UIMarks: quick graphical interaction with specific targets
Olivier Chapuis, Nicolas Roussel

To cite this version:

HAL Id: inria-00533398
https://hal.inria.fr/inria-00533398
Submitted on 8 Nov 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
UIMarks: Quick Graphical Interaction with Specific Targets

Olivier Chapuis
LRI - Univ Paris-Sud & CNRS; INRIA
F-91405 Orsay, France
chapuis@lri.fr

Nicolas Roussel
INRIA
F-59650 Villeneuve d’Ascq, France
nicolas.roussel@inria.fr

Figure 1: Sample use of UIMarks in a drag-and-drop situation. While dragging an icon (A), the user enters the UIMarks mode, points at a previously marked location using the bubble cursor technique (B), leaves the mode (C) and drops the icon (D). Here, the activation of the mark not only moved the cursor but also created a temporary mark at the initial cursor location. The user thus simply has to enter the mode again (E), point at the new mark (F) and leave the mode (G) to return there.

ABSTRACT
This paper reports on the design and evaluation of UIMarks, a system that lets users specify on-screen targets and associated actions by means of a graphical marking language. UIMarks supplements traditional pointing by providing an alternative mode in which users can quickly activate these marks. Associated actions can range from basic pointing facilitation to complex sequences possibly involving user interaction: one can leave a mark on a palette to make it more reachable, but the mark can also be configured to wait for a click and then automatically move the pointer back to its original location, for example. The system has been implemented on two different platforms, Metisse and OS X. We compared it to traditional pointing on a set of elementary and composite tasks in an abstract setting. Although pure pointing was not improved, the programmable automation supported by the system proved very effective.

ACM Classification: H.5.2 [Information interfaces and presentation]: User interfaces - Graphical user interfaces.

General terms: Design, Measurement, Performance, Experimentation, Human Factors

Keywords: Pointing, direct manipulation, macros.

INTRODUCTION
Pointing facilitation techniques aim at improving the acquisition of on-screen targets with a pointing device. Pointing being arguably one of the most fundamental tasks in HCI [1], research on these techniques is usually motivated by the idea that small improvements in speed or accuracy may result in large efficiency gains. Yet very few of the techniques proposed by HCI researchers are actually used in existing systems. One reason for this is probably that most of them are target-aware [25]: they require some knowledge about the size and position of the targets and sometimes the ability to modify them.

Deciding which of the on-screen objects should be considered as potential targets is a complex problem (e.g. windows or widgets, paragraphs or characters). Target-aware pointing techniques also tend to work best on sparse layouts: in dense layouts, occlusion and false activation problems can quickly obviate the potential benefits [1]. Moreover, basic interactions like rubber-band selection also require pointing and clicking on the void space between targets, which can be difficult with some techniques.

Target-aware pointing techniques usually aim at providing quick access to all possible targets at a given time. In this paper we present UIMarks, a system instead designed to facilitate access to a limited and specific set of targets. This system is not intended to replace traditional pointing but rather to supplement it by providing users with an alternative mode they can deliberately activate based on their specific needs. UIMarks supports the creation, configuration and use of user interface marks, graphical objects that explicitly locate on-screen targets and provide quick ways to interact with them. This results in a new interaction model that can probably not
be replicated with any other technique, including customized keyboard accelerators. Typical use consists of entering the mode, selecting a mark using a target-aware technique – we use the bubble cursor [11] – and leaving the mode, which triggers actions possibly associated to the mark (Figure 1).

The paper is organized as follows. After reviewing some related work, we present the design of the UIMarks system. We describe the techniques through which users interact with it and provide some implementation details. We then report on an experiment that compared UIMarks with traditional pointing on elementary and composite tasks and shows that although it does not improve pure pointing, the programmable automation it supports is very effective. We conclude with directions for future work.

RELATED WORK
A substantial body of literature exists on target acquisition. A fundamental tool in this area is Fitts’ Law [10, 19], that models movement time for the acquisition of a target of width $W$ at a distance $D$ as a linear function of the index of difficulty \( \log_2(\frac{D}{W} + 1) \). Target-aware techniques usually try to reduce $D$, to increase $W$, or both to facilitate pointing [1]. Endpoint prediction can be used to temporarily bring targets closer to the pointer or make it jump to them, for example [2, 12, 21]. The ninja cursors [16] reduce the average distance to targets by attaching multiple cursors to a single device and using knowledge about the targets to resolve pointing ambiguities. Expanding targets [20] dynamically change $W$ to provide a larger area to interact with at the focus of attention. Semantic pointing [5] and sticky icons [26] similarly expand the targets, but in motor space only. Another way to enlarge $W$ is to use an area cursor, i.e. a cursor with an activation area larger than the standard one pixel hotspot [15, 26]. The bubble cursor [11] refines this idea by dynamically resizing the area so that exactly one target is selected at any time.

