Spatial Control of Interactive Surfaces in an Augmented Environment

Stanislaw Borkowski, Julien Letessier, and James L. Crowley

Project PRIMA, Lab. GRAVIR-IMAG
INRIA Rhône-Alpes, 655, ave de l'Europe
38330 Montbonnot, France
{Stan.Borkowski, Julien.Letessier, James.Crowley}@inrialpes.fr

Abstract. New display technologies will enable designers to use every surface as a support for interaction with information technology. In this article, we describe techniques and tools for enabling efficient man-machine interaction in computer augmented multi-surface environments. We focus on explicit interaction, in which the user decides when and where to interact with the system. We present three interaction techniques using simple actuators: fingers, a laser pointer, and a rectangular piece of cardboard. We describe a graphical control interface constructed from an automatically generated and maintained environment model. We implement both the automatic model acquisition and the interaction techniques using a Steerable Camera-Projector (SCP) system.

1 Introduction

Surfaces dominate the physical world. Every object is confined in space by its surface. Surfaces are pervasive and play a predominant role in human perception of the environment. We believe that augmenting surfaces with information technology will act as an interaction modality easily adopted for a variety of tasks. In this article, we make a step towards making this a reality.

Current display technologies are based on planar surfaces [8, 17, 23]. Displays are usually treated as access points to a common information space, where users manipulate vast amounts of information with a common set of controls. Given recent developments in low-cost display technologies, the available interaction surface will continue to grow, forcing the migration of interfaces from a single, centralized screen to many, space-distributed interactive surfaces. New interaction tools that accommodate multiple distributed interaction surfaces will be required.

In this article, we address the problem of spatial control of an interactive display surface within an office or similar environment. In our approach, the user can choose any planar surface as a physical support for interaction. We use a steerable assembly composed of a camera and video projector to augment surfaces with interactive capabilities. We exploit our projection-based augmentation to attain three goals: *(a)* modelling the geometry of the environment by using it as a source of information, *(b)* creation of interactive surfaces anywhere in the scene, and *(c)* realisation of novel interaction techniques through augmentation of a handheld display surface.

In the following sections, we present the technical infrastructure for experimentation with multiple interactive surfaces in an office environment (Sections 3 and 4). We then discuss spatial control of application interfaces in Section 5. In Sections 6, 7 and 8 we describe three applications that enable explicit control of interface location. We illustrate interaction techniques with a single interaction surface controlled in a multi-surface environment, but we emphasize that they can be easily extended to the control of multiple independent interfaces controlled within a common space.

2 Camera-Projector Systems

Camera-projector systems are increasingly used in augmented environment systems [11, 13, 21]. Projecting images is a simple way of augmenting everyday objects and allows alteration of their appearance or function. Associating a video projector with a video camera offers an inexpensive means of making projected images interactive. However, standard video-projectors have small projection area which limits their flexibility in creating interaction spaces. We can achieve some steerability on a rigidly mounted projector by moving sub windows within the cone of projection [22], but extending or moving the display surface requires increasing the angle range of the projector beam. This requires adding more projectors, an expensive endeavor. An alternative is to use a steerable projector [2, 12]. This approach is becoming more attractive, due to a trend towards increasingly small and inexpensive video projectors.

Projection is an ecological (non-intrusive) way of augmenting the environment. Projection does not change the augmented object itself, only its appearance. Augmentation can be used to supplement the functionality of objects. In [12], ordinary artefacts such as walls, shelves, and cups are transformed into informative surfaces, but the original functionality of the objects does not change. The objects become physical supports for virtual functionalities. An example of object enhancement is presented in [1], where users can interact with both physical and virtual ink on a projection-augmented whiteboard.

While vision and projection-based interfaces meet most of the ergonomic requirements of HCI, they suffer from lack of robustness due to clutter and insufficiently developed methods for text input. People naturally avoid obstructing projected images, so occlusion is not a problem when camera and projector share the same viewpoint. As for the issue of text input on projected steerable interfaces, currently available projected keyboards like the Canesta Projection Keyboard [16] rely on hardware configuration, which excludes their use on arbitrary surfaces. Resolving this issue is important for development of projection-based interfaces, but it is outside the scope of this work.

3 The Steerable Camera-Projector System

In our experiments, we use a Steerable Projector-Camera (SCP) assembly (Figure 1). It enables us to experiment with multiple interactive surfaces in an office environment.

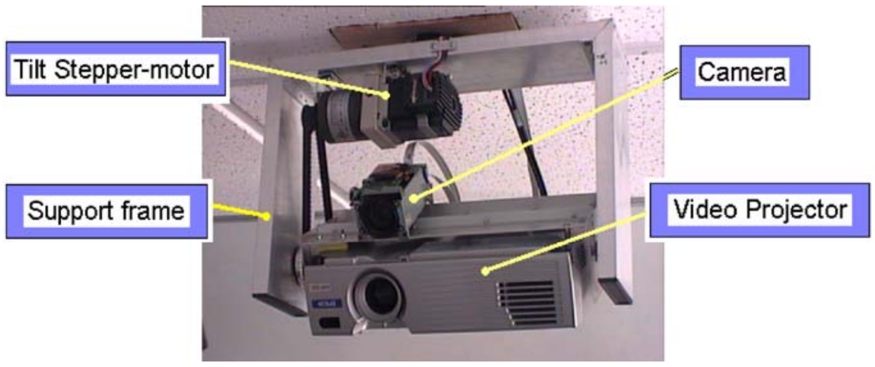


Fig. 1. The Steerable Camera-Projector pair.

The Steerable Camera-Projector (SCP) platform is a device that gives a video-projector and its associated camera two mechanical degrees of freedom, pan and tilt. Note that the projector-camera pair is mounted in such a way that the projected beam overlaps with the camera view. Association of the camera and projector creates a powerful actuator-sensor pair enabling observation of users' actions within the camera field of view. Endowed with the ability to modify the scene using projected light, projector-camera systems can be exploited as sensors (Section 5.2).

4 Experimental Laboratory Environment

The experiments described below are performed in our Augmented Meeting Environment (AME). The AME is an ordinary office equipped with ability to sense and act. The sensing infrastructure includes five steerable cameras, a fixed wide angle camera, and a microphone array. The wide angle camera has a field of view that covers the entire room. Steerable cameras are installed in each of the four corners of the room. A fifth steerable camera is centrally mounted in the room as part of the steerable camera-projector system (SCP).

Within the AME, we can define several surfaces suitable for supporting projected interfaces. Some of these are marked by white boundaries in Figure 2. These regions were detected by the SCP during an automatic off-line environmental model building phase described below (Section 5.2). Surfaces marked with dashed boundaries can be optionally calibrated and included in the generated environment model using the device described in Section 8.

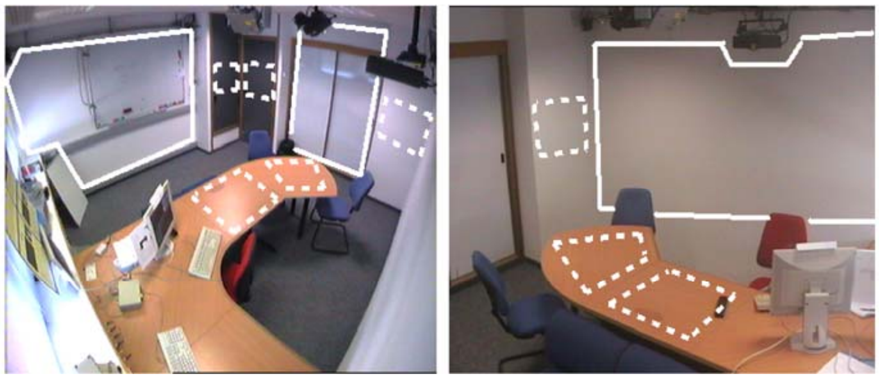


Fig. 2. Planar surfaces in the environment.

5 Spatial control of displays

Interaction combines action and perception. In an environment where users may interact with a multitude of services and input/output (IO) devices, both perception and interaction can be complex. We present a sample scenario in Section 5.1 and describe our approach to automatic environment model acquisition in Section 5.2, but first we discuss the relative merits of our approach to interaction within an augmented environment.

Explicit vs. Implicit. Over the last few years, several research groups have experimented with environments augmented with multiple display surfaces using various devices such as flat screens, whiteboards, video-projectors and steerable video-projectors [3, 8, 11, 13, 21, 23]. Most of these groups focuse on the integration of technical infrastructure into a coherent automated system, treating the problem of new methods for spatial control of interfaces as a secondary issue. Typically, the classic paradigm of drag and drop is used to manipulate application interfaces on a set of wall displays and table display [8]. In such systems, discontinuities in the transition between displays disrupt interaction and make direct adaptation of drag and drop difficult.

An alternative is to liberate the user by letting the system take control of interface location. In [11], the steerable display is automatically redirected to the surface most appropriate for the user. Assuming a sufficient environment model, the interface follows the user by jumping from one surface to another. However, this solution has disadvantages. For one, it requires continuous update of the environment model. More importantly, the system has to infer if the user wants to be followed or not. Such a degree of understanding of human activity is beyond the state of the art.

The authors in [3] combine automatic and explicit control. By default, the interface follows its owner in the augmented room. The user can also choose a display from a list. However, their approach assumes that the user is able to correctly identify the listed devices. Moreover, the method of passing back and forth from automatic to manual control mode is not clearly defined. In this work, we focus on developing interaction techniques that enable users to explicitly control the interface position in space.

Ecological vs. Emmbedded. In ubiquitous computing, panoply of small interconnected devices embedded in the environment or worn by the user are assumed to facilitate continuous and intuitive access to virtual information spaces and services. Many researchers follow this approach and investigate new interaction types based on sensors embedded in artifacts or worn by users [14, 18, 19]. Although embedding electronic devices leads to a number of efficient interface designs, in many circumstances it is unwise to assume that everyone will be equipped with the necessary technology. Moreover, as shown in [1, 3], one can obtain pervasive interfaces by embedding computational infrastructure in the environment instead. Our approach is to create new interaction modes and devices by augmenting the functionality of mundane artifacts without modifying their primary structure.

User-centric vs. Sensor-centric. Coutaz *et al.* [7] highlight the duality of interactive systems. We apply this duality to the analysis of environment models, extending our understanding of the perceived physical space. When building an environment model, the system typically generates a sensor-centric representation of the scene, but this abstraction is not necessarily comprehensible for the human actor. A common understanding of the environment requires translation of the model into a user-centric representation. Such an approach is presented in [3], where the authors introduce an interface for controlling lights in a room. Lamps are shown graphically on a 2D map of the environment, and the user chooses from the map which light to dim or to brighten. The problem is that modeling the real-world environment in order to generate and maintain a human-comprehensible representation of the space is a difficult and expensive task. Moreover, from the user's perspective, the physical location of the controlled devices is not as important as the effect of changing a device's state. Rather than showing the user a symbolic representation of the world, we enrich the sensor-centric model with contextual cues that facilitate mapping from an abstract model to the physical environment.

In summary, we impose the following constraints on multi-surface systems:

1. Users have control of the spatial distribution of applications when they have direct or actuator-mediated access to its interface.
2. Users can control the system both "as they come" without specific tools, and with the use of control devices.
3. The mapping between the symbolic representation of the controller interface and the real world is understandable by an unexperienced user, provided sufficient contextual cues.
4. The underlying sensor-centric model of the environment is generated and updated automatically.

In the following section, we illustrate our expectations of a multi-surface interaction system with a scenario.

5.1 Scenario

John, a professor in a research laboratory, is in his office preparing slides for a project meeting. As the project partners arrive, John hurryly moves the presentation he just finished to a large wall-mounted screen in the meeting room, choosing it from a list of available displays. The list contains almost twenty possible locations in his office and in the meeting room. John has no trouble making his selection because the name of each surface is beside its image as it appears in the scene.

During the meeting, John uses a wide screen to present slides about software architecture. John uses an ordinary laser-pointer to highlight important elements in the slide. The slides are also projected onto a whiteboard so that John can make notes directly on them by drawing on the white board with an ink pen. On command he can record his notations in a new slide that combines his notations with the projected material. At one point, John sees that there is not enough free space on the white board, so he decides to move the projected slide to free some space for notes. He "double-blinks" the laser-pointer on the image, so that the image follows the laser dot.

While the project participants discuss the problem at hand, it becomes apparent that it is useful to split the meeting in three sub-workgroups. John takes one of the groups to his office. From the display list, John chooses the largest surface in his office. He sends the slide to this surface. A second group gathers around the desk in the meeting room. John sends the relevant slide from the wide screen to the desk with the use of a laser-pointer. The third smaller group decides to work in the back of the meeting room. Since there is no display, they take a cardboard onto which they transfer their application interface. They continue their work by interacting directly with the interface projected on the portable screen.

5.2 Environment modeling and image rectification

In our approach to human-computer interaction, it is critical that the system is aware of its working space in order to provide appropriate feedback to the user. The graphical user interfaces enabling explicit control of the display location (Sections 6 and 7) are generated based on the environment model. They contain information facilitating mapping of the virtual sensor-centric model to the physical space.

Although 3D environment models have many advantages for applications involving the use of steerable interfaces, they are difficult to create and maintain. One often makes the simplifying assumption that they exist beforehand and do not change over time [3, 11]. Instead, we propose automatic acquisition of a 2D environment model. The model consists of two layers: *(a)* a labelled 2D map of the environment in the SCP's spherical coordinate system and *(b)* a database containing the acquired characteristics for each detected planar surface. Our environment model directly reflects the available sensor capabilities of our AME.

To acquire the model of the environment, we exploit the SCP's ability to modify the environment by projecting and controlling images in the scene. Model acquisition consists of two phases: first, planar surfaces are detected and labelled with unique identifiers, and second, an image of each planar surface is captured and stored in the model database. In the second phase, the system projects a sample image on each planar surface detected in the environment model and takes a shot of the scene with the camera that has the projected image in its field of view. The images show the available interaction surfaces together with their surroundings. They are used later-on to provide users with contextual information which facilitates the mapping between the sensor-centric environment model and the physical world.

In order to customize the system, users should have the ability to supplement or replace the images in the model database with other data structures (e.g. text labels or video sequences). Using an interaction tool described in Section 8, the model is updated each time a new planar surface is defined in the environment.

Detection of planar surfaces. Most existing methods for projector-screen geometry acquisition provide a 3D model of the screen [5, 25]. However, such methods require the use of a calibrated projector-camera pair separated by a significant base distance. Thus, they are not suitable for our laboratory. In our system, we employ a variation of the method described in [2]. We use a steerable projector and a distant non-calibrated video camera to detect and estimate orientation of planar surfaces in the scene.The orientation of a surface with respect to the beamer is used to calculate a pre-warp that is applied to the projected image. The pre-warp compensates for oblique projective deformations caused by the non-orthogonality of the projector's optical axis relative to the screen surface. Note that the pre-warped image uses only a subset of the available pixels. When images are projected at extreme angles, the effective resolution can drop to a fraction of the projector's nominal resolution. This implies the need for an interface layout adaptation mechanism, that takes into account readability of the interface at a given projector-screen configuration. Adaptation of interfaces is a vast research problem and is not treated in this work.

6 Listing the available resources

In this section, we present a menu-like automatically generated interface enabling a user to choose the location of the display or application interface.

Pop-up and scroll-down menus are known in desktop-based interfaces for at least twenty years. Since planar surfaces in the environment can be seen as potential resources, it is natural to use a menu as a means for choosing a location for the interface.

Together with the projected image as application interface, we project an interactive button that is sensitive to touch-like movements of the user's fingertip. When the user touches the button, a list of available screen locations appears (Figure 3).

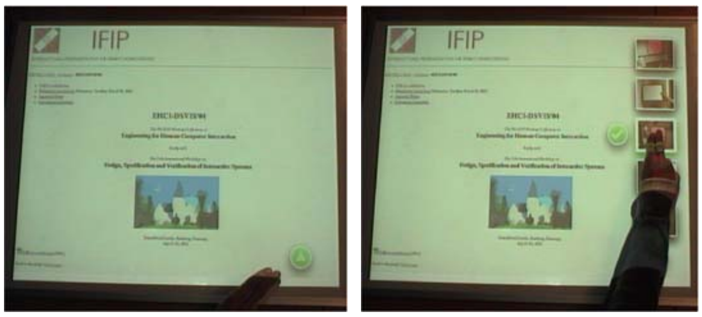


Fig. 3. Interacting with a list of displays (envisionment).

As mentioned in Section 5, we enhance the controller interface with cues that help map the interface elements to the physical world. Therefore, we present each list item as an image taken by one of the cameras installed in the room. We automatically generate the list based on images taken during the off-line model building process (Section 5.2). The images show the available interaction surfaces together with their surroundings. The user chooses a new location for the interface by passing a finger over a corresponding image. Note that one of the images shows a white cardboard, which is an interaction tool described in Section 8. In order to avoid accidental selection, we include a "confirm" button. The user cancels the interaction with the controller application by touching the initialization button again. The list also disappears if there is no interaction for a fixed period of time.

