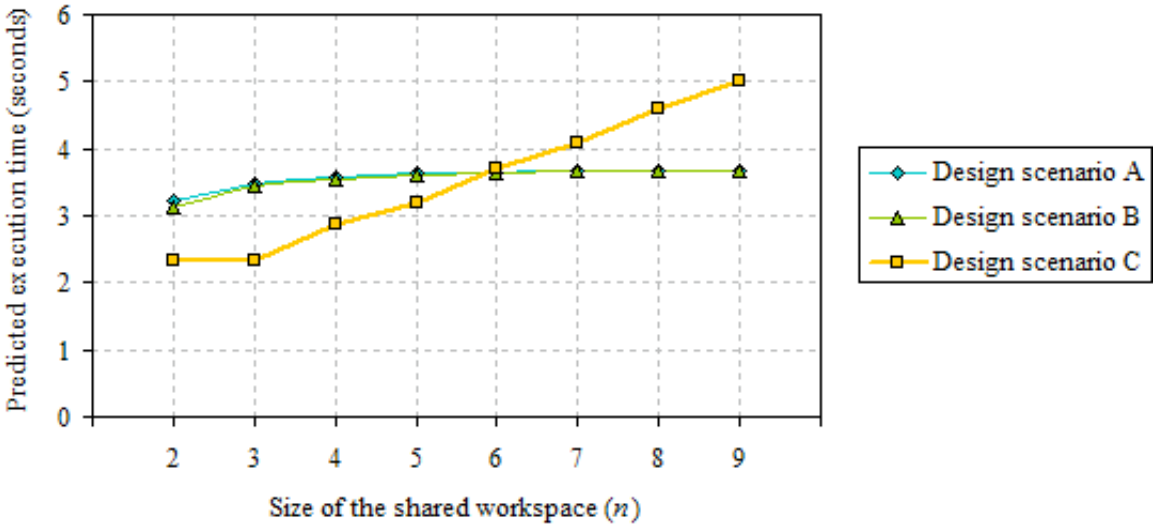

Figure 5: Predicted execution time for locating an updated object in a shared workspace

In summary, Figure 5 shows that: (1) there is almost no difference in using a list of object names or a radar view to locate an updated object, because the predicted execution times in design scenarios A and B are very similar; (2) the times for design scenarios A and B rapidly converge to and reach a maximum

of, 3.7 seconds; (3) design scenario C has a lower execution time for shared workspaces with up to five times the size of the computer display ($n \leq 5$); and (4) for larger shared workspaces, the predicted time in design scenario C increases by about 0.45 seconds with each unit of n.

Additionally, it is interesting to note that the trends displayed in Figure 5 are effectively *independent* of the size of the computer display, meaning that the graph may be seen as an easy-to-use tool in the hands of the designer, whenever the assumptions apply.

Reserving objects

In this second case, we apply the method to evaluate the performance of a shared workspace that enables users to reserve selected objects. A reserved object can only be changed by the user who made the reservation; the other users have to wait for the object release. In these circumstances, it is important that users be aware of which objects are currently reserved, otherwise time may be wasted in failed reservations, or work plans may be rendered inapplicable too often.

When reserving objects, users can experience one of two outcomes: a successful object reservation or a failure. The design challenge is to minimize the time wasted on failed reservations in situations where users try to simultaneously reserve the *same* object, this being the critical scenario under analysis.

Step 1: Groupware interface. The groupware interface provides a public, updated view of the shared workspace. There also exist several private workspaces, one for each user, allowing them to do individual work on reserved objects. However, the modelling of these private workspaces is out of scope, since we are only interested in collaborative actions.

A reservation in the shared workspace operates in the following way: first the objects are selected and then they are dragged out of the shared workspace into the private workspace. The objects are released when they are dragged back into the shared workspace. No awareness about the state of the objects is provided to the group of users; this is design scenario A.

In design scenario B, upon a successful reservation of objects, the shared workspace displays a letter next to them, identifying the current owner. This increases group awareness and reduces inadvertent selections of already reserved objects. The letter disappears when the objects are released.

In design scenario C, while a user is selecting objects, a rectangle that comprises those objects is shown on the shared workspace. The main reason for this refinement is the production of fine-grained and up-to-date awareness information.

We assume that the user's moves are restricted to being done with a single button mouse.

In summary, the groupware interface can be characterized as follows: (1) one shared workspace holds all public objects; (2) awareness information is provided by feed-through; (3) awareness is supported by an owner letter after a reservation and by a rectangle during the selection of objects; (4) there is a loose coupling between individual work and group awareness.

Step 2: Critical scenarios. The critical scenario happens when users try to reserve the same object in parallel. Naturally, only one user will succeed.

In design scenario A, users behave as if all objects are available, because they look the same in the shared workspace. When users start a reservation on the same object(s) at the same time, all but one user will receive an error message.

In design scenario B, users will not try to reserve objects having owner letters attached to them. However, because these letters are only shown after *all* the steps in a reservation have been performed, it is possible that two or more users try to reserve the same, apparently available, objects.

Finally, in design scenario C, besides looking at owner letters, users may also see rectangles being drawn around objects on the shared workspace, meaning that other users are selecting objects presumably to reserve them afterwards. As a consequence, users will likely choose other objects to work with.