Area cursors sometimes make it impossible to select the void space between targets (an intrinsic limitation of the bubble cursor, for example). DynaSpot [7] overcomes this problem by adapting the size of the cursor’s activation area depending on movement speed. Target-aware pointing techniques also tend to work best on sparse layouts, although space partitioning algorithms can be used to alleviate this problem [3]. Lastly, target-aware techniques can be quite visually distracting because of the growing and shrinking of objects, or because of the discontinuities introduced by pointer warping.

Adaptive solutions have been proposed that automatically compute a subset of the targets on which the pointing technique is then applied. Bubbling menus [24], for example, accelerate the selection of frequently used items by increasing their activation areas. It has also been suggested to aggregate pointer clicks and drags into a pseudo-haptic magnetic field to make frequently accessed targets easier to select without requiring prior knowledge of them [14].

Target-agnostic techniques similarly focus on user actions instead of possible targets. The angle mouse, for example, adjusts the control-display gain based on angular deviation [25]. MAGIC uses eye tracking to coarsely define an area of interest in which the pointer is automatically warped [27].

Rake cursor [6] uses the gaze position to select a cursor from a grid of several, obviating the potential disorientation caused by pointer warping at the cost of slightly increased visual clutter. All these techniques have some advantages over the target-aware ones, but have the disadvantage of considering all pixels equal which leads to limited performance improvements [25].

DESIGN PROCESS
The work that led to UIMarks started with the idea that desktop interactions might be facilitated if the pointing device could be used to control more than one on-screen pointer. We did not want the pointers to be system-defined, like with the ninja or rake cursors [16, 6]. Rather, inspired by works such as the local tools [4], we envisioned a system that would make it possible for users to create new pointers anywhere on-screen and to easily choose the one they would like to control at a given time. Different prototypes implementing this idea were shown to several HCI researchers, designers and students (Figure 2). The concept of multiple pointers attached to the same device generated little interest. But users of the prototypes very much liked the ability it provided to mark on-screen places where they might want to return to in the future. Instead of multiple pointers, we thus decided to support the creation, configuration and use of such marks.

The marks were initially envisioned as small on-screen colored disks that users could place using their standard pointing device. In order to minimize visual distraction, we decided that they would be accessible only when in a certain mode. Activating a mark, i.e. having the pointer automatically moved to the corresponding location, was thus a matter of entering the mode, selecting the mark and leaving the mode. We decided to use a bubble cursor and keyboard shortcuts to support this selection. The bubble cursor was chosen because it is particularly efficient in sparse target layouts and appropriate for interfaces with an explicit selection mode [11]. Note, however, that UIMarks is not conceptually tied to this technique. It is an adaptable system that can be used to define a set of targets on which a chosen target-aware technique is to be applied.

We wanted our work to be easily applicable to existing graphical environments without necessarily requiring the modification of applications. We implemented a first version of UIMarks meeting this requirement on two different platforms: OS X and the experimental X Window system Metisse [8]. The use of these implementations in real-life conditions suggested a number of improvements. Realiz-
ing the potential benefits of the system for big and multiple screens, we added the possibility to increase the acceleration of the bubble cursor to reduce clutching in these configurations. We also noticed that after using a mark, we often wanted to go back to the entering point, i.e. the point where we entered the mode.

As our interest shifted from simple pointing tasks to the operations that follow, we got particularly interested in repeated interaction sequences that imply going back and forth between two or more locations. We modified the system so that it automatically created a new mark at the entering point. This made back and forth movements easier but also created a lot of unneeded marks that acted as distractors for the bubble cursor. We tried different ways to alleviate this problem, e.g. by imposing an upper limit on the number of system-created marks or making them temporary. We tried several definitions of that term that combined temporal and activation limits. We also tried to attach marks to specific graphical objects rather than the screen.

Seeing different advantages and drawbacks in the aforementioned possibilities, we concluded there was a need for user-level configuration mechanisms. We created a simple graphical language to represent the different parameters and implemented basic interaction techniques to support their configuration. We then extended these mechanisms to support the specification of actions to be executed when a mark is activated, using some basic forms of end-user programming [18] to go beyond point-and-click interactions.

CURRENT DESIGN

The UIMarks system can be seen as an effort to provide some user programmable automation over direct manipulation interfaces. Marks are somewhat similar to the bookmarklets available in web browsers that trigger the execution of some code whenever an associated URL is visited. They can be placed anywhere on-screen and provide a visual representation of the associated action, which can be customized by the user. As opposed to traditional keyboard accelerators and macros, the system emphasizes recognition over recall and supports compound pointer-based interactions instead of the sole execution of pre-defined commands.