One can easily extend our image-based approach for providing contextual cues from interface control to general control of visual-output devices. For example, instead of showing a map of controllable lamps in a room, we can display a series of short sequences showing the corresponding parts of the room under changing light settings. This allows the user to visualize the effects of interaction with the system before actual execution.

6.1 Vision-based touch detection

Using vision as an user-input device for a projected interface is an elegant solution because *(a)* it allows for direct manipulation, i.e. no intermediary pointing device is used, and *(b)* it is ecological – no intrusive user equipment is required, and bare-hand interaction is possible. This approach has been validated by a number of research projects, for instance the DigitalDesk [24], the Magic Table [1] or the Tele-Graffiti application [20].

Existing vision-based interactive systems track the acting member (finger, hand, or head) and produce actions (visual feedback and/or system side effects) based on recognized gestures. One drawback is that a tracking system can only detect apparition, movement and disparition events, but no "action" event comparable to the mouse-click in conventional user interfaces, because a finger tap cannot be detected by a vision system alone [24]. In vision-based UIs, triggering a UI feature (e.g. a button widget) is usually performed by holding (or “dwelling”) the actuator (e.g. over the widget) [1, 20].

Various authors have tried different approaches to finger tracking, such as correlation tracking, model-based contour tracking, foreground segmentation and shape filtering, etc. While many of these are successful in constrained setups, they perform poorly for a projected UI or in unconstrained environments. Furthermore, they are computationally expensive. Since our requirements are limited to detecting fingers dwelling over button-style UI elements, we don’t require a full-fledged tracker.

Approach. We implement an appearance-based method based on monitoring the perceived luminance over UI widgets. Consider the two areas depicted in Figure 4.

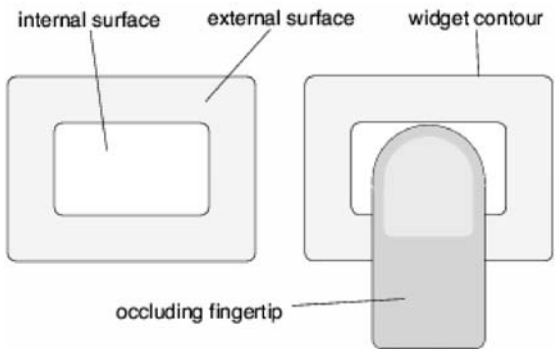


Fig. 4. Surfaces defined to detect touch-like gestures over a widget.

The inner region is assumed to roughly be of the same size as a finger. We denote $L_o(t)$ and $L_i(t)$ to be the average luminance over the outer and inner surface at time t, and

$$\Delta L(t) := \left| L_o(t) - L_i(t) \right|$$

Assuming that the observed widget has a reasonably uniform luminance, ΔL is close to zero at rest, and is high when a finger hovers over the widget. We define the threshold θ to be twice the median value of $\Delta L(t)$ over time when the widget is not occluded. Given the measured values of $\Delta L(t)$, the system generates the event e_0 (or e_1), at each discrete timestep t when $\Delta L(t)<\theta$ (or $\geq\theta$). These events are fed into a simple state machine that generates a *Touch* event after a dwell delay τ (Figure 5).

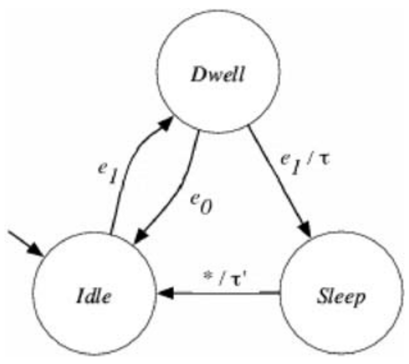


Fig. 5. The finite state machine used to process widget events.

We define two delays: τ to prevent false alarms (the *Dwell* → *Sleep* transition is only triggered after this delay), and τ' to avoid unwanted repetitive triggering (the *Sleep* → *Idle* transition is only triggered after this delay). A *Touch* event is issued whenever entering the *Sleep* state. τ and τ' are chosen equal to 200 ms. This technique achieves robustness against full occlusion of the UI component (e.g. by the user's hand or arm), since such occlusions cause ΔL to remain under the chosen threshold.

Experimental results. Our relatively simple approach provides good results because it is robust to changes in lighting conditions (it is a memory-less process), and occlusions (due to the dynamic nature of event generation and area-based filtering). Furthermore, it is implemented as a real-time process (it runs at camera frequency with less than 50 ms latency), although its cost scales linearly with the number of widgets to monitor.

An example application implemented with our "Sensitive Widgets" approach is shown in Figure 6. The minimal user interface consists of four projected buttons that can be "pressed" i.e. partially occluded with one or more fingers, to navigate through a slideshow.

Using this prototype, we confirm that our approach is robust to arbitrary changes in lighting conditions (the interface remains active during the changes) and full occlusion of widgets.

Integration. We integrate "Sensitive widgets" into a Tk application in an object oriented fashion: they are created and behave as usual Tk widgets. The implementation completely hides the underlying vision process, and provides activation (*Click*) events without uncertainty.

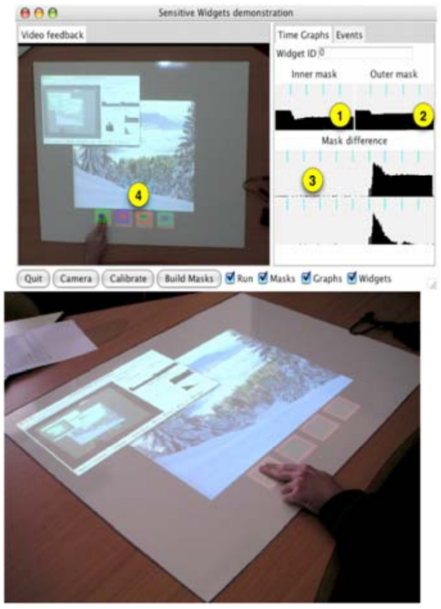


Fig. 6. The "Sensitive Widgets" demonstration interface. *Left:* The graphs exhibit the evolution of a variable in time: *(1)* $L_i(t)$; *(2)* $L_o(t)$; *(3)* $\Delta L(t)$. Notice the high value of ΔL while the user occludes the first widget. The video feedback *(4)* also displays the widget masks as transparent overlays. *Right:* The application interface as seen by the user (the control panel wasn't hidden), in unconstrained lighting conditions (here, natural light).

7 Laser-based control

Having a large display or several display locations demands methods to enable interaction from a distance. Since pointing with a laser is intuitive, many researchers have investigated how to use laser-pointers to interact with computers [4, 9]. Most of them try to translate laser-pointer movements to events similar to those generated by a mouse. According to Myers *et al.* [10], pointing at small objects with a laser is much slower than with standard pointing devices, and less precise compared to physical pointing. On the other hand, pointing with a hand or finger has a very limited range. Standard pointing devices like the mouse or trackball provide interaction techniques that are suitable for a single screen setup, even if the screen is large, but they cannot by adapted to multiple display environments with complex geometry. Hand pointing from a distance provides interesting results [6], but the pointing resolution is too low to be usable, and stereoscopic vision is required.

In our system, we use laser-based interaction exclusively to redirect the beamer (SCP) from one surface to another. This corresponds to moving an application interface to a different location in the scene. Users are free to use their laser pointers in a natural fashion. They can point at anything in the room, including the projected images. The system does not respond unless a user makes an explicit sign.

In our application, interaction is activated with a double sequence of switching the laser on and off while pointing to roughly the same spot on the projected image. If after this sign the laser point appears on the screen and does not move for a short time, the control interface is projected. During the laser point dwell delay we estimate hand jitter in order to scale the controller interface appropriately, as explained below.

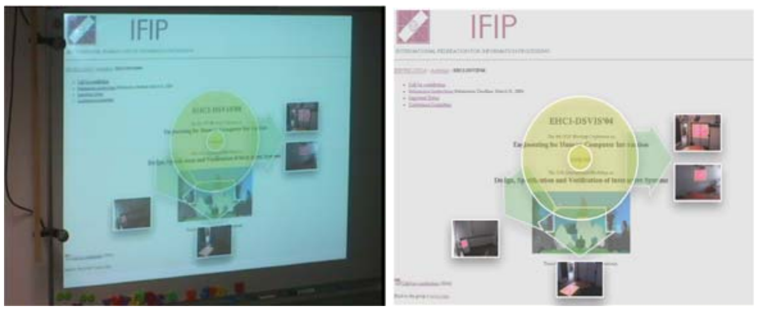


Fig. 7. Laser-based control interface (envisonment)

The interface shown in Figure 7 is a semi-transparent disc with arrows and thumbnail images. The arrows point to physical locations of the available displays in the environment. Similar to the menu-like controller application, the images placed at the end of each arrow are taken from the environment model. They present each display surface as it appears in the scene. The size of the images is a function of the measured laser point jitter. So is the size of the small internal disc representing the dead-zone, in which the laser dot can stay without reaction of the system. The controller interface is semi-transparent in order to avoid breaking users' interaction with the application, in case of a false initialization.

In order to avoid unwanted system reaction, the interface is not active when it appears. To activate it, the user has to explicitly place and keep the laser dot for a short time in any of the GUI's elements (arrow, image or disc). As the user moves the laser point within the yellow outer disc, the system starts to move the interface following the laser point with the center of the disc. This movement is limited to the area of the current display surface. Interface movement is slow for proper user control. When the laser goes outside the yellow disc or enters an arrow, movement halts. The user can then place the laser dot in the image of choice. As the laser point enters an image, the application interface immediately moves across the room to the corresponding surface. The controller interface does not appear on the newly chosen display unless it is again activated. At any time during the interaction process, the user can cancel the interaction by simply switching off the laser pointer.

7.1 Laser tracking with a camera

Several authors have investigated interaction from a distance using a laser pointer [4, 9,10].

Once we achieve geometric calibration of the camera and projector fields of view, detection and tracking the laser pointer dot is a trivial vision problem. Since laser light has a high intensity, a laser spot is the only visible blob on an image captured with a low-gain camera. The detection is then obtained by thresholding the intensity image and determining the barycentre of the connected component. Robustness against false alarms can be achieved by filtering out connected components that have aberrant areas.
As for other tracking systems, the output is a flow of *appear*, *motion* and *disappear* events with corresponding image-space positions. We achieve increased robustness by:

- generating *appear* events only once the dot has been consistently detected over several frames (e.g. 5 frames at 30Hz);
- similarly delaying the generation of *disappear* events.

We are not concerned by varying lighting conditions and shadowing because the camera is set to low gain. Occlusion, on the other hand, is an issue because an object passing through the laser beam causes erratic detections, which should be filtered out.

The overall simplicity of the vision process allows it to be implemented at camera rate (ca. 50Hz) with low latency (ca. 10ms processing time). Thus, it fulfils closed-loop human-computer interaction constraints.

8 A novel user-interface: the PDS

Exploiting robust vision-based tracking of an ordinary cardboard using an SCP unit [2] enables the use of a Portable Display Surface (PDS). We use the SCP to maintain a projected image onto the hand-held screen (PDS), automatically correcting for 3D translations and rotations of the screen.

We extend the concept of the PDS by integrating it in our AME system. As described in the example scenario (Section 5.1), the PDS can be used as a portable physical support for a projected interface. This mode of use is a variation of the “pick and drop” paradigm introduced in [15]. From the system point of view, the only difference between a planar surface in the environment and the PDS is its mobility and the image-correction matrix, so we can project the same interactive-widget-based interface on both static and portable surfaces. In practice, we have to take in account the limits of the image resolution available on the PDS surface.

The portability of this device creates two additional roles for the PDS in the AME system. It can serve as a means for explicit control of the display location and as a tool enabling the user to extend the environment model to surfaces which are not detected during the offline model acquisition procedure. Actually, the two modes are closely coupled and the extension of the environment model is transparent for the user.

To initialize the PDS, the user has to choose the corresponding item in the GUIs described in previous sections. Then, the SCP projects a rectangular region into which the user has to put the cardboard screen. If no rectangular object appears in this region within a fixed delay, the system falls back to its previous state. When the PDS is detected in the projected initialization region, the system transfers the display to the PDS and starts the tracking algorithm. The user can then move in the environment with the interface projected on the PDS. To stop the tracking algorithm, the user touches the "Freeze" widget projected on the PDS. The location of the PDS together with the corresponding pre-warp matrix is thus added to the environment model as new screen surface. This mechanism allows the system to dynamically update the model.

9 Conclusions

The emergence of spatially low-constrained working environments calls for new interaction concepts. This paper illustrates the issue of spatial control of a display in a multiple interactive-surface environment. We use steerable camera-projector assembly to display an interface and to move it in the scene. The projector-camera pair is also used as an actuator-sensor system enabling automatic construction of a sensor-centric environment model. We present three applications enabling convenient control of the display location in the environment. The applications are based on interactions using simple actuators: fingers, a laser pointer and a hand-held cardboard.

We impose a strong relation between the controller application interface and the physical world. The graphical interfaces are derived from the environment model, allowing the user to map the interface elements to the corresponding real-world objects. Our next development step is to couple controller applications with standard operating systems infrastructure.

Acknowledgments

This work has been partially funded by the European project FAME (IST-2000-28323), the FGnet working group (IST-2000-26434), and the RNTL/Proact ContAct project.

References

1. F. Bérard. The magic table: Computer-vision based augmentation of a whiteboard for creative meetings. In *Proceedings of the ICCV Workshop on Projector-Camera Systems*. IEEE Computer Society Press, 2003.
2. S. Borkowski, O. Riff, and J. L. Crowley. Projecting rectified images in an augmented environment. In *Proceedings of the ICCV Workshop on Projector-Camera Systems*. IEEE Computer Society Press, 2003.

3. B. Brumitt, B. Meyers, J. Krumm, A. Kern, and S. Shafer. Easyliving: Technologies for intelligent environments. In *Proceedings of Handheld and Ubiquitous Computing*, September 2000.
4. J. Davis and X. Chen. Lumipoint: Multi-user laser-based interaction on large tiled displays. *Displays*, 23(5), 2002.
5. R. Raskar et al. iLamps: Geometrically aware and self-configuring projectors. In *Appears ACM SIGGRAPH 2003 Conference Proceedings*.
6. Yi-Ping Hungy, Yao-Strong Yangz, Yong-Sheng Cheny, Ing-Bor Hsiehz, and Chiou-Shann Fuhz. Free-hand pointer by use of an active stereo vision system. In *Proceedings of the 14th International Conference on Pattern Recognition (ICPR'98)*, volume 2, pages 1244–1246, August 1998.
7. J.Coutaz, C.Lachenal, and S. Dupuy-Chessa. Ontology for multi-surface interaction. In *Proceedings of the ninth International Conference on Human-Computer Interaction (Interact'2003)*, 2003.
8. B. Johanson, G. Hutchins, T. Winograd, and M. Stone. Pointright: Experience with flexible input redirection in interactive workspaces. *Proceedings of UIST-2002*, 2002.
9. D. R. Olsen Jr and T. Nielsen. Laser pointer interaction. In *ACM CHI'2001 Conference Proceedings: Human Factors in Computing Systems. Seattle, WA*, 2001.
10. B. A. Meyers, R. Bhatnagar, J. Nichols, C.H. Peck, D. Kong, R. Miller, and A.C. Long. Interacting at a distance: measuring the performance of laser pointers and other devices. In *Proceedings of the SIGCHI conference on Human factors in computing systems: Changing our world, changing ourselves*. ACM Press New York, NY, USA, April 2002.
11. G. Pingali, C. Pinhanez, A. Levas, R. Kjeldsen, M. Podlaseck, H. Chen, and N. Sukaviriya. Steerable interfaces for pervasive computing spaces. In *Proceedings of IEEE International Conference on Pervasive Computing and Communications - PerCom'03*, March 2003.
12. C. Pinhanez. The everywhere displays projector: A device to create ubiquitous graphical interfaces. In *Proceedings of Ubiquitous Computing 2001 Conference*, September 2001.
13. R. Raskar, G. Welch, M. Cutts, A. Lake, L. Stesin, and H. Fuchs. The office of the future: A unified approach to image-based modeling and spatially immersive displays. In *Proceedings of the ACM SIGGRAPH'98 Conference*.
14. J. Rekimoto. Multiple-computer user interfaces: "beyond the desktop" direct manipulation environments. In *ACM CHI2000 Video Proceedings*, 2000.
15. J. Rekimoto and M. Saitoh. Augmented surfaces: A spatially continuous workspace for hybrid computing environments. In *Proceedings of CHI'99, pp.378-385*, 1999.
16. Helena Roeber, John Bacus, and Carlo Tomasi. Typing in thin air: the canesta projection keyboard - a new method of interaction with electronic devices. In *CHI '03 extended abstracts on Human factors in computing systems*, pages 712–713. ACM Press, 2003.
17. N. A. Streitz, J. Geißler, T. Holmer, S. Konomi, C. Müller-Tomfelde, W. Reischl, P. Rexroth, P. Seitz, and R. Steinmetz. i-land: An interactive landscape for creativitiy and innovation. *ACM Conference on Human Factors in Computing Systems*, 1999.
18. N. A. Streitz, C. Röcker, Th. Prante, R. Stenzel, and D. van Alphen. Situated interaction with ambient information: Facilitating awareness and communication in ubiquitous work environments. In *Tenth International Conference on Human-Computer Interaction*, June 2003.
19. Zs. Szalavári and M. Gervautz. The personal interaction panel - a two-handed interface for augmented reality. In *Proceedings of EUROGRAPHICS'97, Budapest, Hungary*, September 1997.
20. N. Takao, J. Shi, , and S. Baker. Tele-graffiti: A camera-projector based remote sketching system with hand-based user interface and automatic session summarization. *International Journal of Computer Vision*, 53(2):115–133, July 2003.