Step 3: Boundary selection. We make three assumptions regarding the shared workspace and the work patterns of the group of users:

1. all objects on the shared workspace are visible on the computer display;
2. feed-through is instantaneous (i.e., no network delay);
3. it is unlikely that more than *two* users select the same object at the same time;
4. the first user entering a competition for the same objects always succeeds in making the reservation.

Assumptions 1, 2 and 4 reduce complexity and make convenient the analysis of the shared workspace. Assumption 3 seems somewhat exaggerated given that all objects fit on the computer screen, especially in design scenario A, or when the group has many users. However, this has little importance because we can suppose instead that the reservations are done on a very large shared workspace (with the help of a viewport); this changes none of the functional details of a reservation, while making the assumption more plausible. Thus, for the performance comparison we will consider two users competing for the same object.

Step 4: Shared workspace performance. We now focus on the fine-grained details of how to reserve objects, to the point where this action can be described with keystroke-level model operators. It is interesting to note that the sequence of keystroke-level model operators will be the same in all three design scenarios; the difference in performance will be caused by the availability and timeliness of the awareness information during the critical scenario.

Regarding the keystroke-level model sequence, we assume that the user must first search for one or more objects (to work with) on the shared workspace; this is converted into an `M` operator. Once the objects are located, the user moves the mouse pointer near the top-left corner of an imaginary rectangle that will encompass all the objects of interest, `P`, presses the mouse button, `K` and moves the pointer to the opposite corner of the rectangle, `P`. The user then releases the mouse button, `K`, to complete the selection.

The last part of a reservation is done by dragging the selected objects out of the shared workspace: the user adjusts the mouse pointer so that it rests on top of one of the selected objects, `P`, presses the mouse button, `K`, drags the selected objects out of the shared workspace, `P` (no `M` operator is required because the workspaces are always in the same place) and releases the mouse button, `K`. The complete sequence of operators is `MPKPKPKPK`, which has a predicted execution time of 6 seconds.

After having determined the sequence of operators for making a reservation, we now focus on the comparison of performance in the critical scenario—when two users have the intention of reserving the same objects—for designs A and B. Considering the design scenario A, the best case happens when two users start the reservation for the same object(s) at the same time. In this case, after the 6.0 seconds needed for a complete reservation, the second user (see assumption 4) notices an error message (an `M` operator) and starts again with another object, which takes an additional 6.0 seconds. The best execution time is then 13.2 seconds. The worst case happens when the second user begins just after the first user finishes a reservation; since no awareness information is provided, the total execution time increases to 19.2 seconds (see design scenario A in Figure 6).