To illustrate the potential of the system, let us consider the simple but common case where one wants to facilitate the selection of tools in a palette. One starts by placing a mark \( M_1 \) on the palette. As one usually wants to return to the original locus of interaction after selecting a tool, one can configure \( M_1 \) to create a new temporary mark \( M_2 \) at the entering point. One can also place \( M_1 \) on a tool and configure it so it automatically clicks – i.e. selects the tool – and brings back the pointer to the entering point. This reduces the number of explicit actions but also limits the use of \( M_1 \) to the selection of one particular tool. To alleviate this, one can place additional similarly configured marks on other tools and offset them from their target to facilitate their selection with the bubble cursor. Instead of creating multiple marks, another possibility is to configure \( M_1 \) so that after moving the pointer over the palette, it gives its control back to the user until a button is clicked – i.e. a tool is selected – and then brings back the pointer to the entering point.

Before providing more details about the actual user interactions with the system and our two implementations, we will define more precisely the concept of a mark by specifying its attributes, the actions that can be attached to it and how it is graphically represented.

Mark attributes

A mark is a uniquely identifiable object associated to an on-screen \((x, y)\) position and further characterized by three attributes: its creator, target and lifetime.

Creator – Marks can be created by the user, in anticipation of future use. They can also be created by UIMarks itself as a consequence of the activation of another mark, or on behalf of other applications. An application might request the creation of a temporary mark on the \( \text{Ok} \) button of a dialog box, for example.

Target – Marks can be either attached to the screen, or to a particular graphical object such as a window or possibly a widget. In the latter case, activating the mark will raise the enclosing window and give it the keyboard focus. Moving, resizing, iconifying or closing the window will also impact the mark.

Lifetime – Marks can be permanent or temporary, lasting for only a limited time or number of activations.

Possible values for mark attributes can be summarized as:

\[
\text{creator} \in \{\text{user, uimarks, otherapp}\} \\
\text{target} \in \{\text{screen, window(id)}\} \\
\text{lifetime} \in \{\text{permanent, temporary}\}
\]

Actions

Each mark has an associated primary action that is triggered whenever the mark is activated. Incidental preceding and following actions can also take place immediately before and after the primary one. When executed, all three actions have access to the mark attributes as well as the location of the entering point.

Preceding action [optional] – The only possible preceding action is the creation of a new mark at the entering point. The characteristics of this mark, however, can be fully specified. Note that this action must be executed first because of the side effects the primary one might have on the initial context (e.g. it might change the window stacking order).

Primary action – It can be as simple as “go there”, “click” or “double-click”. It can consist in a user-defined combination of such basic interactions. But it can also be arbitrarily complex, although it should remain describable to the user in a simple way, either textual (e.g. “switch to the virtual desktop on the right”) or graphical.

Following action [optional] – Any action not occurring immediately on activation or taking place at another location. This action might involve user interaction, e.g. “wait for a user click and come back”. It that case, specific feedback is added to the standard pointer representation to show that the mark activation is still in progress.
The execution steps triggered by the activation of a mark can be summarized as follows, those between brackets being optional:

1. [create a mark (with a certain configuration) at the entering point]
2. [raise the window relevant to the mark, if any, and give it the keyboard focus]
3. execute the primary action
4. [execute the following action]

Graphical representation

Each mark is represented on-screen when in UIMarks mode by a disk shown at its location. The external and internal borders of the disk and its fill color are used to visualize the basic attributes of the mark (Figure 3).

![Figure 3: Visual representation of the attributes](image)

The primary action triggered by the mark is represented by drawings inside the disk while incidental actions are represented on the outside (Figure 4). The preceding creation of a new mark is denoted by a forward arrow starting on the left side of the disk. The new mark can be represented with extensive details at the end of this arrow. More compact representations are also possible if it uses standard attribute values and causes no incidental action. Following actions are denoted on the right side of the disk, a backward arrow indicating a pointer translation back to the entering point.

![Figure 4: Visual representation of the actions](image)

To facilitate its selection and activation in certain situations, a mark can be offset from its target. In this case, a small unselectable spot is left to indicate the target, connected to the mark by a line segment (Figure 5).

![Figure 5: UI mark example. An offseted, user-created, window-specific and permanent mark that clicks on the rectangle tool and sends the pointer back to the entering point.](image)

**INTERACTION DETAILS**

In this section, we describe the interactions used to create, select, configure, activate and delete marks. We focus our attention on the principles that guided the design of these interactions rather than implementation details, some of which will be described in the next section.

As previously stated, all UIMarks interactions take place in a specific mode outside which the marks are not visible. In this mode, a semi-transparent overlay is displayed on top of all windows to preserve the user’s context but visually differentiate the marks (Figure 1). The mode is associated with continuous pressure with the non-preferred hand on a specific key, Windows or Fn, for quick and easy access.

We use a slightly adapted version of the bubble cursor: the selected mark is scaled up by a factor of 5/3 to differentiate it, a small cross locates the entering point and the location of the pointer is indicated by a small dot. Leaving the mode activates the selected mark, if any. Note that since no button of the pointing device is used to enter the mode, select a mark and leave the mode, these operations can be executed during drag-and-drop operations (Figure 1). As the bubble cursor always captures one of the existing marks, two operations allow one to leave the mode without activating it: one that moves the pointer back to the entering point and the other that leaves it where it is. These operations are triggered by pressing the Escape and Enter keys, or clicking the Right and Middle buttons of the pointing device.