21. J. Underkofflerand B. Ullmer and H. Ishii. Emancipated pixels: Real-world graphics in the luminous room. In *Proceedings of ACM SIGGRAPH*, pages 385–392, 1999.
22. F. Vernier, N. Lesh, and C. Shen. Visualization techniques for circular tabletop interfaces. In *Advanced Visual Interfaces*, 2002.
23. S.A. Voida, E.D. Mynatt, B. MacIntyre, and G. Corso. Integrating virtual and physical context to support knowledge workers. In *Proceedings of Pervasive Computing Conference*. IEEE Computer Society Press, 2002.
24. P. Wellner. The digitaldesk calculator: Tactile manipulation on a desk top display. In *ACM Symposium on User Interface Software and Technology*, pages 27–33, 1991.
25. R. Yang and G. Welch. Automatic and continuous projector display surface calibration using every-day imagery. In *CECG'01*.

Discussion

[Joaquim Jorge] Could you give some details on the finger tracking. Do you use color information?

[Stanislaw Borkowski] We do not track fingers, but detect their presence over projected buttons. The detection is based on measurements of the perceived luminance over a widget. Our projected widgets are robust to accidental full-occlusions and change of ambient light conditions. However, since we do not use any background model, our widgets work less reliably if they are projected on surfaces with color intensity that is similar to the color of user's fingers.

[Nick Graham] You said you want to perform user studies to validate your approach. What is the hypothesis you wish to validate?

[Stanislaw Borkowski] What we would like to validate is our claim that a sensor-centric environment model enhanced with contextual cues is easier to interpret by humans than a symbolic representation of the environment (such as a 2D map).

[Fabio Paterno] Why don't you use hand pointing instead of laser pointing for display control?

[Stanislaw Borkowski] There are two reasons: First, laser pointing is more precise, which is important for fine tuning the display position. Second, is the issue of privacy. Using hand pointing requires constant observation of the user, and I am not sure whether everyone would feel comfortable with that.

[Fabio] there are so many cameras!

[Stanislaw Borkowski] Yes, but when using our system the user is not necessary aware of presence of those cameras. In contrary, using hand-pointing interaction user would have to make some kind of a "waving" sign to one of the cameras to initialize the interaction.

[Rick Kazman] Your interaction is relatively impoverished. Have you considered integrating voice command to give richer interaction possibilities?

[Stanislaw Borkowski] Not really, because we would encounter the problem of how to verbally explain to the system our requests.

[Rick Kazman] I was thinking more of using voice to augment the interaction, to pass you into specific modes for example, or to enable multimodal interaction (e.g. "put that there").

[Stanislaw Borkowski] Yes, that is a good idea. We should look into it. Right now we need to add a button to the interface which might obscure part of the interface. So in that case voice could be useful.

[Michael Harrison] What would be a good application for this type of system?

[Stanislaw Borkowski] An example could be a project-meeting, which has to split into to working subgroups. They could send a copy of their presentation on which they work to another surface. This surface could be even in a different room. Another application could be for a collaborative document editing. In this situation users could pass the UI between each other and thus pass the leadership of the group. This could help to structure the work of the group.

[Philippe Palanque] Do you have an interaction technique for setting the focus of the video projector?

[Stanislaw Borkowski] The focus should be set automatically, so there is no need for such interaction. We plan to feed the focus lens of the projector to the auto-focus of the camera mounted on the SCP.

[Helmut Stiegler] You don't need perfectly planar surfaces. The surface becomes "planer" by "augmentation".

[Stanislaw Borkowski] That is true, but it would become more complicated to implement the same features on non-planar interfaces. The problem of projection on non-planar surfaces is that the appearance of the projected image depends on the point of view.

[Eric Schol] How is ambiguity solved in touching multiple projected buttons at the same time? Such situation appears when you reach to a button that is farther from the user than some other buttons.

[Stanislaw Borkowski] The accidental occlusion of buttons that are close to the user is not a problem since our widgets "react" only on partial occlusion.

[Pierre Dragicevic] Did you think about using color information during model acquisition phase? This might be useful for choosing the support-surface for the screen, only from surfaces that are light-colored. You could also use such information to correct colors of the projected image.

[Stanislaw Borkowski] Yes, of course I though about it. This is an important feature of surfaces, since the color of the surface on which we project can influence the appearance of the projection. At this stage of development we did not really addressed this issue yet.

[Joerg Roth] Usually users press buttons quickly with a certain force. Your system requires a finger to reside in the button area for a certain time. Get users used to this different kind of interacting with a button?

[Stanislaw Borkowski] To answer your question I would have to perform user studies on this subject. From my experience and the experience of my colleagues who tried our system, using projected buttons is quite natural and easy. We did not encounter problems with using projected buttons.

Manipulating Vibro-Tactile Sequences on Mobile PC

Grigori Evreinov, Tatiana Evreinova and Roope Raisamo

TAUCHI Computer-Human Interaction Unit
Department of Computer Sciences
FIN-33014 University of Tampere, Finland
+358 3 215 8549
{grse, e_tg, rr}@cs.uta.fi

Abstract. Tactile memory is the crucial factor in coding and transfer of the semantic information through a single vibrator. While some simulators can produce strong vibro-tactile sensations, discrimination of several tactile patterns can remain quite poor. Currently used actuators, such as shaking motor, have also technological and methodological restrictions. We designed a vibro-tactile pen and software to create tactons and semantic sequences of vibro-tactile patterns on mobile devices (iPAQ pocket PC). We proposed special games and techniques to simplify learning and manipulating vibro-tactile patterns. The technique for manipulating vibro-tactile sequences is based on gesture recognition and spatial-temporal mapping for imaging vibro-tactile signals. After training, the tactons could be used as awareness cues or the system of non-verbal communication signals.

1 Introduction

Many researchers suppose that the dynamic range for the tactile analyzer is narrow in comparison to visual and auditory ones. This fact is explained by the complex interactions between vibro-tactile stimuli, which are in spatial-temporary affinity. This has resulted in a fairly conservative approach to the design of the tactile display techniques. However, some physiological studies [1] have shown that a number of possible “descriptions” (states) of an afferent flow during stimulation of the tactile receptors tend to have a greater amount of the definite levels than it was previously observed, that is more than 125. The restrictions of the human touch mostly depend on imaging techniques used, that is, spatial-temporal mapping and parameters of the input signals. As opposed to static spatial coding such as Braille or tactile diagrams, tactile memory is the crucial factor affecting perception of the dynamical signals similar to Vibratese language [7], [9].

Many different kinds of devices with embedded vibro-tactile actuators have appeared during the last two years. There is a stable interest to use vibration in games including small-size wearable devices like personal digital assistants and phones [2], [3], [14]. The combination of small size and low weight, low power consumption and noise, and human ability to feel vibration when the hearing and vision occupied by other tasks or have some lacks, makes vibration actuators ideal for mobile applications [4], [10].

On the other hand, the absence of the tactile markers makes almost impossible for visually impaired users interaction with touchscreen. Visual imaging is dominant for touchscreen and requires a definite size of virtual buttons or widgets to directly manipulate them by the finger. Among recent projects, it is necessary to mention the works of Nashel and Razzaque [11], Fukumoto and Sugimura [6] and Poupyrev et al [12]. The authors propose using different kinds of the small actuators such as piezoceramic bending motor [6], [12] or shaking motor [11] attached to a touch panel or mounted on PDA.

If the actuator is placed just under the touch panel, the vibration should be sensed directly at the fingertip. However, fingertip interaction has a limited contact duration, as the finger occupies an essential space for imaging. In a case of blind finger manipulations, a gesture technique becomes more efficient than absolute pointing when making use of the specific layout of software buttons. A small touch space and irregular spreading of vibration across touchscreen require another solution. If the actuator is placed on the backside of the mobile device, vibration could be sensed at the palm holding the unit. In this case, the mass of the PDA is crucial and impacts onto spectrum of playback signals [4], [6].

From time to time vibro-tactile feedback has been added to a pen input device [13]. We have also implemented several prototypes of the pen having embedded shaking motor and the solenoid-type actuator. However, shaking motor has a better ratio of the torque to power consumption in a range of 3 – 500 Hz than a solenoid-type actuator. The vibro-tactile pen certainly has the following benefits:

– the contact with the fingers is permanent and has more touch surface, as a rule, two fingertips tightly coupled to the pen;
– the pen has smaller weight and vibration is easily spread along this unit, it provides the user with a reliable feeling of different frequencies;
– the construction of the pen is flexible and admits installation of several actuators which have a local power source;
– the connection to mobile unit can be provided through a serial port or Bluetooth, that is, the main unit does not require any modification.

Finally, finger grasping provides a better precision compared with hand grasping [5]. Based on vibro-tactile pen we developed a special technique for imaging and intuitive interacting through vibration patterns. Simple games allow to facilitate learning or usability testing of the system of the tactons that might be used like awareness cues or non-verbal communication signals.

2 Vibro-Tactile Pen

The prototype of vibro-tactile pen consists of a miniature DC motor with a stopped rotor (shaking motor), electronic switch (NDS9959 MOSFET) and battery having the voltage of 3 V. It is possible to use internal battery of iPAQ, as an effective current can be restricted to 300 mA at 6 V. Both the general view and some internal design features of the pen are shown in Fig. 1.

There are only two control commands to start and stop the motor rotation. Therefore, to shape an appropriate vibration pattern, we need to combine the pulses of the current and the pauses with definite duration. Duration of the pulses can slightly change the power of the mechanical moment (a torque). The frequency will mostly be determined by duration of the pauses.

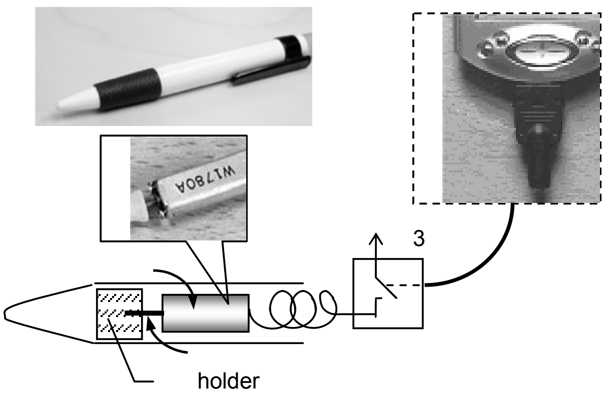


Fig. 1. Vibro-tactile pen: general view and schematics.

We used the cradle connector of Compaq iPAQ pocket PC which supports RS-232 and USB input/output signals. In particularly, DTR or/and RTS signals can be used to realize the motor control.

The software to create vibro-tactile patterns was written in Microsoft eMbedded Visual Basic 3.0. This program allows shaping some number of vibro-tactile patterns. Each of the tactons is composed of two sequential serial bursts with different frequency of the pulses. Such a technique based on contrast presentation of two well-differentiated stimuli of the same modality facilitates shaping the perceptual imprint of the vibro-tactile pattern. The number of bursts could be increased, but duration of the tacton shall be reasonable and shall not exceed 2 s. Durations of the pulses and pauses are setting in milliseconds. Number of pulses determines the duration of each burst. Thus, if the pattern consists of 10 pulses having frequency of 47.6 Hz (1+20 ms) and 10 pulses having frequency of 11.8 Hz, (5+80 ms) vibro-tactile pattern has the length of 1060 ms. All patterns are stored in the resource file “TPattern.txt” that can be loaded by the game or another application having special procedures to decode the description into output signals of the serial port according the script.

3. Method for Learning Vibro-Tactile Signals

Fingertip sensitivity is extremely important for some categories of physically challenged people such as the profoundly deaf, hard-of-hearing people and people who have low vision. We can find diverse advises how to increase skin sensitivity. For instance, Stephen Hampton in “Secrets of Lock Picking” [8] described a special procedure and the exercises to develop a delicate touch.

Sometimes, only sensitivity is not enough to remember and recognize vibration patterns and their combinations, especially when the number of the tactons is more than five. While high skin sensitivity can produce strong sensation, the discrimination of several tactile stimuli can remain quite poor. The duration of remembering tactile pattern depends on many factors which would include personal experience, making of the individual perceptive strategy, and the imaging system of alternative signals [7].

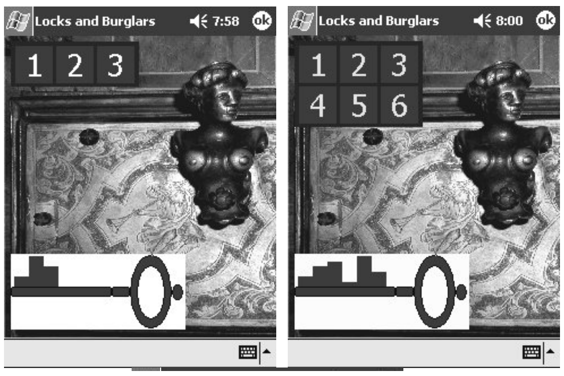


Fig. 2. Three levels of the game “Locks and Burglars”.

We propose special games and techniques to facilitate learning and manipulation by vibration patterns. The static scripts have own dynamics and provoke the player to make an individual strategy and mobilize perceptive skills. Let us consider a version of the game for the users having a normal vision.

The goal of the “Burglar” is to investigate and memorize the lock prototype to open it as fast as possible. There are three levels of difficulty and two phases of the game on each level. In the “training” mode (the first phase), the player can touch the lock as many times as s/he needs. After remembering tactons and their position, the player starts the game. By clicking on the label "Start", which is visible in training phase, the game starts and the key will appear (Fig. 2). The player has the key in hand and can touch it as many times as s/he needs. That is a chance to check the memory.

After player found known tactons and could suppose in which position of the lock button s/he had detected these vibrations before, it is possible to click once the lock button. If the vibration pattern of the button coincides with corresponding tacton of the key piece, the lock will have a yellow shine. In a wrong case, a shine will be red. Repeated pressing of the corresponding position is also being considered as an error.

There is a restricted number of errors on the each level of the game: single, four and six allowed errors. We assumed that 15 s per tacton is enough to pass the third level therefore the game time was restricted to 2.5 minutes. That conditions a selection of the strategy and improves learnability. After the player did not admit the errors at all the levels, the group of tactons could be replaced. Different groups comprising nine tactons allow learning whole vibro-tactile alphabet (27 tokens) sequentially.

All the data, times and number of repetitions per tacton, in training phase and during the game are automatically collected and stored in a log file. Thus, we can estimate which of the patterns are more difficult to remember and if these tactons are equally hard for all the players, their structure could be changed.

Graphic features for imaging, such as numbering or positioning (central, corners) lock buttons, different heights of the key pieces, and "binary construction" of the tactons, each tacton being composed of the two serial bursts of the pulses, should facilitate remembering spatial-temporal relations of the complex signals in the proposed system.

Another approach was developed to support blind interaction with tactile patterns, as the attentional competition between modalities often disturbs or suppresses weak differences of the tactile stimuli. The technique for blind interaction has several features. Screenshot of the game for non-visual interaction is shown in Fig. 3. There are four absolute positions for the buttons "Repeat", "Start" and two buttons are controlling the number of the tactons and the amount of the tactons within a playback sequence. Speech remarks support each change of the button state.

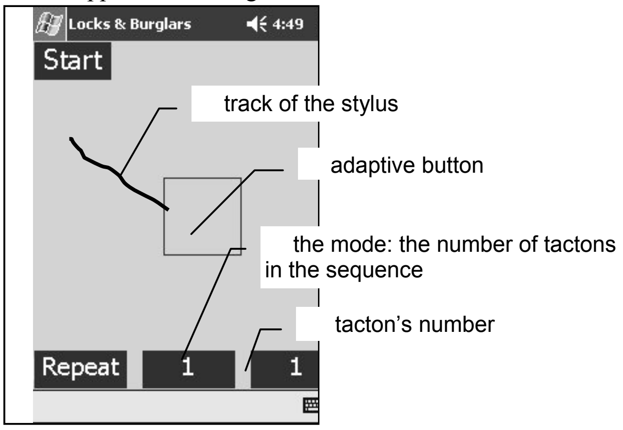


Fig. 3. The version of the game for blind player.

When blindfolded player should investigate and memorize the lock, s/he can make gestures along eight directions each time when it is necessary to activate the lock button or mark once the tacton by gesture and press down the button “Repeat” as many times as needed. The middle button switches the mode of repetition. Three or all the tactons can be played starting from the first, the fourth or the seventh position pointed by the last gesture.