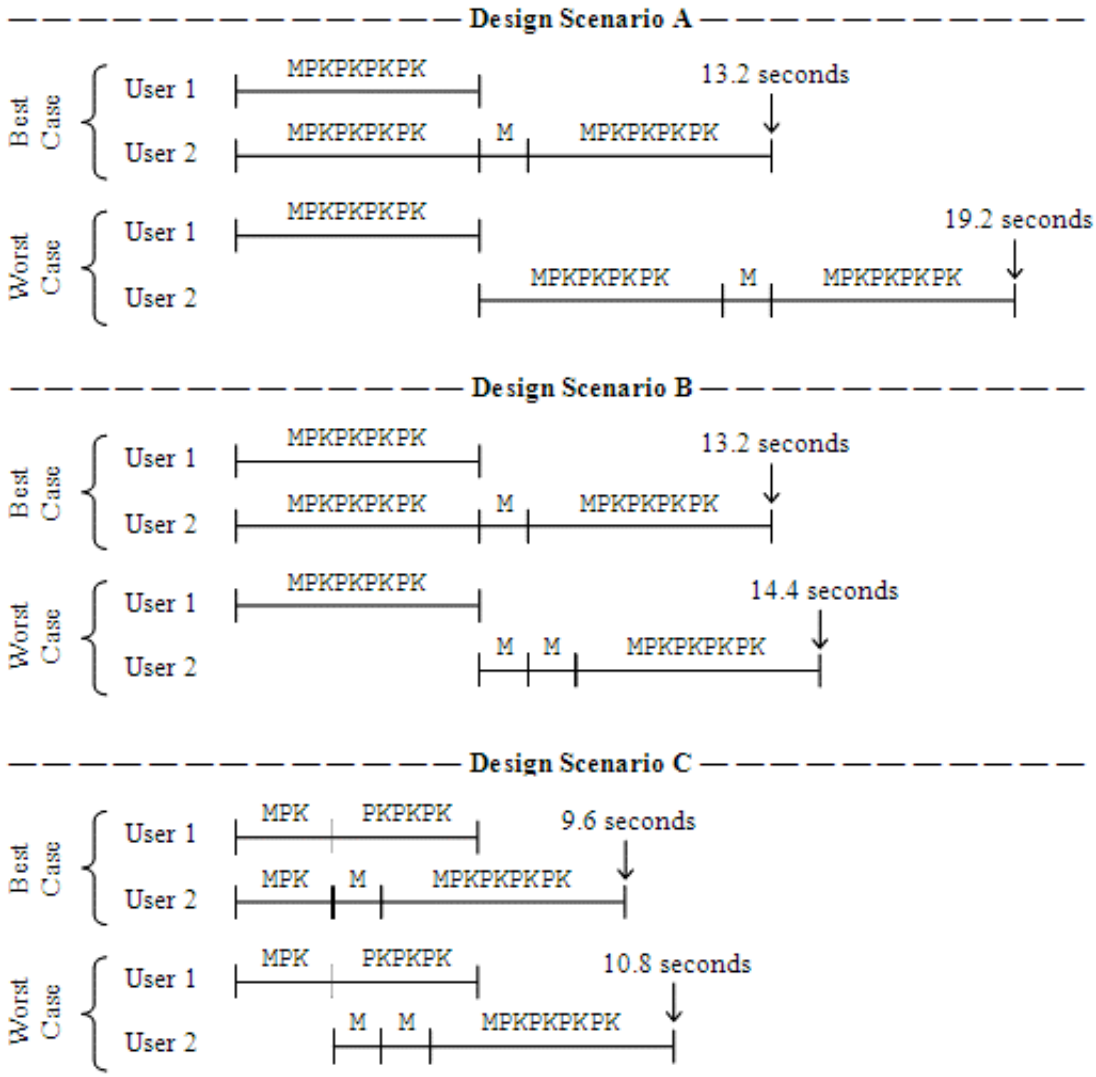

Figure 6: Best and worst execution times for reserving objects on the shared workspace

Before progressing to the next design scenario, we highlight the following point: we assume the time to detect an error message, i.e., to notice a conflict indicator, is equal to the duration of an M operator (see Table 1). It may be argued that it is necessary to test this assumption by using laboratory experiments, as others have done for specific tasks (Olson and Olson 1990). However, our decision here is based upon the *simplifying logic in the keystroke-level model* and, indeed, 1.2 seconds has been assumed before as the duration of generic visual perception operators (Kieras 2003).

For design scenario B, the best case is identical to that of scenario A. However, the execution time for the worst case is significantly reduced because the second user can interrupt an ongoing reservation as soon as the owner letter is displayed on the shared workspace. We represent this situation with two M operators: the first corresponds to the initial M of any reservation, while the second M is for interpreting the critical situation. The total execution time for the worst case is now 14.4 seconds (see design scenario B in Figure 6).

The optimisation considered in design scenario C provides awareness information upon the selection of the first object, i.e., just after a sequence of MPK (instead of the full MPKPKPKPK). In these circumstances both the best and worst cases benefit from reduced execution times (see design scenario C in Figure 6). If the two users start the reservation at the same time, then at about 2.4 seconds they both see their simultaneous selections on the shared workspace. Then, the second user (by assumption 4) decides to stop the current selection and starts another one, an M followed by a new reservation, taking a total of 9.6 seconds. The worst case takes 10.8 seconds; its explanation is analogous to the worst case for scenario B, except the awareness supplied by the owner letter upon a full reservation is substituted by the awareness provided by the selection of the first object.

In summary, the method brought quantitative insights about the role of feed-through information in group work support, predicting that design scenario C is faster than B by 3.6 seconds and that B is faster than A by about 4.8 seconds, but only in the worst case scenario.

Negotiating requirements

In this third case, we demonstrate the application of the method to an existing groupware tool that supports collaborative software quality assessment, using the *software quality function deployment* methodology (Haag *et al*. 1996). The objective of this tool is to facilitate the *software quality function deployment* negotiation process by providing mechanisms in a same-time, different-place mode. Our starting point in this case is a previous experiment with the tool that gathered data in questionnaires and that reported some usability problems, namely that it was considered difficult to use. Further details about this tool and about the previous evaluation can be found in Antunes *et al*. (2006).

Step 1: Groupware interface. The tool has two shared workspaces: the *software quality function deployment* matrix and *current situation*. The matrix allows users to look over a matrix of correlations between product specifications and customer requirements, as well as to observe which correlations are under negotiation. Limited awareness information is provided by the matrix, but there is a coupling mechanism that allows users to look into and *modify*, a cell. This coupling mechanism leads users to the *current situation,* where they can observe the negotiation state in detail, including the proposed correlation, positions in favour or against and supporting arguments and, ultimately, express or update his or her arguments and positions. We briefly characterize the two shared workspaces in terms of awareness input, output and coupling input devices in Figure 7 and Figure 8.