In addition to pointing-based selection using the bubble cursor, users can circulate between marks using the Tab key. By pressing Shift followed by an alphanumeric key, they can also bind that key to the selected mark: from that moment forward, pressing the key will leave the mode and activate the mark, and the bound symbol will be shown next to it.

A button click on an empty space with the pointing device creates a new mark with default attribute values. The newly created mark is automatically selected by virtue of the bubble cursor. Whenever a mark is selected, specific interaction techniques make it possible to (re)configure its target, lifetime and associated actions, the latter being chosen from a predefined set or incrementally specified as explained in the next section. The selected mark can be deleted by pressing the Backspace key. Marks can be moved by a simple drag-and-drop interaction and offsetted the same way using a keyboard modifier (Alt) or alternative button (Right).

We have experimented with the use of the system on a laptop with a touchpad and an external mouse. This combination of devices can be used in two ways. One can be specifically...
assigned to UIMarks and the other to conventional pointing. 
Note that although this supports implicit bimodal interaction, 
it requires good pointing skills with both hands. The devices 
can also be used in a bimanual rather than bimodal way: the 
mouse for selecting marks and the touchscreen as a mode trigger 
and to configure them, for example.

We acknowledge that configuring a mark can be time con-
suming. However, this should not happen too often and the 
power of the marks resides in their use: their fast selection 
and activation, which can trigger operations that go beyond 
simple pointing.

IMPLEMENTATION DETAILS
Implementing UIMarks in a window system requires the 
ability to observe, alter and generate input events, to dis-
play a semi-transparent overlay, and to determine target win-
dows and track them. Because Metisse[8] provides full con-
trol over its input and output mechanisms, implementing 
UIMarks on it was pretty straightforward. Specificities of 
this implementation include the automatic creation of holes 
in windows overlapping marked ones to show the marks in 
context and the availability of several primary actions re-
lated to virtual desktops. Mouse-based interactions support 
the incremental configuration of a mark: scrolling the mouse 
wheel over parts of its graphical representation allows one 
to specify each attribute or action separately by circulating 
between possible values.

Our OS X implementation required a somewhat unusual 
combination of public but unevenly documented APIs. It 
uses Quartz 2D to create an overlay window that covers the 
whole display space independently of the number of phys-
cal displays. It controls pointer acceleration through the HID 
System Manager and uses CoreFoundation’s distributed noti-
fication center to receive mark-related commands from exter-
nal applications written in any language (two lines of Python 
code are all it takes to request the creation of a mark, for 
example). Simple keyboard-based techniques allow users to 
configure the marks. When creating one, pressing the Shift 
modifier attaches it to the window underneath rather than the 
screen, and Ctrl makes it temporary rather than permanent. 
Pressing the t and l keys (for target and lifetime) later 
allows to toggle these settings for a selected mark. Pressing the 
Space key also allows to specify the preceding, primary and 
following actions of a selected mark by cycling through pre-
deﬁned combinations, e.g. “go there, click & come back”, 
“create a temporary mark & go there”. The window man-
agement services offered by Apple’s public APIs being quite 
limited1, we resorted to using the Accessibility API and the 
SetFrontProcess function to track windows and to raise 
them. This causes unwanted ﬂashes when we attach marks 
to windows as we need to temporarily hide the UIMarks overlay 
to query the Accessibility API, as well as occasional modi-
fications of the window stacking order.

Real use of UIMarks on the two platforms suggests that cer-
tain global aspects of it should be left conﬁgurable by the 
user. This includes the default target and lifetime for newly 
created marks, for example, as well as the deﬁnition of the 
temporary lifetime: limited in time, in the number of activ-
ations or both? The ﬁrst author’s system is conﬁgured so 
that marks are permanent and attached to windows by de-
fault. His temporary marks have a lifetime of 30 seconds but 
are automatically destroyed on activation. Moreover, when 
one is created, all other temporary marks within a distance of 
100 pixels are automatically destroyed. These settings help 
maintain a balance between facilitating back-and-forth inter-
actions and keeping the number of marks reasonable.

The impact of some window management operations is also 
difficult to decide. Although it seems reasonable to destroy 
the marks associated to a window being closed and to move 
the marks of a window being moved, what should be done 
when it is iconiﬁed, for example? Hiding the associated 
marks might seem the more coherent thing to do (it is the 
ﬁrst author’s personal choice), but keeping them visible pro-
vides a quick way of deiconifying the window on activation. 
Resizing and scrolling operations are also problematic. Our 
current implementations do nothing special about them (we 
will come back to that at the end of the paper).