Spatial-temporal mapping for vibro-tactile imaging is shown in Fig. 4. Playback duration for the groups consisting of 3, 6 or 9 tactons can reach 3.5 s, 7.2 s or 11 s including earcon to mark the end of the sequence. This parameter is important and could be improved when stronger tactile feedback could be provided with actuator attached to the stylus directly under the finger. In practice, only the sequence consisting of three tactons facilitates recognizing and remembering a sequence of the tactile patterns.

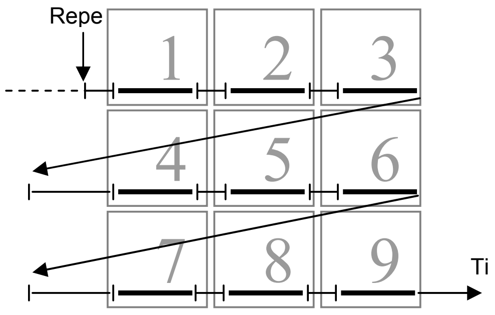


Fig. 4. Spatial-temporal mapping for vibro-tactile imaging: T_1 = 60 ms, T_2 = 1100 ms, T_3 = 300 ms.

To recognize gestures we used the metaphor of the adaptive button. When the player touches the screen, the square shape (Fig. 3) automatically changes position and finger or stylus occurs in the center of the shape. After the motion was realized (sliding and lifting the stylus), the corresponding direction or the button position of the lock will be counted and the tacton will be activated.

The button that appears on the second game phase in the bottom right position activates the tactons of the virtual key. At this phase, the middle button switches number of tactons of the key in a playback sequence. However, to select the button of the lock by gesture the player should point before what key piece s/he wishes to use. That is, the mode for playback of a single tacton should be activated. The absolute positions of software buttons do not require additional markers.

4. Evaluation of the Method and Pilot Results

The preliminary evaluation with able-bodied staff and students took place in the Department of Computer Sciences University of Tampere. The data were captured using the version of the game “Locks and Burglars” for deaf players. The data were collected concerning 190 trials in a total, of 18 players (Table 1). Despite of the fact, that the tactons have had low vibration frequencies of 47.6 Hz and 11.8 Hz, we cannot exclude an acoustic effect, as the players had a normal hearing. Therefore, we can just summarize general considerations regarding the difficulties in which game resulted and overall average results.

Table 1. The preliminary average results.

Level (tactons)	Trials	Selection time per tacton	Total selection time	Repeats per tacton	Err, %
1 (3)	48	3.8 s	12.4 s	4-7	7.7
2 (6)	123	3.4 s	16.8 s	3-13	13.3
3 (9)	19	1.7-11 s	47.3 s	4-35	55.6

The first level of the game is simple as memorizing of 2 out of 3 patterns is enough to complete the task. The selection time (decision-making and pointing the lock button after receiving tactile feedback in corresponding piece of the key) in this level did not exceed 3.8 s per tacton or 12.4 s to find matching of 3 tactons. The number of the repetitions to memorize 3 patterns was low, about 4 - 7 repetitions per tacton. The error rate (Err) was 7.7%. The error rate was counted as follows:

$$Err = \frac{[wrong\_selections]}{[trials]\times[tactons]}\times 100\% \ . \qquad (2)$$

The second level of the game (memorizing six tactons) was also not very difficult. An average time of the selection per tacton was about 3.4 s and 16.8 s in a total to find matching of six tactons. The number of the repetitions to memorize six patterns was varied from 3 to 13 repetitions per tacton. However, the error rate increased up to 13.3%, it is also possible due to the allowed number of errors (4).

The third level (nine tactons for memorizing) was too difficult and only three of 19 trials had finished by the win. The average time of the selection has been changed from 1.7 s up to 11 s per tacton and reached 47.3 s to find matching of nine tactons. While a selection time was about 30% of the entire time of the game, decision-making occupied much more time and players lost a case mostly due to limited time. The number of repetitions to memorize nine patterns in training phase varied significantly, from 4 up to 35 repetitions per tacton. Thus, we can conclude that nine tactons require of a special strategy to facilitate memorizing. However, the playback mode of the groups of vibro-tactile patterns was not used in the tested version. The error rate was too high (55.6%) due to the allowed number of errors (6) and, probably, because of the small tactile experience of the players.

The blind version of the game was briefly evaluated and showed a good potential to play and manipulate by vibro-tactile patterns even in the case when audio feedback was absent. That is, the proposed approach and the tools implemented provide the basis for learning and reading of the complex semantic sequences composed of six and more vibro-tactile patterns.

5. Conclusion

We designed a vibro-tactile pen and software intended to create tactons and semantic sequences consisting of the vibro-tactile patterns on mobile devices (iPAQ pocket PC). Tactile memory is the major restriction for designing a vibro-tactile alphabet for the hearing impaired people. We proposed special games and techniques to facilitate learning of the vibro-tactile patterns and manipulating by them. Spatial-temporal mapping for imaging vibro-tactile signals has a potential for future development and detailed investigation of the human perception of the long semantic sequences composed of tactons. After training, the tactons can be used as a system of non-verbal communication signals.

Acknowledgments

This work was financially supported by the Academy of Finland (grant 200761), and by the Nordic Development Centre for Rehabilitation Technology (NUH).

References

1. Antonets, V.A., Zeveke, A.V., Malysheva, G.I.: Possibility of synthesis of an additional sensory channel in a man-machine system. Sensory Systems, 6(4), (1992) 100-102
2. Blind Games Software Development Project. http://www.cs.unc.edu/Research/assist/et/projects/blind_games/
3. Cell Phones and PDAs. http://www.immersion.com/consumer_electronics/
4. Chang, A., O'Modhrain, S., Jacob, R., Gunther, E., Ishii, H.: ComTouch: Design of a Vibrotactile Communication Device. In: Proceedings of DIS02, ACM (2002) 312-320
5. Cutkosky, M.R., Howe, R.D.: Human Grasp Choice and Robotic Grasp Analysis. In S.T. Venkataraman and T. Iberall (Eds.), Dextrous Robot Hands, Springer-Verlag, New York (1990), 5 –31
6. Fukumoto, M. and Sugimura, T.: Active Click: Tactile Feedback for Touch Panels. In: Proceedings of CHI 2001, Interactive Posters, ACM (2001) 121-122
7. Geldard, F.: Adventures in tactile literacy. American Psychologist, 12 (1957) 115-124
8. Hampton, S.: Secrets of Lock Picking. Paladin Press, 1987
9. Hong Z. Tan and Pentland, A.: Tactual Displays for Sensory Substitution and Wearable Computers. In: Woodrow, B. and Caudell, Th. (eds), Fundamentals of Wearable Computers and Augmented Reality, Mahwah, Lawrence Erlbaum Associates (2001) 579-598
10. Michitaka Hirose and Tomohiro Amemiya: Wearable Finger-Braille Interface for Navigation of Deaf-Blind in Ubiquitous Barrier-Free Space. In: Proceedings of the HCI International 2003, Lawrence Erlbaum Associates, V4, (2003) 1417-1421
11. Nashel, A. and Razzaque, S.: Tactile Virtual Buttons for Mobile Devices. In: Proceedings of CHI 2003, ACM (2003) 854-855

12. Poupyrev, I., Maruyama, S. and Rekimoto, J.: Ambient Touch: Designing Tactile Interfaces for Handheld Devices. In: Proceedings of UIST 2002, ACM (2002) 51-60
13. Tactylus ^{tm} http://viswiz.imk.fraunhofer.de/~kruijff/research.html
14. Vibration Fuser for the Sony Ericsson P800. http://support.appforge.com/

Discussion

[Fabio Paterno] I think that in the example you showed for blind users a solution based on screen readers would be easier than the one you presented based on vibro-tactile techniques.

[Grigori Evreinov] A screen reader solution would not be useful for deaf and blind-deaf users.

[Eric Schol] Did you investigate the use of force-feedback joystick ?

[Grigori Evreinov] Yes, among many other devices ; like force-feedback mouse, etc. But main goal of the research was the application (game), not the device

Formalising an understanding of user–system misfits

Ann Blandford[1], Thomas R. G. Green[2] and Iain Connell[1]

[1] UCL Interaction Centre, University College London, Remax House, 31-32 Alfred Place London WC1E 7DP, U.K.
{A.Blandford,I.Connell}@ucl.ac.uk
http://www.uclic.ucl.ac.uk/annb/
[2] University of Leeds, U.K.

Abstract. Many of the difficulties users experience when working with interactive systems arise from misfits between the user's conceptualisation of the domain and device with which they are working and the conceptualisation implemented within those systems. We report an analytical technique called CASSM (Concept-based Analysis for Surface and Structural Misfits) in which such misfits can be formally represented to assist in understanding, describing and reasoning about them. CASSM draws on the framework of Cognitive Dimensions (CDs) in which many types of misfit were classified and presented descriptively, with illustrative examples. CASSM allows precise definitions of many of the CDs, expressed in terms of entities, attributes, actions and relationships. These definitions have been implemented in Cassata, a tool for automated analysis of misfits, which we introduce and describe in some detail.

1 Introduction

Two kinds of approach have dominated traditional work in usability of interactive systems: *heuristic* (or *checklist-based*) approaches giving a swift assessment of look-and-feel (usually independent of the tasks the system is designed to support), such as Heuristic Evaluation [17]; and *procedure-based* approaches for assessing the difficulty of each step of typical user tasks, such as Cognitive Walkthrough [20].

We present a technique based on a third approach, the analysis of *conceptual misfits* between the way the user thinks and the representation implemented within the system. Such misfits pertain to the concepts and relationships the user is manipulating in their work. Some misfits are surface-level – for example, users may work with concepts that are not directly represented within the system; conversely, users may be required to discover and utilise system concepts that are irrelevant to their conceptual models. Other misfits are structural, emerging only when the user manipulates the structure of some representation and finds that changes that are conceptually simple are, in practice, difficult to achieve.

We outline an approach to usability evaluation called Concept-based Analysis of Surface and Structural Misfits (CASSM), and present Cassata, a prototype analysis tool that supports the analyst in identifying misfits. As will become apparent, in CASSM structural misfits are not analysed directly in terms of the procedures that users follow to make a change, as might happen using a procedural approach; instead, CASSM identifies which elements of a structure are and are not accessible to a user and amenable to direct modification, thereby deriving warnings of potential misfits.

1.1 Misfits and their analysis

Many approaches to usability evaluation, including work in the previously-mentioned traditions of heuristic and procedure-based analysis, have generated lists of specific user problems with a given design, but have failed to impose any structure on the lists. Each user difficulty that is spotted is a thing in itself. From one occurrence we learn nothing about how to predict further occurrences, nor how to improve design practice.

CASSM builds on the approach known as the 'Cognitive Dimensions of Notations' framework (CDs) [3,4,14,15], in which some important classes of structural misfits have been articulated and described. For example, 'viscosity' describes the 'degree of resistance to small changes': in a viscous system, something is more difficult to change than it should be – a single conceptual action demands several device actions. An example would be adding a new figure near the beginning of a document then having to increment all subsequent figure numbers and within-text references to those numbers: some word processing applications explicitly support this activity but most do not, making it very repetitive. Viscosity may be a serious impediment to the user's task or it may be irrelevant to that task, if for instance the user is searching for a target but not trying to make a change; the CDs framework therefore distinguishes types of user activity and offers a conjecture as to how each dimension affects each activity.

The Cognitive Dimensions framework as originally created [12] was intended to promote quick, broad-brush evaluation, giving non-specialists a usability evaluation technique that was based on cognitive analysis yet required no expertise from the analyst. It relied purely on definition by example. To a degree this was successful. The idea of viscosity is intuitively appealing; examples can illustrate the idea; and a vocabulary of such ideas can be used to support discourse and reasoning about features of a design, with a view to improving that design [3]. However, despite the development of a CDs tutorial [14], and a questionnaire-based evaluation tool [2], potential users have found that they need to learn too many concepts and that those concepts are not defined closely enough to avoid disagreement over the final analysis.

More than one attempt has been made to sharpen the definitions of CDs [11,19] but those attempts have lost the feel of quick, broad-brush evaluation, making them unappealing to the intended user, the non-specialist analyst.

In this paper, we show that several CDs and related user–system misfits can be represented reasonably faithfully in a form that better preserves the original quick-and-dirty appeal of CDs. With these definitions, not only are the misfit notions clarified, but it becomes possible for potential misfit occurrences to be automatically identified within Cassata, the tool that we shall describe below.

It must be kept in mind throughout that our form of analysis can only describe *potential* user problems. Whether a particular misfit causes real difficulties will depend on circumstances that are not described here.

2 CASSM and Cassata: a brief introduction

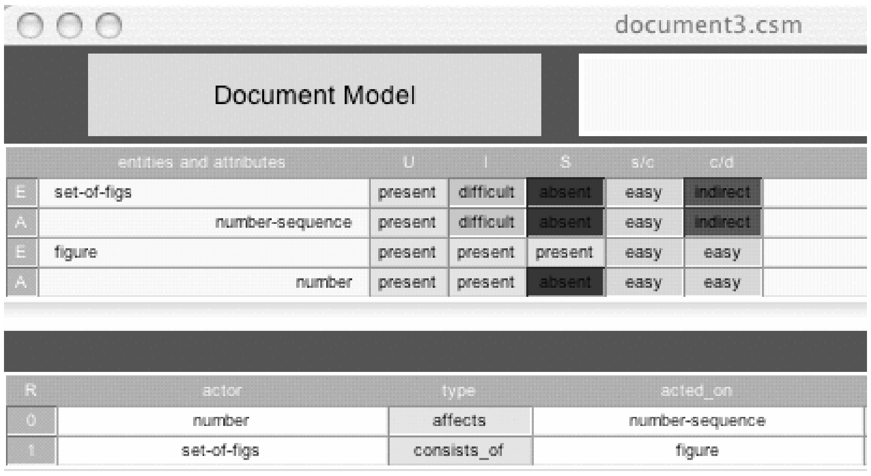
CASSM is a usability evaluation technique that focuses on the misfits between user and device. It was formerly known as Ontological Sketch Modelling (OSM [10]), because the approach involves constructing a partial (Sketchy) representation (Model) of the essential elements (Ontology) of a user–system interaction; the name has recently been changed to reflect a shift of focus towards the two types of misfits rather than the ontology representation.

CASSM developed from our earlier work on Entity Relationship Modelling of Information Artifacts (ERMIA [11]) and Programmable User Modelling (PUM [8]). It has also been informed by the work of others on what could broadly be termed misfit analysis, such as Moran's External Task Internal Task (ETIT) analysis [16] and Payne's Yoked State Spaces [18]. The basis of CASSM is to compare the concepts that users are working with (identified by an appropriate data gathering technique such as interviews, think-aloud protocols or Contextual Inquiry [1]) with the concepts implemented within the system and interface (identified by reference to sources such as system documentation or an existing implementation). Conceptual analysis involves identifying the concepts users are working with, drawing out commonalities across similar users (see for example [7]) to create the profile of a typical user of a particular type,; the analyst can then assess the quality of fit between user and system. As analysis proceeds, the analyst will start to distinguish between entities and attributes (as defined below), and to consider what actions the user can take to change the state of the system. Finally, for a thorough analysis, various relationships between concepts are enumerated to identify structural misfits. Each of these stages of misfit analysis is discussed in more detail below.

To support analysis, a demonstrator tool called Cassata is under development. Screen shots included in this paper are taken from version 2.1 of the tool. (Version 3 can be downloaded from the project web page [9].) The tool has provided a focus for developing the precise definitions of misfits included in this paper, and also a means of testing those definitions against a repertoire of examples that have previously been discussed informally.

Figure 1 shows the Cassata window for a partial description of a word processor document. For clarity, the picture is cropped from the right. This particular description is discussed in more detail in section 4.1; here we simply outline its main features.

It is a description of a set of figures (pictures or diagrams) in a document, which consists of one or more individual figures. For the user, there is the important idea that the figures should be sequentially numbered – so the *number-sequence* is important, and is an attribute of the *set-of-figs*. Each *figure* has an attribute which is its particular *number*, and changing a figure number changes the overall sequence of figure numbers.



	entities and attributes	U	I	S	s/c	c/d
E	set-of-figs	present	difficult	absent	easy	indirect
A	number-sequence	present	difficult	absent	easy	indirect
E	figure	present	present	present	easy	easy
A	number	present	present	absent	easy	easy

R	actor	type	acted_on
0	number	affects	number-sequence
1	set-of-figs	consists_of	figure

Fig. 1. Cassata data table for a partial description of a document. The upper table describes concepts (i.e. entities and their attributes); the lower describes relationships between those concepts.

The top half of the window shows information about concepts (entities such as *figure* and attributes such as *number*): for each concept, three columns show whether it is present, difficult or absent for the user, interface and system respectively; the next two columns show how easy it is to set or change the value of an attribute, or to create or delete an entity; the final column is a notes area in which the analyst can add comments. To take the first row as an example: the `set-of-figs` is a conceptual entity that is meaningful to the user, is not clearly represented at the interface ('difficult') and absent from the underlying system model. It is easy to create a set of figures, (because this happens automatically as the user adds figures) but harder to delete it (done indirectly because that requires deleting *all* the individual figures).