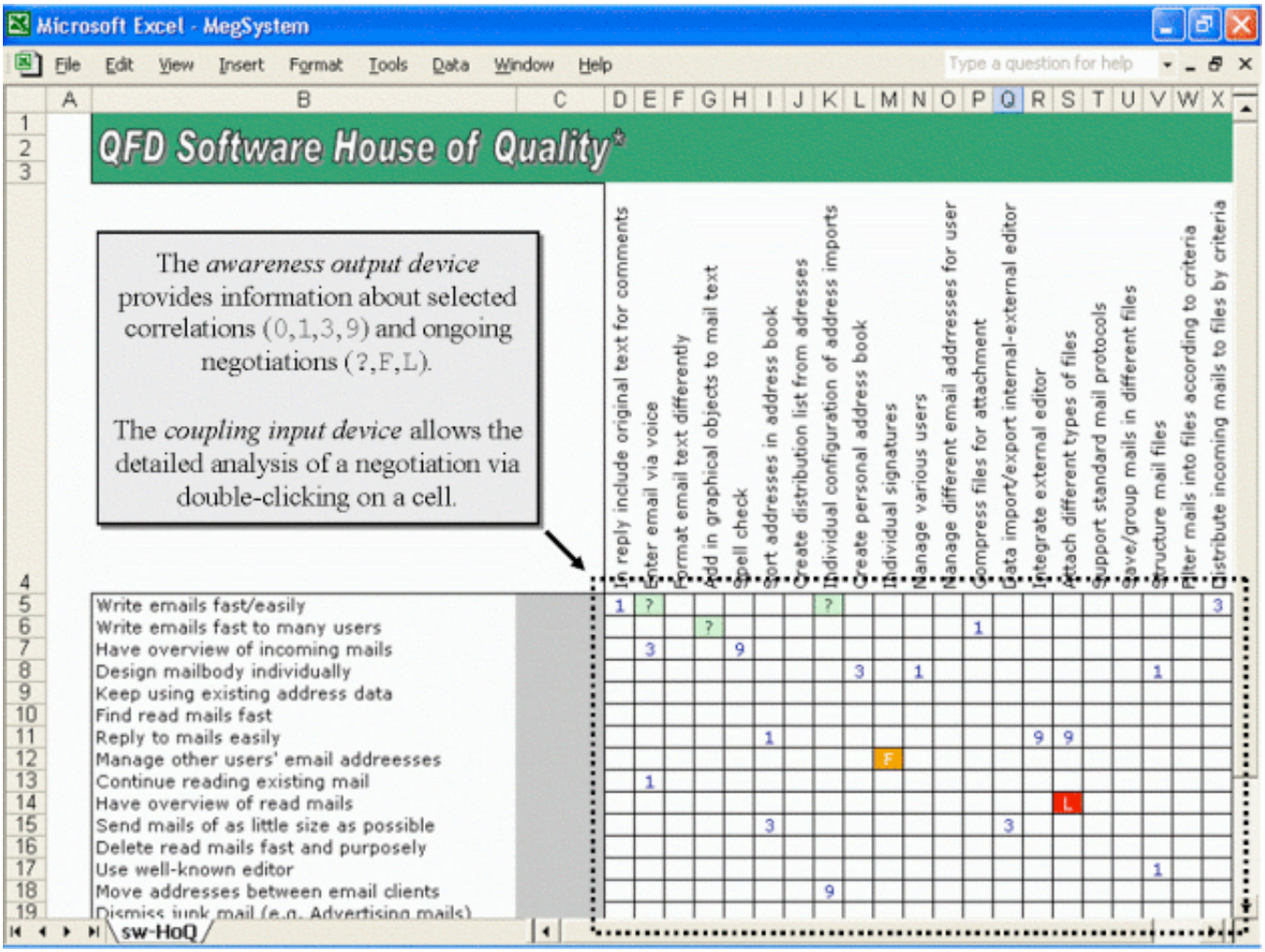

Figure 7: The *software quality function deployment* matrix

The digits inside cells in Figure 7 represent the correlations between customer requirements (listed on the

left) and product specifications (top of the matrix), going from *weak* (1) up to *strong* (9). When consensus is not verified for a particular cell, the digit is replaced by the symbol ? and, in more extreme cases, by an F or an L, which mean a user has issued a firm position or locked the cell respectively.