EXPERIMENT
Informal use of UIMarks in real-life conditions suggested it 
might provide advantage over traditional pointing in certain 
situations. Before considering whether people would want 
to use our system and whether they would manage to do 
so, we wanted to clearly establish whether and when they 
could expect signiﬁcant performance beneﬁts. We thus de-
cided to evaluate UIMarks from a low-level perspective, in a 
controlled setting.

The use of a speciﬁc mode for target-aware pointing raises 
a complex question: under which conditions do the poten-
tial beneﬁts of the bubble cursor and mark-associated actions 
outweigh the mode-switching costs? Answering that ques-
tion is not easy because of the many cases to consider. The 
target can be directly under, close to or far from a mark, for 
example, and various actions can be thought of. The tasks to 
consider are composite ones mixing mode switching, bubble 
cursor pointing, pointer warping, traditional pointing and au-
tomated actions. They are different from the simple tasks 
used in previous work on the bubble cursor, for example, 
which makes the results of this work hardly usable in our 
case.

In the rest of this section, we report on a lab experiment that 
compared UIMarks to traditional pointing on abstract com-
posite tasks designed after common real-life scenarios. The 
main factor of the experiment was the technique (TECH): 
UIMarks (UIM) or traditional pointing (STD).

Scenarios and Tasks
The first scenario we considered is a simple target acquisition 
like clicking on a button, for example. In the case of STD, 
the user has to point at the target and click on it. We call this task 
SimpleClick. UIM supports this scenario through the following 
tasks:

(i) Simple - A mark that clicks is over the target. The user just 
needs to activate it.

1A special connection to the window server is notably required to manage 
other applications’ windows, which is exclusively maintained by the Dock.
(ii) SimpleClick - A mark is over the target. The user has to activate it, then click on the target.

(iii) Approach - There is no mark over the target. The user has to activate a mark close to it, then point at it and click.

The second scenario we considered is the selection of an item in a pull-down or pop-up menu. In the case of STD, it is a two-step process: the user has to point at a target and click on it, which reveals a second target within a short distance to acquire in the same way. We call this task Menu. A corresponding UIM task is:

(iv) Menu - A mark that clicks is on the first target. The user has to activate it, then point at the revealed target and click on it.

The UIM definition of Menu is very similar to Approach. Since we are mainly interested in pointing rather than target search, we can even consider the two tasks as quasi-equivalent if we assume the user knows where the second target will appear. Note however that when comparing the UIM and STD techniques, Approach with UIM should be compared to SimpleClick with STD, not Menu.

The third scenario we considered corresponds to return tasks consisting of a forward and a backward sub-tasks, the latter bringing the pointer back to the starting point or close to it. As we already said, after selecting a tool in a palette (SimpleClick) or an item in a menu (Menu), one indeed often returns to the original locus of interaction to resume it. In the case of STD, the backward sub-task can be seen as a new target acquisition: the user has to point at a target close to or at the starting point and click on it to delimit the task. We call this sub-task Back and combining it with the forward STD tasks gives two return tasks: SimpleClick|Back and Menu|Back. For UIM, we consider the following return tasks:

(v) Back - Follows a forward sub-task (i–iv) on a mark that creates a temporary mark at the entering point. The user comes back by activating the temporary mark and has then to click to delimit the backward sub-task.

(vi) AutoBack - Follows a forward sub-task (i–iv) on a mark that automatically moves the pointer back to the entering point. The user simply has to click.

(vii) BackApproach - same as Back except the final target is not at the entering point, where the temporary mark is, but close to it. The user has to activate the temporary mark, point at the final target and then click.

(viii) AutoBackApproach - same as AutoBack except the final target is again not at the entering point but close to it. The user has to point at the final target and click.

To summarize, for STD, we considered 2 forward tasks (SimpleClick and Menu), 1 backward task (Back) and thus 2 return tasks (SimpleClick|Back and Menu|Back). For UIM, we considered 4 forward tasks (i–iv), 4 backward tasks (v–viii) and thus 16 return tasks. Covering all these tasks is not possible in a single reasonably sized experiment. We therefore decided to focus on a few return tasks chosen so that all the forward and backward sub-tasks would be used at least once, taking into account the quasi-equivalence of Menu and Approach with UIM. Table 1 shows the tasks actually used for the experiment and Figure 6 illustrates the environment in which they were performed.

<table>
<thead>
<tr>
<th>UIM</th>
<th>STD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Menu</td>
<td>Back</td>
</tr>
<tr>
<td>Simple</td>
<td>AutoBackApproach</td>
</tr>
<tr>
<td>SimpleClick</td>
<td>AutoBack</td>
</tr>
</tbody>
</table>
| Table 1: Experiment tasks. Rows indicate which STD task should be used for comparison for each UIM task.

Figure 6: Users need to position the pointer inside the Start target to start a trial. They should wait until the target changes color, which indicates they can execute the task. Timing starts on their first action.