The bottom half of the window shows information about relationships (such as *affects* and *consists_of*) between concepts. In this particular case, the two lines of input state that changing any *number* (of a *figure*) affects the *number-sequence* (of the *set-of-figs*) and that a *set-of-figs* consists of (many) *figures*.

Having briefly presented the background to CASSM and Cassata, we now focus in more detail on the definitions of various kinds of misfits.

3 Surface Misfits

Surface misfits are those that become apparent without considering the details of structural representations within the system and how those representations are changed. Within 'surface', there are three levels of misfit: just identifying system and user concepts, with little reference to the interface between the two (section 3.1); more detailed analysis in terms of how well each concept is represented by the user, interface and system (section 3.2); and analysis in terms of what actions are needed to change the system, and whether there are problems with actions (section 3.3).

3.1 Level 1: Misfits between the user and the system

Misfits between user and system are probably the most important surface-level misfits. There are three important cases: user concepts that are not represented within the system; system concepts that are inaccessible to the user; and situations where a user concept and a system concept are similar but not identical.

User concepts that are not represented within the system cannot be directly manipulated by the user. The *set-of-figs* discussed above is an example of such a concept. Other examples are using a field in an electronic form to code information for which that form was not actually designed, or keeping paper notes alongside an electronic system to capture information that the system does not accept.

Unrepresented concepts are often the most costly form of misfit; they may force users to introduce workarounds, as users are unable to express exactly what they need to, and must therefore use the system in a way it was not designed for. They sometimes result in structural misfits such as viscosity, as described below.

System concepts that are not immediately available to the user need to be learned. At a trivial level, these might include strictly device-related concepts like scroll-bars, which may be simple to use but nevertheless need to be learnt. A slightly more complex example is the apparatus of layers, channels and masks found in many graphics applications – these can cause substantial user difficulties, particularly for novice users.

For users, these misfits may involve no more than learning a new concept, or they may require the users' constant attention to the state of something that has little significance to them, such as the amount of free memory.

User- and system concepts that are similar but non-identical, and which are often referred to by the same terms, can cause more serious difficulties. One example in the domain of diaries is the idea of a 'meeting'. When a user talks about a meeting, they usually mean a pre-arranged gathering of particular individuals at an agreed location with a particular broad purpose (and perhaps a detailed agenda). Within some shared diary systems, a meeting has a much more precise definition, referring to an event about which only other users of the same shared diary system can be kept fully informed, and which has a precise start time and precise finishing time, and possibly a precise location. The difference between these concepts is small but significant [5].

Another example, within the domain of ambulance dispatch, is the difference between a *call* and an *incident*. A particular system we studied processed information strictly in terms of calls, whereas staff dealt with incidents (about which there may be one or many calls); this was difficult to detect initially because the staff referred to them as 'calls' [7], but the failure of the system to integrate information about difference calls added substantially to staff workload as they processed the more complex incidents.

These misfits may cause difficulties because the user has to constantly map their natural understanding of the concept onto the one represented within the system, which may have a subtly different set of attributes.

3.2 Level 2: Adding Interface Considerations

The second level starts to draw out issues concerning the interface between user and system. For each of user, interface and system, a concept may be *present*, *difficult* or *absent.*

In all cases, *present* means clearly represented and *absent* means not represented. We assume that underlying system concepts are either present or absent, whereas for the user or at the interface there are concepts that are present but *difficult* in some way.

For users, *difficult* concepts are most commonly ones that are implicit– ideas they are aware of if asked but not ones they expect to work with. An example would be the end time of a meeting in the diary system mentioned above: if one looks at people's paper diaries, one finds that many engagements have start times (though these are often flagged as approximate – e.g. '2ish') but few have end times, whereas electronic diaries require every event to have an end time. This forces users to make explicit information that they might not choose to. Another source of difficulty might be that the user *has to learn* the concept.

Similarly, there are various reasons why a concept may be represented at the interface but in a way that makes it difficult to work with. Difficulties that interface objects may present include:

Disguised: represented, but hard to interpret;

Delayed: represented, but not available to the user until some time later in the interaction;

Hidden: represented, but the user has to perform an explicit action to reveal the state of the entity or attribute; or

Undiscoverable: represented only to the user who has good system knowledge, but unlikely to be discovered by most users.

Which of these apply in any particular case – i.e. why the interface object might cause user difficulties – is a further level of detail that can be annotated by the analyst; for the sake of simplicity, this additional level of detail is not explicitly represented within Cassata.

At the simplest level, anything that is *difficult* or *absent* represents a misfit that might cause user difficulties. As discussed above, concepts that are difficult or absent for the user are ones that need to be learnt and worked with; how much difficulty these actually pose will depend on the interface representation. Conversely, concepts that are present for the user but absent from the underlying system will force the user to find work-arounds. In addition, as discussed above, poor interface representations are a further source of difficulty that is not considered at level 1.

3.3 Level 3: Considering Actions

At levels 1 and 2, we have referred to ‘concepts’ without it being necessary to distinguish between them. For deeper analysis, it becomes necessary to distinguish between entities and attributes. A description in terms of entities and attributes is illustrated in the screen-shot from the Cassata tool shown in Figure 1 (above). There, we used the terms ‘entity’ and ‘attribute’ without precisely defining them.

An *entity* is a concept that can be created or deleted, or that has attributes which the analyst wants to enumerate. In figure 1, entities are shown in the left-hand column, left-justified. Note also the ‘E’ in the left margin.

An *attribute* is a property of an entity. In Figure 1, attributes are shown right-justified in the left-hand column. Note also the ‘A’ in the left margin. Attributes can be set (‘**S**/C’) or changed (‘**C**/D’).

For economy of space, the same columns are used to define how easy it is to create (‘S/**C**’) or delete (‘C/**D**’) entities. Each of these actions can be described as follows:

Easy: no user difficulties.

Hard: difficult for some reason (e.g. undiscoverable action, moded action, delayed effect of action). For example, it is possible to select a sentence in MS Word by pressing the control key (‘apple’ key on a Mac) and clicking anywhere in the sentence; few users are aware of this.

Indirect: effect has to be achieved by changing something else in the system; for example, as discussed above, it is not possible to directly change the sequence of figure numbers.

Cant: something that cannot be changed, that the analyst thinks might cause subsequent user difficulties.

Fixed: something that cannot be changed, that is not, in fact, problematic; for example, an entity may be listed simply because it has important attributes that need to be enumerated or analysed.

BySys: this denotes aspects of the system that may be changed, but not by the user (this may include by other agents – e.g. over a network, or simply other people). Many of these cases are not actually problems, and it is up to the analyst to consider implications.

Just as describing concepts as ‘present’, ‘absent’ or ‘difficult’ helps to highlight some conceptual difficulties, so describing actions in terms of ‘easy’, ‘hard’ , indirect’, ‘cant’, ‘fixed’ and ‘bySys’ highlights conceptual difficulties in changing the state of the system.

3.4 Surface-level misfits and their Cognitive Dimensions

We turn now to the use of CASSM to articulate part of the Cognitive Dimensions framework introduced above, starting with surface-level misfits – notably abstraction level and visibility.

Abstraction level: devices may be classed as imposing the use of abstractions ('abstraction-hungry' in Green's terminology), rejecting the use of abstractions ('abstraction-hating'), or allowing but not imposing abstractions ('abstraction-neutral'); further, the abstractions themselves may be domain-based or device-based. CASSM can express these distinctions reasonably well and can therefore detect some of the misfits, among them:

- domain abstractions that are part of the user's conceptual but are not implemented within the device;
- device abstractions imposed upon the user.

Imposed device abstractions have to be learnt in order to work effectively with the device, such as style sheets or graphics masks, and are therefore easy or difficult to learn according to how well they are represented at the interface (as discussed above).

Visibility: the user's ability to view components readily when required, preferably in juxtaposition to allow comparison between components. CASSM cannot at present express either inter-item juxtaposability nor the number of search steps required to bring a required item to view ('navigability') but captures the essence of visibility by designating those concepts that are hidden, disguised, delayed or undiscoverable as 'difficult' in the interface representation.

4 Structural misfits: taking account of relationships

As discussed above, structural misfits refer to the structure of information, and how the user can change that structure. Here, we present the structural misfits of which we are currently aware. These are a subset of Green's Cognitive Dimensions [3]. It is worth noting that structural misfits only apply to systems where the system state can be changed in a meaningful way by the user. Thus, systems such as web sites or vending machines do not generally suffer from structural misfits. However, systems such as drawing programs, word processors, music composition systems and design tools are prone to these misfits.

Another point to note is that although structural misfits are much finer-grained than the bolder surface-level misfits discussed above, they can be immense sources of user frustration and inefficiency.

Structural misfits depend on relationships that hold within the data. Five kinds of relationships are currently defined within Cassata. These are: *consists_of*, *device_constraint*, *goal_constraint*, *affects*, and *maps_onto*. As for entities and attributes, it is possible (though not always necessary) to state how well these relationships are represented at the interface, to the user, or in the underlying system.

Consists_of takes two arguments, which we call Actor and ActedOn, which are both concepts. This means that the first consists_of the second: chapter consists_of paragraphs; set-of-paragraphs consists_of paragraphs (e.g. sharing a paragraph style); etc.

Device_constraint also takes two arguments, both concepts. The value of Actor constrains the possible values of ActedOn. For example, considering drawing a map on the back of an envelope, the starting_position (for drawing) constrains the location of a particular instruction. An easier example is that the field-width for a data entry field constrains the item-width for any items to be put in that field.

Goal_constraint takes only one argument (ActedOn), which is the concept on which there is some domain-based constraint. For example, when writing a conference paper such as this one, it is common to have a limit on the length of a document.

Affects is concerned with side-effects: that changing the value of one concept will also change the value of another. For example, changing the number of words in a document will change its length.

Maps_onto is a simple way of expressing the idea that two concepts are very similar but not quite identical. These are most commonly a domain-relevant concept and a device-relevant one. For example, a (user) meeting maps_onto a (diary-entry) meeting but, depending on the form of the diary, the two meeting types may have importantly different attributes.

We now consider three important classes of structural misfits: viscosity (section 4.1), premature commitment (section 4.2) and hidden dependencies (section 4.3). In what follows, we take A to be an entity of interest with an attribute P, and B to be some other entity with attribute Q. these are defined in the top window by juxtaposition (i.e. attributes always appear immediately below the entity to which they pertain).

4.1 Viscosity

As discussed above, "viscosity" captures the idea that a system is difficult to change in some way. Green [13] distinguished two types of viscosity, repetition and knock-on, which can be defined as follows.

1) Repetition viscosity occurs when a single action within the user's conceptual model requires many, repetitive device actions.

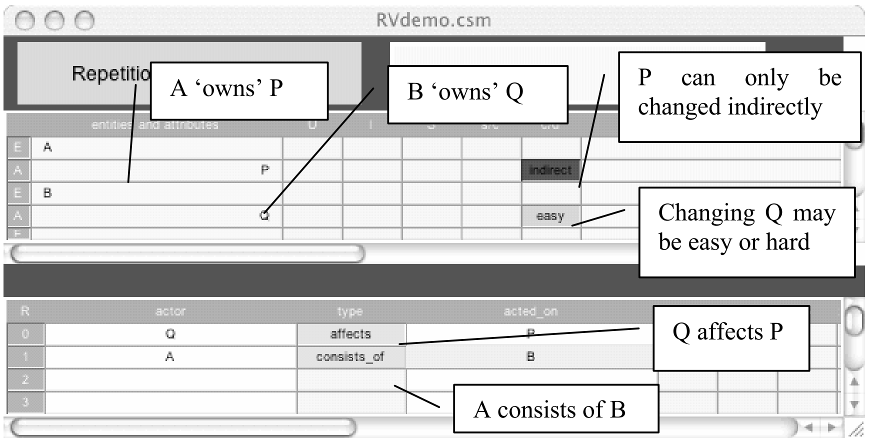
Changing attribute P of entity A, A(P), needs many actions if:

```
A(P) is not directly modifiable
B(Q) affects A(P)
B(Q) is modifiable
A consists-of B
```

For example, as discussed above (section 2), we get repetition viscosity on figure numbers in a document because whenever a figure is added, deleted or moved, a range of figures need to be re-numbered one by one. Stated more formally:

```
set-of-figs(number-sequence) is not directly modifiable
figure(number) affects set-of-figs(number-sequence)
figure(number) is modifiable
set-of-figs consists-of figure
```

Figure 2 shows the basic requirements on a model for it to exhibit Repetition Viscosity. Note in particular the use of 'indirect' to denote something that can be changed, but not directly. Figure 3 shows the output when this particular model is assessed by Cassata.



	entities and attributes	
E	A	
A	P	indirect
E	B	
A	Q	easy

R	actor	type	acted_on
0	Q	affects	P
1	A	consists_of	B
2			
3			

Fig. 2. Repetition Viscosity.

```
Repetition Viscosity Check ---- Repetition Viscosity Model

  attribute "Q" affects "P"
  entity "A" consists_of "B"
 "A " owns "P"
 "P " is not directly modifiable
 "B " owns "Q"
 "Q " is directly modifiable

possible case of repetition viscosity:
to change "P" user may have to change all instances of "Q"
```

Fig. 3. Output from Repetition Viscosity analysis in Cassata.

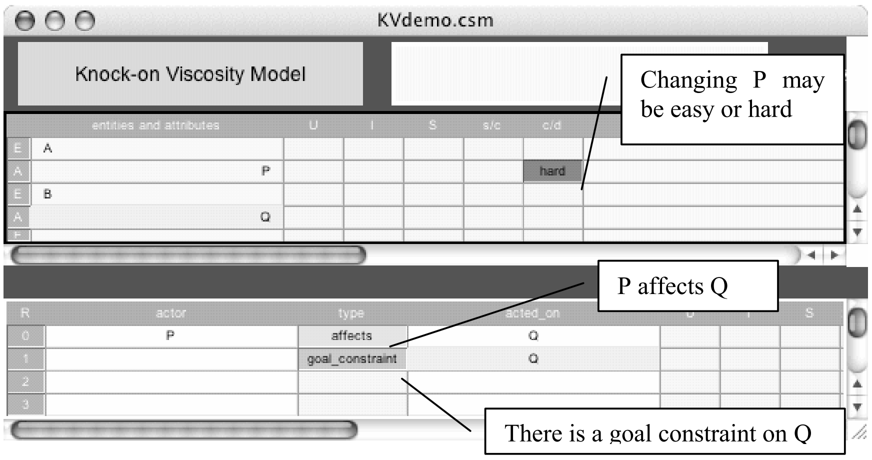
2) Knock-on viscosity: changing one attribute may lead to the need to adjust other things to restore the internal consistency. (In North America, a better-known phrase for the same concept appears to be ‘domino effect’.)

Changing A(P) has possible knock-on if:

```
A(P) is modifiable
modifying A(P) affects B(Q)
there is a domain constraint on B(Q)
```

Timetables and schedules typically contain high knock-on viscosity; if one item is re-scheduled, many others may have to be changed as well.

Figure 4 shows the conditions for a model to exhibit Knock-on Viscosity. Figure 5 shows the output when this model is assessed by Cassata.


KVdemo.csm
Knock-on Viscosity Model
Changing P may be easy or hard
P affects Q
There is a goal constraint on Q

	entities and attributes	U	I	S	s/c	c/d
E	A					
A	P					hard
E	B					
A	Q					

R	actor	type	acted_on	U	I	S
0	P	affects	Q			
1		goal_constraint	Q			
2						
3						

Fig. 4. Knock-on Viscosity.

```
Knock-on Viscosity Check ---- Knock-on Viscosity Model

   attribute "P" affects "Q"
   there is a goal_constraint on "Q"
  "P " is directly modifiable

possible case of knock-on viscosity
modifying "P" may violate a domain constraint for "Q"
```

Fig. 5. Output from Knock-on Viscosity analysis in Cassata.

4.2 Premature Commitment

Informally, premature commitment occurs when the user has to provide information to the system earlier than they would wish or are prepared for. We have several sets of conditions that alert to possible premature commitment.

1) Non-modifiability premature commitment: As discussed above (under actions), if an attribute cannot be changed after it has been set then the system possibly demands premature commitment:

```
A(P) is settable
A(P) is not modifiable
```

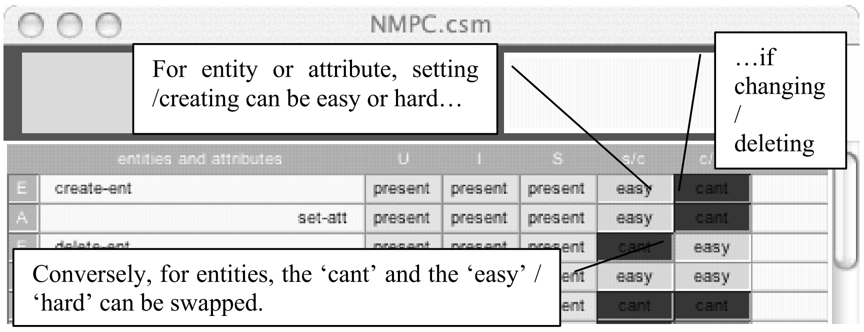
Some painting tools exhibit this type of premature commitment: that the width and colour of a line cannot be changed once it has been set.