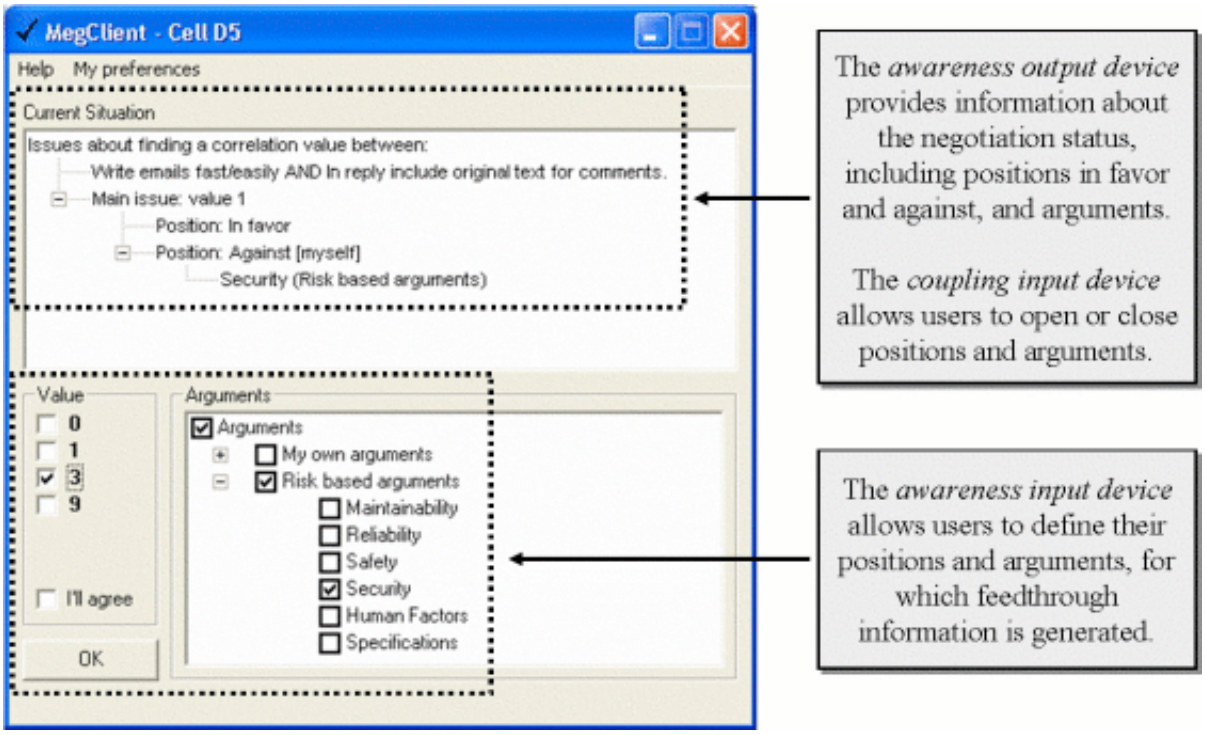

Figure 8: The *Current Situation* shared workspace

For the purpose of this case, we focus on the condition where a negotiation is in progress. The design challenge is to minimise the time needed for a user to express or update his or her position to help the group reach a faster consensus about a particular cell.

Step 2: Critical scenarios. In this analysis we assume the user has arrived at the *current situation* shared workspace with the purpose of examining the negotiation state in detail. As currently implemented by the tool, this information is hierarchically organized, showing: (1) the product specifications and customer requirements under negotiation; (2) the currently proposed correlation; (3) positions in favour, followed by positions against the currently proposed correlation; and (4) arguments supporting positions in favour or against. We call this design scenario A.

An alternative design scenario B considers a variation in the way status information is shown to the user. We assume that users assign importance to aggregate information about the number of positions against or in favour, neglecting positions where there is a clear push towards one side or the other and analysing arguments in detail only when positions are balanced.

The selected critical scenario considers the proposal, by a user, of an alternative correlation value in *current situation,* after having analysed the negotiation state. This is a critical scenario because it reflects a core and repetitive activity during the negotiation process, therefore influencing individual and group performance.

We also consider a variation in the number of users involved in the negotiation process. The *current situation* displays the positions and arguments for up to three users (see Figure 8). Beyond this number, the user has to scroll down the window to completely analyse the situation.

Step 3: Boundary selection. Since this case is based upon the improvement of an existing tool, the design space for the shared workspace and its operation by the users has already a practical and manageable dimension. However, given the nature of design scenario B, we consider, additionally, the following three assumptions:

1. Users assign importance to aggregate information (see description in the previous step);
2. Two conditions, with three and six users, will be involved in the critical scenario;
3. The probability of having unbalanced positions is 25% (This is the probability of having an absolute majority with three or six voters, assuming a uniform distribution. For three voters, the absolute majority requires having all in favour or against, i.e., two out of eight combinations, or 25%.).

Regarding assumption 2, we assume that having more than six users negotiating the same cell is a rare event, which does not deserve further analysis.

Step 4: Shared workspace performance. In design scenario A with three users, we have: the interpretation of the negotiation status, M, followed by a decision, M, which is expressed by the selection of a check box, a PKK and pressing the *ok* button, PKK. This gives MMPKKPKK, which has a total execution time of 5.0 seconds. With six users, the execution time increases to 8.6 seconds, corresponding to MPKPK MMPKKPKK, in which the MPKPK operators are related to scrolling.

We should note that some M operations analysed by this method may extend beyond the routine tasks typically modelled by the keystroke-level model. For instance, the first M in the MMPKKPKK sequence above is associated with the interpretation of the negotiation status, which is significantly more complex that the selection of a check box modelled by the second M. The likely consequence of this situation is that the times measured would become inconsistent. However, our intention is not to define precise time values for sequences of operators, but to compare various sequences of operators in a *consistent* way across several alternative designs. Of course, there is a risk associated with modelling more complex cognitive tasks with a single M, which has to be understood and assumed by the designer, but our assumption is that this risk is equally distributed among the alternative designs, so they may still be compared.

Considering design scenario B, two situations can happen: either the positions are balanced (a tie or a simple majority), or they are unbalanced (i.e., absolute majority). In the unbalanced case, we assume the user will neglect arguments and, thus, we have MMPKKPKK (5.0 seconds to execute), similar to the previous scenario with three users.

In the balanced case, the user will analyse the positions in detail by first interpreting the negotiation status, M, followed by the opening of the list of favourable arguments, PKK and corresponding analysis, M, upon which the list is closed, PKK, to give room for the opening and interpretation of the against arguments, PKK M, so that, finally, the decision is made, M and a check box is selected, PKKPKK. The total execution time for the balanced case, MPKKMPKKPKKMMPKKPKK, is, then, 11.3 seconds. We note that these measures apply to the scenarios with three and six users.

We also assume that the probability of having unbalanced positions is 25% (see assumption 3). Hence, in these circumstances, the average execution time for scenario B is about (0.75×11.3) + (0.25×5.0) ≈ 9.8 seconds, which is higher than scenario A for both three and six users.

In summary, design scenario B may be better than or equal to scenario A, but there is a 75% probability that it is worse than scenario A, which severely penalizes the overall appreciation of design scenario B.

Discussion and implications for design

The three cases: locating updated objects, reserving objects and negotiating requirements, heavily depend on shared workspaces to orchestrate multiple users accomplishing collaborative tasks. The design of these workspaces is, thus, critical to the overall group performance. The method we describe in this paper provides a common criterion (execution time in critical scenarios) to evaluate shared workspace performance, allowing designers to benchmark various solutions to predict which functionalities offer the

best performance.