**Apparatus**

We conducted the experiment on a high-end workstation running X Window and a specifically designed OpenGL application. The display was a 30" LCD monitor with a resolution of 2560×1600 (100 dpi). We used a 600 dpi optical mouse making it possible to cross the whole screen without clutching using the default X Window acceleration function.

**Distances and Widths**

The tasks used in the experiment depended on a series of “factors”. Our main factors, besides the technique, were the size of the targets (SIZE) and the distance from the starting point to the first point (DIST). All the targets in a given instance of a task had the same size: they were either small (ST, 6 pixels) or normal (NT, 24 pixels). The distance to the first target was either small (SD, 512 pixels), medium (MD, 1280 pixels) or large (LD, 2048 pixels).

We fixed the distance to the secondary target of the Approach and Menu sub-tasks to 96 pixels in the movement direction. For UIM, we also had to specify the number and arrangement of on-screen marks besides those required for the tasks. To

---

2We use the | sign to delimit the forward and backward sub-tasks.
simplify the experiment and because UIMarks is rather intended for sparse layouts, we used a grid of extra marks arranged every 368 pixels starting from the location of the first target of the task and at least that distance away from the starting point.

Note that the DIST factor inevitably interacted with the grid of extra marks. When the distance was small (SD), there could be no mark between the starting point and the first target. When the distance was medium (MD), there were necessarily some marks between the starting point and the first target, and beyond it as well. For large distances (LD), we made sure there would be no mark accessible beyond the first target. This made it possible for users to take advantage of the screen edge to facilitate the acquisition of the target.

**Design**

Twelve unpaid volunteers participated in the experiment (2 female). They ranged in ages from 22 to 35 and were all experienced mouse users.

The experiment was a within-participant 2 (Tech) × 2 (Size) × 3 (Dist) design for each task. Participants always performed the STD tasks in the following order: SimpleClick|Back, then Menu|Back (simple to complex). For UIM, two orders were used: SimpleClick|BackApproach, Menu|Back, SimpleClick|AutoBack & Simple|AutoBackApproach; and the reverse order.

For each task, we blocked by Tech and then Dist, Size being randomly assigned. Participants either began with UIM or STD, and in the case of UIM, either with the first order or the reverse one. That gives four orders that we crossed with three orders of Dist obtained with a latin square to get the required twelve orders for the Tech×Dist crossing.

The experiment lasted approximately 45 minutes. Participants started each task with a training phase consisting of a series of 4 trials for each full condition. The measured phase then consisted of a series of 6 trials for each full condition. Participants thus performed $2 \times 2 \times 3 \times (4 + 6) = 120$ trials for each task, 72 being measured. Overall, a total of $12 \times 72 = 864$ measures were collected for each task ($12 \times 6 = 72$ for each full condition).

Participants were instructed to perform the tasks as quickly as possible and to minimize click errors. Note that since all the sub-tasks except Simple required participants to click on one or more target(s), click errors had to be corrected and thus impacted negatively on the measured time. UIMarks-specific errors in UIM tasks invalidated the trial, which the participant had then to redo. These errors include the selection of an incorrect mark, for example, or the use of the UIMarks mode for an incorrect number of times.

**Results**

Our results were analyzed considering the full factorial model

$$TMT \sim Tech \times Size \times Dist \times Random(Participant)$$

for each task, where TMT is the task movement time. As we were mainly interested in the potential benefit of UIMarks, we did not take into account invalid UIM trials, i.e. those during which participants had made a UIMarks-specific error (1.75% of them).

Table 2 shows the results of the ANOVA for the Tech factor and the Tech×Size and Tech×Dist interactions for TMT. The effect of Size and Dist are always significant ($p < 0.0001$) and are not included in that table. As one would expect based on the existing literature, participants were faster with the normal-sized targets than with the small ones and TMT increased as Dist increased (Figures 7 and 8). We thus turned our attention to the Tech×Size and Tech×Dist interactions: does Size impact the difference between UIM and STD? Does Dist have the same effect on UIM and STD?

The differences found between Dist conditions were lower for UIM than STD (see Figure 8 and the results of the post-hoc Tukey test in the last column of Table 2). However, the Tech×Dist interactions are not very strong compared to the Tech×Size ones: the Tech×Dist interaction is never strong enough to change the result of the global post-hoc test between UIM and STD for a given Dist (second column of Table 2), which is often the case with the Size factor. One can indeed see on Figure 7 and in the third column of Table 2 that UIM is especially efficient for small targets.

We now detail our results for each UIM task of the experiment, providing additional information to Table 2 and Figures 7 and 8 obtained by looking at intermediate times omitted for brevity:

**Menu|Back**: UIM is faster than STD (more than 25% faster for small targets). Intermediate times show that the same is true for Menu alone. This sub-task’s automated click probably provides the decisive advantage on small targets, accentuated by the precise pointer warping of Back.