Extending this to entities, we may get potential non-modifiability premature commitment if entities can be created but not subsequently deleted:

```
A is creatable
A is not deletable
```

In principle the converse may hold too, but there are few situations in which that would class as premature commitment (rather than simply an irreversible action).

Figure 6 shows the conditions for a model to exhibit this kind of Premature Commitment. Figure 7 shows the output when this particular model is assessed by Cassata.


NMPC.csm
For entity or attribute, setting /creating can be easy or hard…
…if changing / deleting
Conversely, for entities, the 'cant' and the 'easy' / 'hard' can be swapped.

	entities and attributes	U	I	S	s/c	c/
E	create-ent	present	present	present	easy	cant
A	set-att	present	present	present	easy	cant
E	delete-ent	present	present	present	cant	easy
				ent	easy	easy
				ent	cant	cant

Fig. 6. Non-modifiability Premature Commitment.

```
Non-modifiability Premature Commitment ---- test NMPC Model

possible non-modifiable premature commitment:
  entity "create-ent" can be created but not deleted
====

possible non-modifiable premature commitment:
  attribute "set-att" can be set but not changed
====

possible non-modifiable premature commitment:
  entity "delete-ent" can be deleted but not created
====
```

Fig. 7. Output from Non-modifiability PC analysis in Cassata.

2) Abstraction-based premature commitment: If a user has to define an abstraction in order to avoid repetition viscosity, and that abstraction has to be defined in advance, then the system potentially creates abstraction-based premature commitment. Frequently that abstraction will be a simple grouping. A common example of potentially premature commitment to abstractions is the defining of paragraph styles before starting to create a technical document. The purpose is to avoid repetition viscosity by allowing all paragraphs of one type to be reformatted in one action, but the problem is to foresee the required definitions. A more technical example would be the creation of a class hierarchy in object-oriented programming.

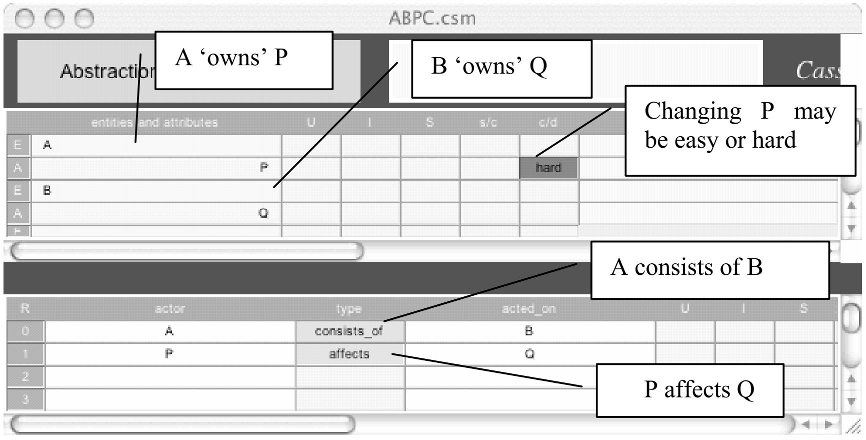
The conventional analysis in the Cognitive Dimensions framework is to treat the abstraction management components of the system as a separate sub-device, which may have its own properties of viscosity, hidden dependencies, etc [4]. In CASSM we take a simplified approach such that this type of premature commitment is highlighted if:

```
A consists-of B
A(P) is directly modifiable
A(P) affects B(Q)
```

The paragraph styles case would be represented thus:

```
Paragraph has attribute style
Set-of-paragraphs has attribute style-description
Set-of-paragraphs consists-of paragraph
Style-description is directly modifiable
Changing style-description causes style to change
```

Figure 8 shows the basic requirements on a model for it to exhibit Abstraction-based Premature Commitment. Figure 9 shows the output when this particular model is assessed by Cassata.



	entities and attributes		U	I	S	s/c	c/d
E	A						
A		P					hard
E	B						
A		Q					

R	actor	type	acted_on	U	I	S
0	A	consists_of	B			
1	P	affects	Q			
2						
3						

Fig. 8. Abstraction-based premature commitment.

```
Abstract-based Premature Commitment Check ---- Abstraction-based PC Model

  attribute "P" affects "Q"
  entity "A" consists_of "B"
 "A " owns "P"
 "P " is directly modifiable
 "B " owns "Q"

possible case of abstract-based premature commitment:
need to create an abstraction "A" to change all instances of "Q"
```

Fig. 9. Output from Abstraction-based PC analysis in Cassata.

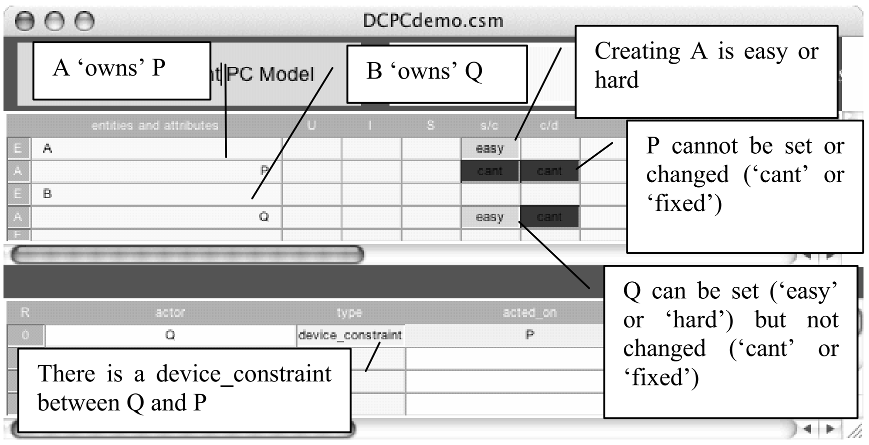
3) Device-constraint premature commitment: Here, setting an attribute of one entity constrains the way that new instances of another entity can be created:

```
B(Q) is settable
A(P) is not settable
There is a device constraint between B(Q) and A(P)
It is possible to add more As
```

As mentioned above (when defining *device constraint*), one example of this is drawing a map on the back of an envelope; another is that of setting the field width in a data structure when the size of all items to be entered in that field is not known (here, “>=” is an example of a device constraint):

```
field(width) is settable
item(width) is not settable
field(width)>=item(width)
more items can be added
```

Figure 10 shows the basic requirements on a model for it to exhibit Device-constraint Premature Commitment. Figure 11 shows the output when this particular model is assessed by Cassata.


DCPCdemo.csm
PC Model
A ‘owns’ P
B ‘owns’ Q
Creating A is easy or hard
P cannot be set or changed (‘cant’ or ‘fixed’)
Q can be set (‘easy’ or ‘hard’) but not changed (‘cant’ or ‘fixed’)
There is a device_constraint between Q and P

	entities and attributes	U	I	S	s/c	c/d
E	A				easy	
A	P				cant	cant
E	B					
A	Q				easy	cant

R	actor	type	acted_on
0	Q	device_constraint	P

Fig. 10. Device-constraint premature commitment.

```
Device-constraint Premature Commitment Check ---- Device-constraint PC Model

  attribute "Q" imposes a device_constraint on "P"
 "Q " can be set but not changed
 "P " cannot be either set or changed
 "A " can be created

possible case of device-constraint premature commitment:
attribute "P" may be constrained by "Q"
```

Fig. 11. Output from Device-constraint PC analysis in Cassata.

4.3 Hidden Dependencies

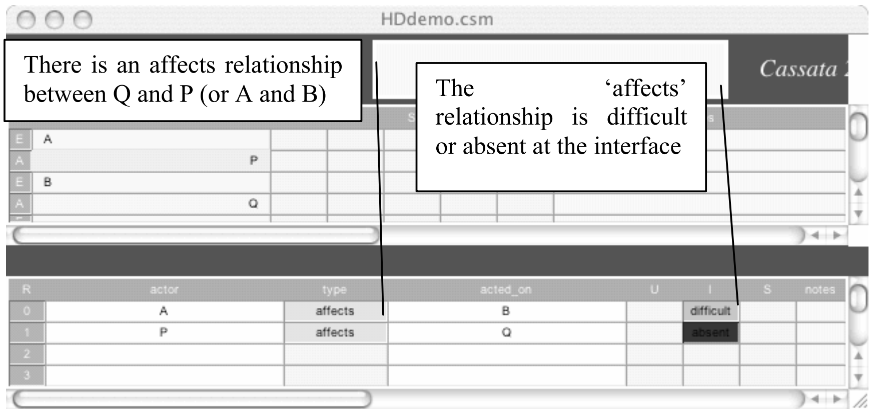
A hidden dependency occurs when important links between concepts are not visible (or otherwise readily available to the user). Spreadsheets contain many hidden dependencies, so that changing a value or formula somewhere in a sheet can have unanticipated knock-on effects elsewhere in the sheet. Similarly, changing a style in MS Word can have unexpected knock-on effects on other styles through the style hierarchy. This is formalised simply:

```
Changing C affects D
The relationship is not visible
```

Here, C and D are concepts (entities or attributes). They may even be the same concept. For example, in the word processor because the concept 'style definition' denotes an aggregate of styles formed into a hierarchy, changing any one definition potentially changes other definitions that refer to it, so we have the reflexive relationship:

```
Changing style-definition affects style-definition
The relationship is not visible
```

Figure 12 shows the basic requirements on a model for it to exhibit Hidden Dependencies. Figure 13 shows the output when this particular model is assessed by Cassata.


HDdemo.csm
Cassata
There is an affects relationship between Q and P (or A and B)
The 'affects' relationship is difficult or absent at the interface

E	A
A	P
E	B
A	Q

R	actor	type	acted_on	U	I	S	notes
0	A	affects	B		difficult		
1	P	affects	Q		absent		
2							
3							

Fig. 12. Hidden Dependencies.

```
Hidden Dependencies Check ---- Hidden Dependencies Model

  "A" affects "B"

possible case of hidden dependency:
there may be hidden dependency between "A" and "B"
====
  "P" affects "Q"

possible case of hidden dependency:
there may be hidden dependency between "P" and "Q"
```

Fig. 13. Output from Hidden Dependencies analysis in Cassata.

5 Conclusions

In this paper, we have presented a particular approach to assessing the usability of an interactive system based on the idea of 'quality of fit' between user and system. In particular, we have used the ontology of CASSM (considering entities, attributes, actions and a set of defined relationship types, and properties of each of these) to deliver precise definitions of various kinds of surface and structural misfits. The structural misfits are all based on Green's [12] Cognitive Dimensions. Some of the surface misfits can also be identified as CDs, but most are not, and all have been independently derived from the basic CASSM ontology.

The prototype Cassata tool allows CASSM-based descriptions of systems to be created quickly and with a minimum of special concepts. When a CASSM description has been entered into Cassata, potential occurrences of both surface and structural misfits can be automatically identified, thereby alerting analysts to possible usability problems. With the help of Cassata we have preserved the original quick-to-do feel of the Cognitive Dimensions analysis, unlike previous efforts at formalising the Cognitive Dimensions framework [11,19].

In practice, we have found that it is usually easier to identify structural misfits informally (as has been done historically with CDs) than by generating the full CASSM representation in Cassata; in this case, the role of the formalisation is to validate that informal understanding and make it more precise. The Cassata tool provides simple but valuable support in identifying both surface and structural misfits.

We are not claiming that the set of misfits presented here is complete. There are many different *kinds* of misfits between users and systems, many of which are outside the scope of CASSM – for example, inconsistencies in procedures for similar tasks would be picked up by other techniques but are not directly addressed within CASSM. In this work, we have focused on conceptual misfits, which have not been widely recognised in earlier work on usability evaluation.

The work reported here is ongoing; elsewhere, we have reported the application of CASSM to various kinds of interactive systems [7,10]. Current work is addressed at refining the Cassata prototype, extending the set of structural misfits and scoping CASSM by comparison with other usability evaluation techniques (e.g. [6]). We believe that this work makes an important contribution to the overall repertoire of evaluation approaches for interactive systems.

Acknowledgements

This work is supported by EPSRC grant GR/R39108.

References

1. Beyer, H., Holtzblatt, K.: *Contextual Design*. San Francisco : Morgan Kaufmann. (1998).
2. Blackwell, A.F., Green, T.R.G.: A Cognitive Dimensions questionnaire optimised for users. In A.F. Blackwell & E. Bilotta (Eds.) *Proceedings of the Twelfth Annual Meeting of the Psychology of Programming Interest Group* (2000).137-152.
3. Blackwell, A., Green, T. R. G.: Notational systems – the Cognitive Dimensions of Notations framework. In J. Carroll (ed.), *HCI Models, Theories and Frameworks*, Morgan Kaufmann. (2003) 103-134.
4. Blackwell, A., Hewson, R., Green, T. R. G.: The design of notational systems for cognitive tasks. E. Hollnagel (ed.) In E. Hollnagel (Ed.), *Handbook of Cognitive Task Design*. Mahwah, N.J.: Lawrence Erlbaum. (2003) 525-545.
5. Blandford, A. E., Green, T. R. G.: Group and individual time management tools: what you get is not what you need. *Personal and Ubiquitous Computing*. Vol 5 No 4. (2001) 213–230.
6. Blandford, A., Keith, S., Connell, I., Edwards, H.: Analytical usability evaluation for Digital Libraries: a case study. In *Proc. ACM/IEEE Joint Conference on Digital Libraries*. (2004) 27-36.
7. Blandford, A. E., Wong, B. L. W., Connell, I. W., Green, T. R. G.: Multiple viewpoints on computer supported team work: a case study on ambulance dispatch. In X. Faulkner, J. Finlay & F. Détienne (eds), *Proc. HCI 2002 (People and Computers XVI*), Springer (2002) 139-156.

8. Blandford, A. E., Young, R. M.: Specifying user knowledge for the design of interactive systems. *Software Engineering Journal*. 11.6, (1996) 323-333.
9. CASSM: Project web site www.uclic.ucl.ac.uk/annb/CASSM.html
10. Connell, I., Green, T., Blandford, A.: Ontological Sketch Models: highlighting user-system misfits. In E. O'Neill, P. Palanque & P. Johnson (Eds.) *People and Computers XVII, Proc. HCI'03*. Springer. (2003) 163-178.
11. Green, T. R. G., Benyon, D.: The skull beneath the skin: entity-relationship models of information artifacts. *International Journal of Human-Computer Studies*, 44 (1996) 801-828
12. Green, T. R. G.: Cognitive dimensions of notations. In A. Sutcliffe and L. Macaulay (Eds.) *People and Computers V*. Cambridge University Press. (1989) 443-460
13. Green, T.R.G.: The cognitive dimension of viscosity - a sticky problem for HCI. In D. Diaper and B. Shackel (Eds.) *INTERACT '90*. Elsevier. (1990)
14. Green, T. R. G., Blackwell, A. F.: Cognitive dimensions of information artefacts: a tutorial. http://www.cl.cam.ac.uk/~afb21/CognitiveDimensions/CDtutorial.pdf (1998)
15. Green, T. R. G., Petre, M.: Usability analysis of visual programming environments: a 'cognitive dimensions' framework. *J. Visual Languages and Computing*, 7, (1996) 131-174.
16. Moran, T. P.: Getting into a system: external-internal task mapping analysis, in A. Janda (ed.), *Human Factors in Computing Systems*, (1983) pp.45-49.
17. Nielsen, J.: Heuristic evaluation. In J. Nielsen & R. Mack (Eds.), *Usability Inspection Methods*, New York: John Wiley (1994) 25-62.
18. Payne, S. J., Squibb, H. R., Howes, A.: The nature of device models: the yoked state space hypothesis, and some experiments with text editors. *Human-Computer Interaction*, 5. (1990) 415-444.
19. Roast, C., Khazaei, B., Siddiqi, J.: Formal comparison of program modification. In *IEEE Symposium on Visual Languages*, IEEE Computer Society (2000). 165-171.
20. Wharton, C., Rieman, J., Lewis, C., Polson, P.: The cognitive walkthrough method: A practitioner's guide. In J. Nielsen & R. Mack (Eds.), *Usability Inspection Methods*. New York: John Wiley (1994) 105-140.

Discussion

[Willem-Paul Brinkman] In the case of misfits, the evaluator has to come up with an idea of what concepts/ideas users are using, and whether or not they map on the concepts of the system (system model/image). However, how does the evaluator check, if his/her ideas/concepts map with ideas/concepts the users have?

> [Ann Blandford] You present your finding to the users, and ask them whether they agree with having/using these concepts. At the moment this seems the best and most practical way.

[Jürgen Ziegler] How do dimensions like 'viscosity' relate to other, more established usability measures like 'effectiveness'?

> [Ann Blandford] Effectiveness might be a higher level concept, viscosity addresses sub aspects.

[Tom Ormerod] The distinction between concepts and tasks is interesting, though examples seemed to be about the tasks. Is CASSM about discovering concepts?