We note that the three cases studied in this paper are quite distinct. In the locating updated objects case, the performance differences occurred because of the alternative ways of manipulating a coupling mechanism, a viewport, to navigate through shared workspaces with varying sizes. In the reserving objects case, we focused on the availability and timeliness of awareness information, to evaluate the performance in environments where users act opportunistically. Finally, in the negotiating requirements case, we analysed how a coupling mechanism could be designed to conserve individual cognitive effort. Taken as a whole, the method contributed to formative evaluation and offered indications about the potential performance of users working with shared workspaces.

The proposed method has two important limitations, which we discuss here. First, it assumes a narrow-band view about collaboration, restricted to shared workspaces and their mediation roles. This contrasts with other groupware evaluation methods, that offer a wide-band view about collaboration, encompassing, for example, multiple communication channels, coordination policies and broader issues, such as group decision making or learning. However, the trade-off to ponder is that the method restricts the view to increase the detail about the mediating role of shared workspaces. This restricted view has ample justification in contexts where shared workspaces are heavily used, even when users perform intellective tasks (such as in the negotiating requirements case, where users apply their expertise to evaluate software quality, but are still requested to repetitively operate the tool).

Second, the method is somewhat limited by the selection of the critical scenarios. As designers and evaluators, we have to consider whether the selected critical scenarios are representative and have sufficient impact on the overall collaborative task to deserve detailed analysis. We conjecture that the bias that may exist in preliminary phases of the design process (caused by new technologies or applications, by lack of knowledge about the collaborative context, or for other reasons) can be reduced by applying the method at a later time, e.g., to enable a richer understanding of how people collaborate; this has happened in the negotiating requirements case, in which usability problems were identified by users of an existing tool, this being the starting point for the selection of the shared workspaces and the critical scenarios. This possibility of using the proposed method in tandem with other evaluation methods, such as field studies, laboratory experiments, or discount methods, gives designers and evaluators an additional tool to rely on. Finally, we note that critical scenarios are commonly used as a sampling strategy in qualitative inquiry, allowing generalization (Miles and Huberman 1994). The proposed method combines qualitative and quantitative approaches to the same purpose.

The last point of this discussion relates to the framing of our approach in the larger picture of groupware acceptance and success with interest groups and organizations. Our focus on shared workspace performance is linked to the *efficiency* of coworkers, a fundamental element of usability, which plays an important role in intensive, recurring tasks performed by experts, such as military applications, collaborative games and collaborative virtual environments (Baeza-Yates and Pino 2006). In these scenarios, the cost of applying our approach may be justified by the benefits gained. We recognise, however, that there are additional factors that influence groupware acceptance and success, such as the *disparity of work and benefit* and *disruption of social processes* (Grudin 1994), for which other evaluation methods may be the most appropriate ones.

Conclusions

Confronting the obtained results with the driving forces mentioned in the Introduction, we may conclude from this research that the proposed method can be used to quantitatively predict and compare the performance of shared workspaces, without requiring users or the development of functional prototypes. Specifically, available knowledge about human information processing models can be applied to predict execution time in critical scenarios, which have a potentially important effect on individual and group performance.

Our aim with this method is that it becomes a handy tool for groupware designers: a tool for making quick measures and calculations, enabling performance optimisations; a tool that complements the perspectives and outcomes provided by other evaluation methods.

Research described in this paper is a preliminary step in the direction of exploring human information processing models to evaluate shared workspace design. Our performance estimates were based upon experimental measures of time spent by humans executing single user operations. Experimental research with groupware will be accomplished in the future, in an attempt to provide estimates for typical groupware interactions in critical scenarios.

Acknowledgements

This work was partially supported by the Portuguese Foundation for Science and Technology, Project POSI/EIA/57038/2004 and by Fondecyt (Chile) Project No. 1080352.

References

- Antunes, P., Ramires, J. & Respicio, A. (2006). Addressing the conflicting dimension of groupware: a case study in software requirements validation. *Computing and Informatics*, **25**(6), 1001–1024.
- Baker, K., Greenberg, S. & Gutwin, C. (2002). Empirical development of a heuristic evaluation methodology for shared workspace groupware. In *Computer Supported Cooperative Work. Proceedings of the 2002 ACM Conference on Computer Supported Cooperative Work, New Orleans, Louisiana, USA*, (pp. 96–105). New York, NY: ACM Press.
- Baeza-Yates, R. & Pino, J.A. (2006). Towards formal evaluation of collaborative work and its application to information retrieval. *Information Research*, **11**(4), paper 271. Retrieved 4 August, 2008 from http://informationr.net/ir/11-4/paper271.html. (Archived by WebCite® at http://www.webcitation.org/5dJeeBOhk)
- Beyer, H. & Holtzblatt, K. (1998). *Contextual design: defining customer-centered systems*. San Francisco, CA: Morgan Kaufmann Publishers.
- Card, S.K., Moran, T.P. & Newell, A. (1980). The keystroke-level model for user performance time with interactive systems. *Communications of the ACM*, **23**(7), 396–410.
- Card, S.K., Newell, A. & Moran, T.P. (1983). *The psychology of human-computer interaction*. Mahwah, NJ: Lawrence Erlbaum Associates.
- Cosquer, F.J.N., Antunes, P. & Verissimo, P. (1996). Enhancing dependability of cooperative applications in partitionable environments. In A. Hlawiczka, J. Silva & L. Simoncini (Eds.) *Proceedings of the 2nd European Dependable Computing Conference, Taorima, Italy*. (pp. 335–352). Berlin: Springer Verlag. (*Lecture Notes in Computer Science*, *1150*). Retrieved 24 December, 2008 from http://www.di.fc.ul.pt/~paa/papers/edcc-96.pdf. (Archived by WebCite® at http://www.webcitation.org/5dJerTJqR)
- Dewan, P. & Choudhary, R. (1995). Coupling the user interfaces of a multiuser program. *ACM Transactions on Computer-Human Interaction*, **2**(1), 1–39.
- Douglas, S.A. & Kirkpatrick, A.E. (1999). Model and representation: the effect of visual feedback on human performance in a color picker interface. *ACM Transactions on Graphics*, **18**(2), 96–127.
- Fjermestad, J. & Hiltz, S. (1999). An assessment of group support systems experimental research: methodology and results. *Journal of Management Information Systems*, **15**(3), 7–149.
- Grudin, J. (1994). Groupware and social dynamics: eight challenges for developers. *Communications of the ACM*, **37**(1), 92–105.
- Gutwin, C., Benford, S., Dyck, J., Fraser, M., Vaghi, I. & Greenhalgh, C. (2004). Revealing