Note that if we consider the UIM Menu sub-task as an Approach sub-task and thus compare the UIM Menu|Back task time to the STD SimpleClick|Back one (instead of STD Menu|Back), STD be-

<table>
<thead>
<tr>
<th>Box Contents</th>
<th>Tech</th>
<th>Tech×Size</th>
<th>Tech×Dist</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$F_{1,11}$ Tukey HSD</td>
<td>$p$</td>
<td>$F_{1,11}$ Tukey HSD</td>
</tr>
<tr>
<td>Menu</td>
<td>Back</td>
<td>43.3 &lt;0.0001</td>
<td>82.6 &lt;0.0001</td>
</tr>
<tr>
<td>Simple</td>
<td>AutoBack</td>
<td>131 &lt;0.0001</td>
<td>353 &lt;0.0001</td>
</tr>
<tr>
<td>Simple</td>
<td>AutoBack</td>
<td>131 &lt;0.0001</td>
<td>353 &lt;0.0001</td>
</tr>
<tr>
<td>Simple</td>
<td>AutoBack</td>
<td>131 &lt;0.0001</td>
<td>353 &lt;0.0001</td>
</tr>
</tbody>
</table>

Note, that the D factor inevitably interacted with the grid of extra marks. When the distance was small (SD), there could be no mark between the starting point and the first target. When the distance was medium (MD), there were necessarily some marks between the starting point and the first target, and beyond it as well. For large distances (LD), we made sure there would be no mark accessible beyond the first target. This made it possible for users to take advantage of the screen edge to facilitate the acquisition of the target.
For normal sized targets). In other words, using UIM probably accentuates this to provide the decisive advantage for normal-sized targets and cancel the benefit on the small ones. Intermediate times show that SimpleClick (more than 25% faster for normal sized targets). The backward approach seems again costly, even though it is automatically initiated. Using UIMarks to simply approach a target did not lead to any benefit and was even slower than directly acquiring it. UIMarks is not a general-purpose pointing facilitation system: the costs introduced by its use cannot be compensated unless one takes full advantage of the benefits it offers (e.g., precise pointer warping and programmability). However, real-life situations might be more favorable to it than this “pure pointing” experiment. As an example, a UIMarks approach performs the additional action of raising the window on which the mark was placed, which can be useful if it is fully or even partially overlapped. Note also that the UIM Back task brings the pointer back to the exact location where the user entered the UIMarks mode. We believe the benefit of this might be higher in real-life settings where that precise point might be of importance and difficult to locate. More generally, even in cases where precision does not matter, UIMarks might provide some benefit by simply reducing the number of possible choices for that location [22].

At the end of the experiment, participants were asked whether they felt they had been faster with UIMarks and whether they found it easy to use. They were asked to rate these feelings on a five point scale: very fast/easy (5), fast/easy (4), neither fast-slow/easy-difficult (3), slow/difficult (2) and very slow/difficult (1). All the ratings are ≥ 4 for speed and ≥ 3 for easiness with means of 4.08 and 3.92. This is really encouraging and in accordance with our quantitative results and the very low error rate in UIM tasks.

**Summary**

Difference in performance between UIM and STD depends on the nature of the forward and backward sub-tasks and on the SIZE factor. More precisely: (i) Approach and BackApproach put UIM at a disadvantage independently of the target size; (ii) Simple, SimpleClick, Menu, Back & AutoBackApproach are neutral or slightly advantage UIM for normal-sized targets and clearly advantage it for the small ones; and (iii) AutoBack provides a strong advantage to UIM even in the case of normal-sized targets.

Extrapolation based on an additive model

Given the results of the above experiment, we can estimate the movement times for all the tasks initially considered (cf. Scenarios and Tasks) by using a simple additive model: the movement time for a task A|B can be estimated by adding the intermediate times for the sub-tasks A and B measured through two experiment tasks A|C and D|B. Note however that such a simple additive model can only lead to approximate calculations of the movement times because of chunking and the fact that several tasks are often performed in parallel [17].

As an example, the movement time for Simple|AutoBack can be estimated from Simple|AutoBackApproach and SimpleClick|AutoBack. Doing so suggests a strong advantage for UIM on this task (more than 60% speed-up). The movement time similarly estimated for Approach|BackApproach suggests a 25% advantage for STD independent of the target size (this is presumably the worst case possible for UIM, none of the targets being under a mark). The movement time estimated for Approach|AutoBack suggests an advantage for UIM of at least 10% on both target sizes. The backward approach AutoBack approach again impedes performance, strong enough to reverse the result observed with SimpleClick|AutoBack on normal sized targets and cancel the benefit on the small ones.
sizes. AutoBack might thus be able to compensate the cost of a forward Approach.

To verify this additive model, we conducted a second lighter experiment similar to the first one using the following tasks: \textit{Simple}/BackApproach and \textit{Menu}/AutoBackApproach. These tasks were chosen for two reasons. First, the estimated differences on them between STD and UIM were not very important. Second, \textit{Simple} had been combined with an automated task in the first experiment and \textit{Menu} with a non-automated one, so we wanted to invert this choice.