> [Ann Blandford] With the figure-numbering example, it is about making explicit an issue that is implicit, so yes

[Tom Ormerod] What would CASSM offer to the easier example of the problem of understanding the layers concept?

[Ann Blandford] It suggests a search for ways to represent the layers explicitly at the interface.

Supporting a Shared Understanding of Communication-Oriented Concerns in Human-Computer Interaction: a Lexicon-based Approach

Simone Diniz Junqueira Barbosa[1], Milene Selbach Silveira[2], Maíra Greco de Paula[1], Karin Koogan Breitman[1]

[1]Departamento de Informática, PUC-Rio
Marquês de São Vicente, 225 / 4º andar RDC
Gávea, Rio de Janeiro, RJ, Brazil, 22453-900

[2] Faculdade de Informática, PUCRS
Av.Ipiranga, 6681, Prédio 30, Bloco 4
Porto Alegre, RS, Brazil, 90619-900

simone@inf.puc-rio.br, milene@inf.pucrs.br, mgreco@inf.puc-rio.br, karin@les.inf.puc-rio.br

Abstract. This paper discusses the role of an enhanced extended lexicon as a shared communicative artifact during software design. We describe how it may act as an interlingua that captures the shared understanding of both stakeholders and designers. We argue for the need to address communicative concerns among design team members, as well as from designers to users through the user interface. We thus extend an existing lexicon language (LEL) to address communication-oriented concerns that user interface designers need to take into account when representing their solution to end users. We propose that the enhanced LEL may be used as a valuable resource in model-based design, in modeling the help system, and in engineering the user interface elements and widgets.

Keywords: communication-centered design, model-based design of human-computer interaction, semiotic engineering, language extended lexicon

1 Introduction

In this paper, we describe a lexicon-based representation to express domain and application concepts during the design process. We propose that, by doing so, designers, users and other stakeholders may have a shared understanding of the application, detailing its relevant concepts and their relationships. We have argued elsewhere that we need representations that will make possible a more balanced participation of stakeholders and team players from different interdisciplinary background during design [3]. This paper will focus on the communicative concerns that (esp. interaction) designers must deal with throughout the design process. We follow Preece et al.'s definition of interaction design: "designing interactive products to support people in their everyday and working lives" [26, p.6]. This definition is in accordance with Mullet & Sano's perspective that human-computer interaction (HCI) is "concerned most directly with the user's experience of a form in the context of a specific task or problem, as opposed to its functional or aesthetic qualities in isolation" [20, p.1]. Within HCI, semiotic engineering [9,10] has emerged as a semiotics-based theory [11,24] that describes and explains HCI phenomena, adopting primarily a media perspective on the use of computer artifacts [16].

Scenarios have been used as the primary representation to foster communication among team members and stakeholders [6]. We propose that an enriched lexicon can complement scenarios by representing together the different perspectives of each sign, which are typically scattered in many scenarios. This lexicon can be used to establish a common vocabulary throughout various design stages. By doing so, we believe it would be easier to build the design models taking both the lexicon and the scenarios as a starting point. In particular, such a lexicon can be used to derive three important kinds of resources: the user interface signs, which users should understand and learn to manipulate to make the most of their interaction with application [9,10]; the help content [29, 30]; and ontologies [13, 14], which can be employed in user, dialog and task modeling, especially in adaptive user interfaces [22] and the semantic web [4].

2 Semiotic Engineering and Communication-centered Design

Semiotic Engineering focuses on the engineering of signs that convey what HCI designers and users have in mind and what effect they want to cause in the world of things, practices, ideas and experiences [9,10]. The interface signs constitute a message sent from designers to users, representing the designers' solution to what they believe is the users' problems, what they have interpreted as being the users' needs and preferences, what the answer for these needs is and how they implemented their vision as an interactive system. In particular, semiotic engineering proposes a change of focus from *producing* to *introducing* design artifacts to users [10].

Our work builds on semiotic engineering by attempting to ensure that domain concepts are well represented and understood by every team member[8] before proceeding to later design stages. We need to promote the shared understanding among the team members (for instance, by representing domain concepts and their interrelationships), and to allow designers to represent communication-centered concerns developed for improving designer-to-user communication during interaction [9,10]. Our basic assumption is that, in order to increase the chances of engineering adequate signs at the user interface to convey the designers' vision and thus properly introduce the design artifact, we need to first establish this vision and communicate it

[8] By "team members" we mean the stakeholders (clients and users) and the designers (members of the development team from various disciplines, such as software engineering, human-computer interaction, graphics design, linguistics, psychology and so on).

effectively among team members themselves, always from a user's point of view (Fig. 1).

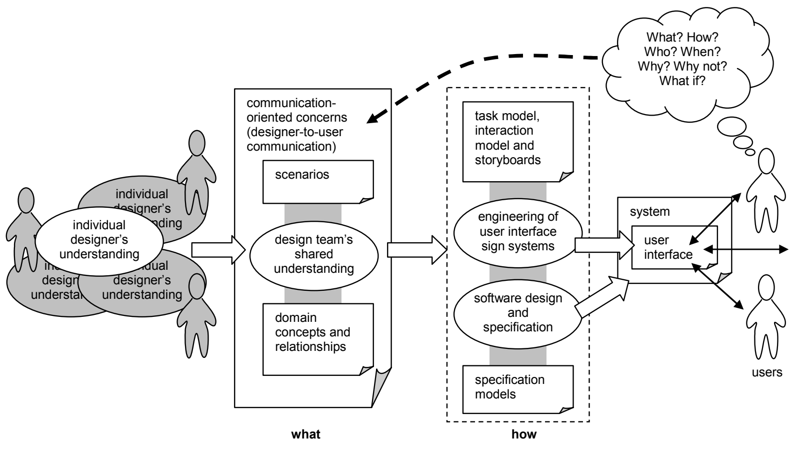

What? How? Who? When? Why? Why not? What if?
individual designer's understanding
communication-oriented concerns (designer-to-user communication)
scenarios
design team's shared understanding
domain concepts and relationships
task model, interaction model and storyboards
engineering of user interface sign systems
software design and specification
specification models
system
user interface
users
what
how

Fig. 1. Communication-centered design.

The communication-oriented concerns we will address in this paper are derived from studies about users' frequent doubts [1,28], as indicated by the dashed arrow in Fig. 1. These concerns will be described in section 4.

If designers are unable to convey their vision to each other and to every stakeholder, they will hardly succeed in conveying it to users (through carefully designing the user interface). If, on the other hand, they succeed in promoting designer-designer communication via communication artifacts, they will be better equipped to communicate with users through the user interface, i.e., to engineer the user interface sign systems. This way, we aim to take one step towards a communication-centered approach to interactive software design and development.

3 The Language Extended Lexicon (LEL)

As a starting point to building our communication artifacts, we take on the requirements engineering work of the Language Extended Lexicon (LEL) [18]. The LEL is a representation of the signs in the language of the application domain. LEL is anchored on the idea that one must first "understand the *language* of the problem, without worrying about understanding the *problem*" [18]. Researchers in different areas have pointed out the strong relationship between culture and language. In semiotics, in particular, the works of Eco and Danesi pay special attention to the web of language, culture and social environments [8,11]. In software design, the strength of using language to promote a shared understanding of the problem design domain and also of the solution accounts for the success of scenario-based approaches in various design stages [6].

To capture the language of the application domain and represent it in a Universe of Discourse (UofD), each term in LEL has two types of description: (i) *notion*, the denotation of the term or phrase; and (ii) *impact*, extra information about the context at hand[9]. In addition, each lexicon term is classified in four categories: object, subject, verb and state. The strong points in LEL are the principles of *closure* and of *minimal vocabulary*. The principle of closure attempts to "maximize the use of signs in the meaning of other signs", whereas the principle of minimal vocabulary "demands that external vocabulary be minimized and reduced to the smallest set possible". The external vocabulary is the set of terms that lie outside of the UofD. These terms should belong to the basic vocabulary of the natural language in use, i.e., be clearly known to every stakeholder.

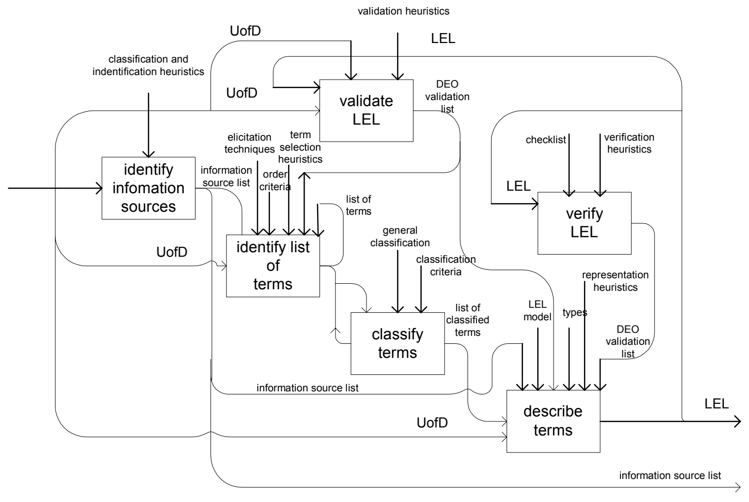


Fig. 2. Lexicon construction process [17].

Kaplan and co-authors describe in detail the process of constructing a LEL representation [17]. It comprises six steps, as depicted in Fig. 2. First one needs to identify the main information sources of the UofD, such as people and documents. Then, one must identify a list of relevant terms to be included in the UofD. By observing how people work and interviewing them, as well as by reading the documents and inspecting the artifacts they generate or use, a candidate list of terms is generated. Each term is then classified into object, subject, verb or state. The fourth step is to describe the meaning of each term —define its notion and impact—, being

[9] LEL authors state that the impact, formerly known as behavioral description, describes the "*connotation*, that is., and additional meaning of a word" [18]. From a semiotic point of view, however, the use of the term *connotation* in this sense is not accurate, and thus will not be used in this paper.

careful so as to respect the the principles of closure and minimal vocabulary. This step typically unveils additional terms to be included in the lexicon, and which undergo a similar process. In the last two steps, the lexicon is verified by inspection and validated by the stakeholders. As with scenarios, the lexicon is written in natural language, which makes it easy for non-experts to understand, question, and validate. The lexicon is also represented as a hypertext, which makes it easy to navigate between any two related terms.

In the context of the semantic web, there is a growing need to represent the semantics of the applications [4]. The need is fully met by the LEL, which provides both the meaning and relationships among its terms. However, the fact that the LEL is coded in natural language format prevents is from being automated by machines. Ontologies, in our understanding, are the formalization of the concepts captured by the LEL in a machine processable language, e.g., DAML+Oil or OWL [15, 19]. Readers who are interested in deriving formal ontologies may refer to [5], which describes how to derive a machine-processable ontological representation from the lexicon.

We argue that the quality of the resulting lexicon depends highly on the experience and domain knowledge of its builders. Moreover, in following a semiotic engineering approach to HCI, we would like the meaning descriptions to reflect the designers' assumptions about the users' knowledge and expectations of the domain and application. As we will see in the next sections, these assumptions may be captured in the form of answers to questions related to the users' most frequent doubts. In this context, this paper proposes to extend LEL to enhance its capacity as a communicative artifact among team members, and as a concrete resource for model-based design of interactive artifacts.

It is important to note that we do not suggest to use LEL in isolation. Instead, we propose to use it to complement scenarios [6]. Scenarios give all stakeholders an understanding of the domain and of the application being designed, in a contextualized manner. However, we felt the need to centralize the definitions of goals, tasks, agents and objects, because if they are scattered throughout scenarios, problems of inconsistency and incompleteness may prevent designers to build an adequate conceptual model of the domain (and later of the solution). This would make it harder to engineer the signs that will be conveyed to users through the user interface. Designers need both the contextualization of the scenarios and the different perspectives that LEL gathers together for each sign.

4 Communication-oriented concerns in model-based interaction design

Although LEL is a useful tool for representing domain concepts and their interrelationships, we want to shift the focus to communication-oriented concerns involved in user-system interaction. These concerns were explored in previous work on communicability evaluation [25] and help systems design [29]. In this section, we outline the communication-oriented concerns that, we believe, need to be represented throughout the design process.

Traditional model-based approaches to user interface design are rooted in cognitive theories or ergonomic approaches, which focus on the human interacting with the system image [21]. Our work is based on semiotic engineering [9], which takes on a communicative perspective to HCI, viewing the user interface as a metamessage sent from designers to users. This message is created in such a way as to be capable of exchanging messages with users, i.e., allowing human-system interaction. In semiotic engineering, the high-level message sent from the designer to users can be paraphrased as follows [9]:

> *"Here is my understanding of who you [users] are, what I've learned you want or need to do, in which preferred ways, and why. This is the system that I have therefore designed for you, and this is the way you can or should use it to fulfill a range of purposes that fall within this [my] vision."*

Because semiotic engineering brings to the picture designers themselves as communicators, we need to provide tools to better support them in this communicative process, ultimately via the user interface. One way to accomplish this is by investigating communication problems users experience when interacting with an application. These problems may be expressed by their frequent doubts and needs for instructions and information, i.e. help content. In the literature about help systems, we find that users would like to receive answers to their most frequent doubts, as summarized in Table 1 [1,28].

Table 1. Taxonomy of users' frequent doubts.

Types of Questions	Sample Questions
Informative	*What kinds of things can I do with this program?*
Descriptive	*What is this? What does this do?*
Procedural	*How do I do this?*
Interpretive	*What is happening now? Why did it happen? What does this mean?*
Navigational	*Where am I? Where have I come from? Where can I go to?*
Choice	*What can I do now?*
Guidance	*What should I do now?*
History	*What have I done?*
Motivational	*Why should I use this program? How will I benefit from using it?*
Investigative	*What else should I know? Did I miss anything?*

We propose that the questions related to the users' most frequent doubts be explicitly addressed throughout the various design stages, starting from requirements elicitation (and the construction of the LEL). Our ultimate goal is to provide designers with a comprehensive understanding of the domain and of the effects of their design decisions on the final product (i.e. the user interface), as viewed from a user's point-of-view. By using these potential user questions, we help designers to reflect while they make important design decisions, engaging in reflection-in-action [27]. At the same time, we would want to encourage the representation of these design decisions, thus building the design rationale of the envisaged application.

From the users' point-of-view, we make use of communicability and help utterances that allow users to better express their doubts during interaction [29] (Table

2). By anticipating users' doubts during design, the team members will be better equipped to deal with the users' communicative needs, either by designing applications that avoid interaction breakdowns altogether, or by giving users better chances for circumventing them [31].

Table 2. Communication-oriented utterances related to users' doubts during interaction breakdowns.

Original Communicability Utterances	(Additional) Help Utterances
What's this?	How do I do this? (Is there another way to do this?)
What now? (What can I do? What should I do? Where can I go?)	What is this for? (Why should I do this?)
What happened?	Whom/What does this affect?
Why doesn't it (work)?	On whom/what does this depend?
Oops!	Who can do this?
Where is *it*?	Where was I?
Where am I?	
I can't do it.	

An answer to the "What's this?" communicability utterance can be easily found in the *notion* part of each LEL term. For other utterances, however, the answers are not so straightforward, and depend highly on how meaning is described as an *impact* in LEL. In the next section, we describe how LEL definitions may include key elements needed in our design approach.

5 Enhancing LEL to provide a communicative artifact for design team members

In the previous sections, we have argued for the importance of providing a common vocabulary to promote the stakeholders' shared understanding of the domain using the LEL, and how relevant design decisions should be addressed and represented from a communication-oriented standpoint while building the design models. In this section, we explore how these two approaches may be coupled, i.e., how the answers to important design decisions can be recorded as part of the LEL, making it easier to take advantage of them in later design and specification stages.

Taking into consideration the communication-oriented concerns described in the previous section, we propose to enhance the LEL to incorporate the various communicative dimensions related to each concept or relationship. By doing so, we aim not only to create consensus among team members, but also to provide solid grounds for engineering the user interface sign systems that will minimize the effects of interaction breakdowns.

To show how our approach can be put to practical use, we briefly describe a case study we've developed: a system for managing conference submissions and reviews. Before building LEL, we felt the need for some guidance in identifying the first relevant signs. Inspired by traditional HCI work, we decided to start by building scenarios describing some of the users' roles, goals and tasks (Fig. 3). From the users' roles, we identified candidate roles (subjects in LEL), and from the goals and tasks we extracted a first set of verbs and objects.