delay in collaborative environments. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Vienna, Austria*, (pp. 503–510). New York, NY: ACM Press.
- Gutwin, C. & Greenberg, S. (1999). The effects of workspace awareness support on the usability of real-time distributed groupware. *ACM Transactions on Computer-Human Interaction*, **6**(3), 243–281.
- Haag, S., Raja, M.K. & Schkade, L.L. (1996). Quality function deployment usage in software development. *Communications of the ACM*, **39**(1), 41–49.
- Hill, J. & Gutwin, C. (2003). Awareness support in a groupware widget toolkit. In *Proceedings of the 2003 international ACM SIGGROUP Conference on Supporting Group Work, Sanibel Island, Florida, USA*, (pp. 258–267). New York, NY: ACM Press. Retrieved 24 December, 2008 from http://hci.usask.ca/publications/2003/maui-group03.pdf (Archived by WebCite® at http://www.webcitation.org/5dJfpLGgC)
- Hughes, J., King, V., Rodden, T. & Andersen, H. (1994). Moving out from the control room: ethnography in system design. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work, Chapel Hill, North Carolina, USA*. (pp. 429-439). New York, NY: ACM Press.
- John, B.E. & Kieras, D.E. (1996). Using GOMS foruser interface design and evaluation: which technique? *ACM Transactions on Computer-Human Interaction*, **3**(4), 287-319. Retrieved 24 December, 2008 from https://eprints.kfupm.edu.sa/74628/1/74628.pdf (Archived by WebCite® at http://www.webcitation.org/5dJgMtnqv)
- Kieras, D. (2003). GOMS models for task analysis. In D. Diaper & N. Stanton (Eds.), *The handbook of task analysis for human-computer interaction* (pp. 83–116). Mahwah, NJ: Lawrence Erlbaum Associates.
- Kieras, D.E. & Santoro, T.P. (2004). Computational GOMS modeling of a complex team task: lessons learned. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Vienna, Austria, April 24-29, 2004* (pp. 97–104). New York, NY: ACM Press.
- Malone, T.W. & Crowston, K. (1994). The interdisciplinary study of coordination. *ACM Computing Surveys*, **26**(1), 87–119.
- Miles, M.B. & Huberman, M. (1994). *Qualitative data analysis: an expanded sourcebook*. Thousand Oaks, CA: Sage Publications.
- Min, D., Koo, S., Chung, Y.H. & Kim, B. (1999). Distributed GOMS: an extension of GOMS to group task. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, Tokyo, Japan 12-15 October, 1999*. Volume 5. (pp. 720–725). NewYork, NY: IEEE Press
- Olson, J. & Olson, G. (1990). The growth of cognitive modeling in human-computer interaction. *Human-Computer Interaction*, **5**(2), 221–265.
- Pinelle, D. & Gutwin, C. (2002). Groupware walkthrough: adding context to groupware usability evaluation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Changing our World, Changing Ourselves, Minneapolis, Minnesota, USA*, (pp. 455–462). New York, NY: ACM Press.
- Pinelle, D., Gutwin, C. & Greenberg, S. (2003). Task analysis for groupware usability evaluation: modeling shared-workspace tasks with the mechanics of collaboration. *ACM Transactions on Computer-Human Interaction*, **10**(4), 281-311. Retrieved 24 December, 2008 from http://bit.ly/VyJj
- Rajan, S., Craig, S.D., Gholson, B., Person, N.K. & Graesser, A.C. (2001). AutoTutor: incorporating back-channel feedback and other human-like conversational behaviors into an intelligent tutoring system. *International Journal of Speech Technology*, **4**(2), 117–126.
- van der Veer, G. & van Welie, M. (2000). Task based groupware design: putting theory into practice. In *Proceedings of the 3rd Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques, New York, NY, USA*. (pp. 326–337). New York, NY: ACM Press. Retrieved 24 December, 2008 from http://www.cs.vu.nl/~gerrit/gta/docs/Dis2000.pdf (Archived by WebCite® at

http://www.webcitation.org/5dJiINa43)
- Wensveen, S.A.G., Djajadiningrat, J.P. & Overbeeke, C.J. (2004). Interaction frogger: a design framework to couple action and function through feedback and feedforward. In *Proceedings of the 5th Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques, Cambridge, Mass., USA*. (pp. 177–184). New York, NY: ACM Press. Retrieved 24 December, 2008 from http://bit.ly/hBPX (Archived by WebCite® at http://www.webcitation.org/5dJihkHwb)