We followed the exact same design and procedure as in the first experiment. The same 12 volunteers participated again. The experiment lasted approximately 25 minutes and we recorded only 1.1% of UIM errors. Figure 9 shows the results of this second experiment as well as the estimations obtained from the first one.

Concerning \textit{Menu}/AutoBackApproach, we found an effect of the technique ($F_{1,11} = 50.2, p < 0.0001$), UIM being faster than STD. We also found a \textit{Tech}$\times$\textit{Size} interaction ($F_{1,11} = 411, p < 0.0001$). While estimated values suggested no significant difference for normal sized targets, UIM was in fact significantly faster than STD for both target sizes (16% speed-up for the small ones, 8% for the normal sized ones). A close look at intermediate times shows that participants were slightly faster in experiment 2 than in experiment 1 on each sub-task. The performance improvement using UIM might thus be caused by some learning effect.

Concerning \textit{Simple}/BackApproach, we found a significant effect of the technique ($F_{1,11} = 9.02, p = 0.0120$) although it is not very strong compared to most of the others we have. We also found a \textit{Tech}$\times$\textit{Size} interaction ($F_{1,11} = 40.7, p < 0.0001$): UIM was significantly faster than STD for small targets (17% speed-up), STD being faster for the normal sized ones (3.7%) but the difference not being significant. This contrasts with estimates based on the first experiment that suggested no \textit{Tech} effect and a significant \textit{Tech}$\times$\textit{Size} interaction (UIM 6% faster than STD for small targets and STD 16% faster for the normal sized ones). Participants were actually faster than expected on this task with UIM. A close look at intermediate times shows that most of the differences between estimations and actual measurements are concentrated in the \textit{Simple} forward sub-task. Considering that we used Simple|AutoBackApproach to estimate it, it seems plausible that the automatic pointer warping it triggered slowed down the participants, which in turn suggests that our additive model might be too simplistic. Again, a \textit{UIM} learning effect might also have occurred.

**SUMMARY AND FUTURE WORK**

In this paper, we reported on the design, implementation and evaluation of UIMarks, a system that lets users specify on-screen targets and associated actions by means of a graphical marking language and provides a quick way to activate these marks.

While target-aware pointing techniques are known to be efficient in controlled settings, very few have been implemented in real systems. Doing so indeed poses two problems: one needs to identify the targets and also to integrate the technique with traditional pointing. With UIMarks, we propose a first solution to these problems for the bubble cursor. As demonstrated by our two implementations\(^3\), our solution is easily applicable to existing graphical environments without necessarily requiring the modification of applications. And as we explained, our solution is not conceptually tied to the bubble cursor. The first contribution of this work is thus the proposal of a framework that has the potential to leverage existing target-aware pointing facilitation techniques.

The second contribution of this work concerns the evaluation of the proposed solution. The comparative evaluation of traditional pointing and UIMarks in a lab experiment allowed us to assess its strengths and weaknesses in an abstract setting. Our results notably show that for normal-sized targets, the mode-switching costs outweigh the benefits provided by the sole bubble cursor. But our results also show that the costs introduced by UIMarks can largely be compensated when marks are adequately placed and one takes advantage of their programability. This is a crucial step towards a better comprehension of the strengths and weaknesses of the system in more complex settings. Future work on the evaluation will probably include contextual factors such as window management configurations or operations, multiple similar targets or huge distances imposing some cluthing.

Like most studies of pointing techniques, ours focused on efficiency. We did not consider other usability criteria such as the learnability or memorability of the system, or user satisfaction. Additional user feedback will be useful for improving and extending the system in these respects. It should help us decide, for example, how window-specific marks should behave in case of scrolling and resizing operations. Should marks be associated to inner controls rather than windows, or is the current way of (not) dealing with these operations acceptable? Accessibility APIs, pixel-based reverse engineering techniques or hybrid approaches could be used to provide the structural knowledge required to compare these alternatives [23, 9, 13].

The suitability of our representations and configuration techniques is another question we would like to study in the fu-

\(^3\)available from \url{http://insitu.lri.fr/uimarks/}
ture. Should we go on with a graphical language based on composable elements, or should we switch to a set of pre-determined icons? The answer is clearly related to the interaction techniques used to create and configure the marks. Some might prefer to “program” them, as is possible with our Metisse implementation, while others might prefer to just choose them in a predefined set, as on OS X. These questions will probably lead us to more general ones related to multi-attribute data specification and visualization.

ACKNOWLEDGEMENTS

We would like to thank Caroline Appert, Michel Beaudouin-Lafon, Renaud Blanch, Pierre Dragicevic, James Eagan, Nicolas Gaudron, Wendy Mackay, Emmanuel Pietriga and the anonymous reviewers for their feedback.

REFERENCES