How to cite this paper

Ferreira, A., Antunes, P. & Pino, J.A. (2009). "Evaluating shared workspace performance using human information processing models" *Information Research*, **14**(1) paper 388. [Available at http://InformationR.net/ir/14-1/paper388.html]

Appendix: Equation for average number of viewport moves

In this appendix, we present a generic equation for $\overline{m}$, i.e., the average number of viewport moves to reach an arbitrary object that is outside of the viewport. This equation is applicable to the critical scenario locating updated objects and more specifically to design scenario C.

Because we are interested in a *generic* equation; that is, one where the shared workspace may be a rectangle of *any* proportion, we replace *n* with integers *X* and *Y* (see examples in Figure 9).

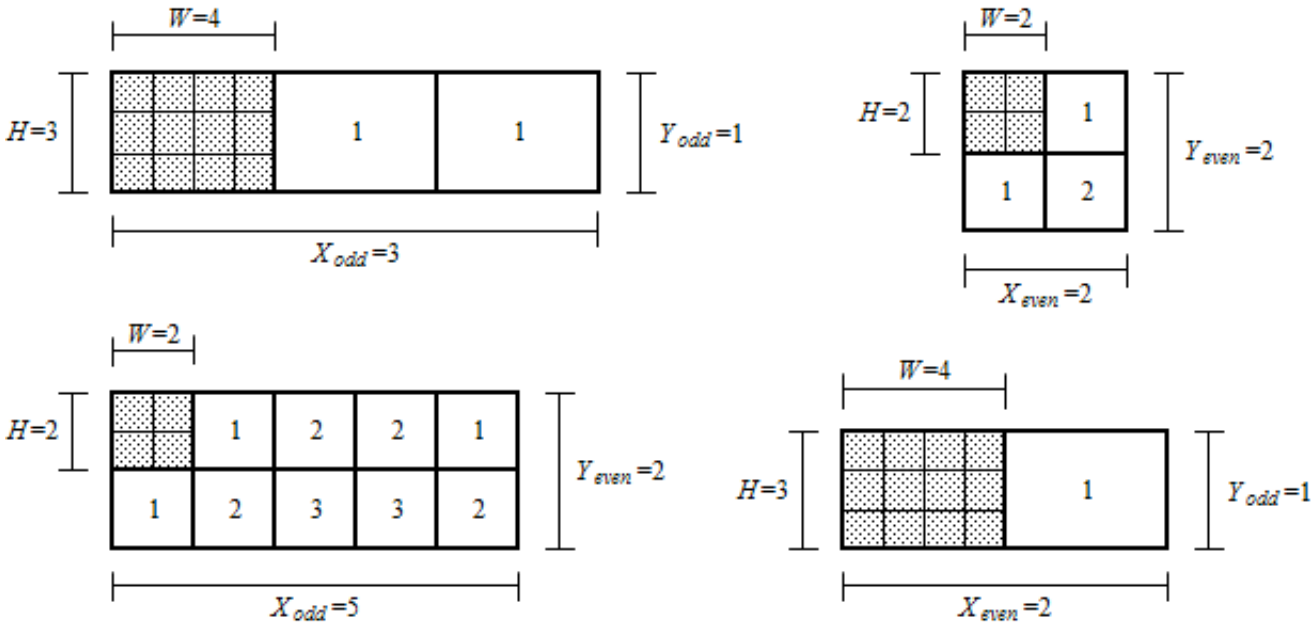

Figure 9: Shared workspaces with even or odd *X* and *Y* sizes. For convenience, the viewport (with grey background) is on the top-left corner of the shared workspace

The numbers in Figure 9 indicate the minimum number of viewport moves to reach objects in those parts of the shared workspace and were obtained by following assumptions 7 and 8. Given these numbers and

based upon assumptions 3 and 4, we derived Equation 1 for determining the average number of viewport moves (i.e., cursor key presses) for any configuration of shared workspace and viewport.

$$\overline{m}(X,Y) = \begin{cases} 0 & \text{if } X = 1,\ Y = 1, \\ \dfrac{X+Y}{4} & \text{if } X \text{ is odd},\ Y \text{ is odd}, \\ \dfrac{X(Y^2+XY-1)}{4(XY-1)} & \text{if } X \text{ is even},\ Y \text{ is odd}, \\ \dfrac{Y(X^2+XY-1)}{4(XY-1)} & \text{if } X \text{ is odd},\ Y \text{ is even}, \\ \dfrac{XY(X+Y)}{4(XY-1)} & \text{if } X \text{ is even},\ Y \text{ is even}. \end{cases}$$

Equation 1: Average number of viewport moves to reach a random object that is outside of the viewport

The data in Figure 5 were calculated by applying Equation 1 to cases where $X = Y = n$, from two to nine.

Last updated: 24 December, 2008