Scenario 1. PC chair assigns submissions to reviewers. *The **deadline** for the ABC 2004 **conference** has arrived, and Mark, the **PC chair**, needs now to start the **reviewing** process. First he **assigns** the **submissions** to the **reviewers**, based on the **maximum number of submissions** each **reviewer** has determined, as well as on the expertise level of each **reviewer** with respect to the**conference topics**. He would like to have at least 3 **reviews** of each **submission**. To avoid having problems of fewer **reviews**, he decides to **assign** each **submission** to at least 4 **reviewers**. [...] One month later, Mark **receives** the **reviews** and must now **decide** upon the **acceptance** or **rejection** of each **submission**. Since there are a few **borderline submissions**, whose **grades** do not make clear whether it should be accepted or rejected, he decides to **examine** the **distribution of submissions per conference topic**. In doing so, he decides, from among **submissions** with similar **ratings**, those that will **ensure** some **diversity** in the **conference program**. However, this is not enough to **decide** about the **acceptance** of all **submissions**, and thus he **assigns** the remaining cases to additional **reviewers**, asking them for a quick **response**.*

Scenario 2. Reviewer judges submissions. *John, an HCI **expert**, **accepts** Mark **invitation** to become a **reviewer** for ABC 2004. He tells Mark that he will only be able to **review 3 submissions**, though. To help Mark with the **submissions assignment**, he **chooses** from among the **conference topics** those he wishes to **review**, i.e., in which he is an expert and interested. [...] He **receives** 4 **submissions** (one more than he'd asked for), but decides to **review** them all. He carefully **reads** every **submission**, and **grades** them according to the **form** Mark gave him, with the **criteria** of: **originality**, **relevance** to ABC 2004, **technical quality**, and **readability**. For the **submissions** that he **judged** acceptable, he **makes** some **comments** that he thinks will help **authors** to **prepare** the **final version**. For the **submission** he thinks must be **rejected**, his **comments** suggest **improvements** in the **work** itself, for future **submissions**.*

Fig. 3. Sample scenarios, describing user roles, the corresponding goals and tasks, and highlighting the candidate LEL signs in boldface.

By coupling LEL's basic elements — object, subject, verb and state— with communicability utterances, we allow design team members to thoroughly represent and understand the domain concepts from a user's point-of-view. At later design stages, designers may also use it to reflect on how the application should support users' tasks in this domain [27]. For each pair <element, utterance>, we suggest the identification of key elements that are needed to respond to the corresponding utterance. These questions work with LEL in a way analogous to the systematic questioning of scenarios proposed in [7]. Tha major difference is that the questions we use are grounded on users' most frequent doubts.

In the following, we relate the possible kinds of answers to each pair <element,utterance>, as well as the elements designers should try to include in their phrasing in order to provide such answers (Tables 3 to 6).

Table 3. Communicative utterances and suggested content for the description of LEL subjects.

subject	elements included in the sign meaning	comm. utterances
basic notion	13. what goals the subject {may | must | must not} achieve;	*What's this?*
		What's this for?
impact	14. which goal(s), task(s) and action(s) are available;	*How do I do this?*
		Why should I do this?
	15. what task sequences (are assumed that) the subject will prefer for each goal	*What now? (What can I do?)*
	16. breakdowns that hinder the performance of an action or task, or the achievement of a goal	*What happened?*

Table 4. Communicative utterances and suggested content for the description of LEL objects.

object	elements included in the sign meaning	comm. utterances
basic notion	17. object type, with respect to a generalization/specialization hierarchy of object-signs;	*What's this?*
	18. object composition, with respect to a partonomy of object-signs and a set of attribute-signs	
impact	19. which goal(s) {produce | destroy | modify | require } the object;	*What's this for?*
	20. which task(s) or action(s) {produce | destroy | modify | require } the object, and why (associated with which goal)	
	21. which subject(s) {may | must | must not} { create | destroy | modify | view } the object	*Who can do this?*

Table 5. Communicative utterances and suggested content for the description of LEL verbs.

verb	elements included in the sign meaning	comm. utterances
basic notion	22. subtasks or subordinate atomic actions;	*What's this?*
	23. what objects are {produced | destroyed | modified | required}	

impact	24. subjects who {may | must | must not} achieve the goal;	*Who can do this?*
	25. subjects who {may | must | must not} perform the action or task	*(I can't do it.)*
	26. associated user goal(s);	*What's this for?*
	27. reasons for choosing this task or action over another that achieves the same goal(s)	*Why should I do this?*
	28. task or action sequences available for achieving the goal	*How do I do this?* *Is there another way to do this?*
	29. possible outcomes of the action;	*What happened?*
	30. for outcomes that may represent a breakdown, actions for circumventing it	
	31. subjects affected by the achievement of the goal or performance of the task or action;	*Whom/What does this affect?*
	32. the possible resulting status of the objects after the goal, task or action	

	33. preconditions for performing the action or task, or for achieving the goal;	*On whom/what does this depend? (I can't do it.)*
	34. subjects that restrict the achievement of the goal or performance of the task or action;	
	35. the necessary status of the objects before the goal, task or action	
	36. task sequence(s) necessary to reverse the action	*Oops!*

Table 6. Communicative utterances and suggested content for the description of LEL status.

status	**elements included in the sign meaning**	**comm. utterances**
basic notion	37. objects or subjects to which this status corresponds	*What's this?*
impact	38. tasks or actions that change this status	*What's this for?*
	39. how this status can be reached (through which task(s) or action(s))	*How do I do this?*
	40. explanation on how the current state was (or may have been) reached;	*Oops!*
	41. corrective measures to allow the user to reverse the effects of the task or action	

	42. how to change the status to achieve a goal;	*What now?*
	43. for status that may represent a breakdown, suggested actions for circumventing it	*(I can't do it)*
	44. how the status was reached	*What happened?*
		Where was I?

In these tables, we have extended the LEL to include some of the communication-oriented utterances, but we have maintained the independence of the technological solution. To answer the remaining utterances (*Where is it?, Where am I?, Where was I?*, and *Why doesn't it?*), it is necessary to provide more detail with respect to the interactive solution. The level of detail represented in LEL, in our view, should reflect the design decisions that have been made at each design stage.

While modeling the tasks or designing the interaction, it should be possible to answer the following questions (Table 7):

Table 7. Descriptions of LEL elements to be completed during interaction design.

Subject

LEL	**elements included in the sign meaning**	**comm. utterances**
impact	45. at each interaction step, the current "position" relative to a goal	*Where am I?*
	46. at each interaction step, the previous step;	*Where was I?*
	47. how to go back to the previous step	

At a later stage, while designing the user interface, it should be possible to answer the following questions:

Table 8. Descriptions of LEL elements to be completed during user interface design.

Object

LEL	elements included in the sign meaning	comm. utterances
impact	48. widget that corresponds to the object;	*Where is it?*
	49. location of the widget at the user interface	

Verb

LEL	elements included in the sign meaning	comm. utterances
impact	50. the kind of feedback issued after triggering the action;	*Why doesn't it?*
	51. the associated goal(s) to detect mismatches between users' goals and user interface elements	

Many of the responses associated to the pairs <element, utterance> are interrelated. The hypertextual nature of LEL makes it easier for team members to traverse from one concept to related questions in another concept, using the utterances as a navigation aid [18]. This mechanism is analogous to the layering technique used in the minimalist approach [12] and to the help access mechanisms proposed in [29,30].

Table 9 presents a sample of the enriched LEL for the conference management system described in the aforementioned scenarios.

Table 9. Sample of the enriched LEL for the conference management system[10].

Object: Submission

LEL	elements included in the sign meaning	comm. utterances
basic notion	52. A document describing a research work that is submitted by an author to be considered for publication in the conference.	*What's this?*
	53. Is reviewed with respect to quality.	
	54. May be accepted or rejected.	
impact	55. PC chair must assign submissions to adequate reviewers.	*What's this for?*
	56. PC chair must decide about acceptance of borderline submissions, either by assigning submissions to additional reviewers or by checking for diversity of submissions with respect to conference topics.	*Who can do this?*
	57. Reviewer tells PC chair how many submissions he'd be willing to review, so that he doesn't receive too many submissions.	
	58. Reviewer grades submissions to review.	
	59. PC chair ranks submissions according to reviews.	

[10] For reasons of clarity, these tables do not show the hypertext links. As in the original LEL, if any LEL sign A is found in the meaning of the current sign B, A would be marked as hypertext link to the LEL definition of A.

Subject: Reviewer

LEL	elements included in the sign meaning	comm. utterances
basic notion	60. Expert in some of the conference topics.	*What's this?*
	61. Responsible for reviewing submissions.	*What's this for?*
impact	62. May set number of desired submissions to review.	*What can I do?*
	63. May define expertise and expectations with respect to keywords/topics, to review only submission for which you are an expert.	
	64. Must grades and comment submissions according to their quality.	
	65. May need to decline an assignment due to conflict of interest or lack of knowledge.	*What happened?*

Verb : Review (submission) [11]

LEL	elements included in the sign meaning	comm. utterances
basic notion	66. To evaluate the quality of the submission.	*What's this?*
	67. To comment on the content of the submission to guide authors in preparing the final version, if the submission is acceptable, or a future submission, if it is unacceptable.	*What's this for?*
impact	68. Reviewers must review the submissions assigned to him.	*Who can do this?*
	69. Own authors and interested parties must not review the submission.	*(I can't do it.)*
	70. Non-experts should not review the submission.	
	71. No one may review a submission not assigned to him.	
	72. To help the PC chair in deciding on the acceptance or rejection of submissions.	*What's this for?*
		Why should I do this?
	73. There must be grades to the following criteria: originality, relevance to conference, technical quality, and readability.	*How do I do this?*
		Is there another way to do this?

[11] A verb in LEL typically corresponds to a goal, task or action, but we define it in terms of the objects it manipulates.

74. The PC chair decisions about acceptance or rejection depend on the reviews.	*Whom/What does this affect?*
75. A review may be completed and sent in time, or may be late or missing.	
76. The PC chair is responsible for assigning submissions for reviewers to review.	*On whom/what does this depend? (I can't do it.)*
77. If the reviewer makes a mistake in the review, he needs to be able to modify or destroy it.	*Oops!*

By exploring the answers to the questions related to each LEL element from the users' standpoint, designers not only move towards achieving a shared understanding of the domain and how the application should support the users, but also are able to envisage the consequences of their design decision with respect to the user's future interactive exchanges with the application. Also, by doing so designers are developing a large portion of the help content for the final product *pari passu* the design decisions [30]. We believe this may facilitate not only the application evolution, but also the generation of user interfaces for multiple platforms and devices.

From the responses to the communication-oriented questions, designers may then proceed to modeling the application. Fig. 4 illustrates a possible schema for modeling the designers' concerns [29] as related to the communication-oriented questions.

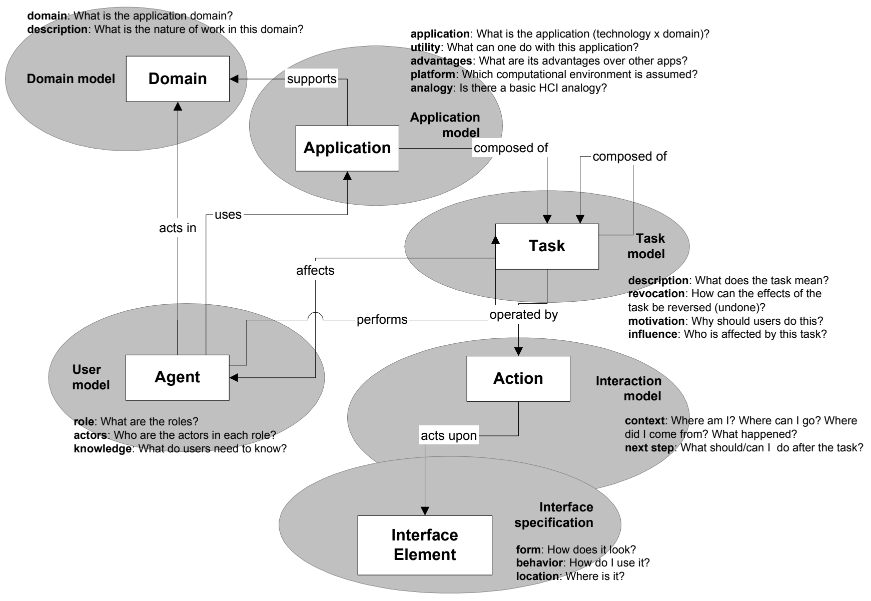

domain: What is the application domain?
description: What is the nature of work in this domain?
Domain model
Domain
supports
application: What is the application (technology x domain)?
utility: What can one do with this application?
advantages: What are its advantages over other apps?
platform: Which computational environment is assumed?
analogy: Is there a basic HCI analogy?
Application model
Application
composed of
composed of
uses
acts in
Task
Task model
affects
description: What does the task mean?
revocation: How can the effects of the task be reversed (undone)?
motivation: Why should users do this?
influence: Who is affected by this task?
performs
operated by
User model
Agent
Action
Interaction model
role: What are the roles?
actors: Who are the actors in each role?
knowledge: What do users need to know?
acts upon
context: Where am I? Where can I go? Where did I come from? What happened?
next step: What should/can I do after the task?
Interface specification
Interface Element
form: How does it look?
behavior: How do I use it?
location: Where is it?

Fig. 4. Schema for representing information in model-based design of human-computer interaction.

From a first version of this schema, HCI designers may then proceed into detailed interaction modeling [2,3] and storyboarding, whereas software designers have resources to specify the system's functional aspects.

6 Concluding Remarks

In this paper, we have described a communication-oriented design approach that brings together a technique for eliciting requirements and a design method driven by users' frequent doubts. Our goal was twofold: to create a shared understanding of the domain and how the application should support users in that domain, and to provide resources (and possible the underlying design rationale) for designing the interaction and engineering the user interface signs.

We illustrated the proposed approach by briefly describing some aspects of a case study system for conference submission and reviewing. During the case study, we noticed at least two important benefits of the proposed approach. First, the communication-oriented utterances, coupled with the elements to be included in the sign meaning (described in the tables at the previous section), helped designers inspect LEL, uncovering additional signs and refining previously-defined meanings of existing signs. Second, by explicitly representing the communicative concerns associated with each domain concept, design team members succeeded in forming a comprehensive vision of the domain and the application, and could thus envisage alternative technological solutions at the users' workplace. The case study described in this paper is still underway, and we plan to evaluate the communicability of the resulting application, and also a usability inspection to compare it with an existing application of a similar kind.

To gather stronger evidence about the advantages of this approach, we are currently developing multiple case studies, in the following domains: web content publication and location-based instant messaging in mobile devices. One of the issues we want to explore is whether the LEL structure or its classification should be changed to better accommodate the communicative concerns and the evolution of each concept's definition during different design stages, to capture the underlying design rationale and to provide different levels of focus and detail to address the relevant design concerns at each moment. The reason for investigating whether LEL structure should be changed is that, in our case study, at times we were tempted to structure LEL's descriptions according to users' goals and tasks, as in common HCI practice. Also, we felt that some elements do not fit well into LEL's classification, such as "expertise" or "submission deadline". We intend to analyze in the future whether modifiers and constraints should also receive a first-class status in LEL and thus be considered relevant signs with their own set of communication-oriented questions. For now, we have treated them as generic signs, for which the only associated question is "What's this?".

As future work, we intend to elaborate a set of guidelines for deriving communication-oriented interaction models [2] and for engineering user interface signs [9] from the enhanced LEL. In addition, we want to investigate the benefits of adopting the approach described in this paper in the design of an adaptive system, by deriving formal ontologies and explicitly incorporating to these systems the users' beliefs, goals, and plans.

Acknowledgments

Simone D. J. Barbosa, Maíra Greco de Paula and Karin Breitman would like to thank CNPq for providing financial support to this work. Simone D.J. Barbosa, Milene Selbach Silveira and Maíra Greco de Paula thank their colleagues at the Semiotic Engineering Research Group at PUC-Rio for many discussions that have contributed to this work.

References

1. Baecker, R.M. et al. (1995). *Readings in Human-Computer Interaction: toward the year 2000*. San Francisco: Morgan Kaufmann Publishers, Inc.

2. Barbosa, S.D.J.; de Souza, C.S. ; Paula, M.G. (2003) "The Semiotic Engineering Use of Models for Supporting Reflection-In-Action". *Proceedings of HCI International 2003*. Crete, Greece.

3. Barbosa, S.D.J; Paula, M.G. (2004) "Adopting a Communication-Centered Design Approach to Support Interdisciplinary Design Teams". *Bridging the Gaps II: Bridging the Gaps Between Software Engineering and Human-Computer Interaction*, ICSE 2004 workshop, Edinburgh, Scotland.

4. Berners-Lee, T.; Hendler, J.; Lassila, O. (2001) "The Semantic Web", Scientific American, May 2001. Available online at: http://www.scientificamerican.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&catID=2

5. Breitman, K. and Leite, J. (2003) Ontology as a Requirement Engineering Product .In: *11th IEEE International Requirements Engineering Conference*. Monterey Bay, California, USA, pp. 309-319.

6. Carroll, J.M. (ed., 1995) *Scenario-based Design: Envisioning Work and Technology in System Development.* New York, NY. John Wiley and Sons.

7. Carroll, J.M.; Mack, R.L.; Robertson, S.P.; Rosson, M.B. (1994) "Binding Objects to Scenarios of Use", *International Journal of Human-Computer Studies* **41**:243-276. Academic Press.

8. Danesi, M., Perron, P. (1999) *Analyzing Cultures: An Introduction and Handbook*, Indiana University Press.

9. de Souza, C.S. (in press) *The Semiotic Engineering of Human-Computer Interaction*. The MIT Press.

10. de Souza, C.S. (in press) Semiotic engineering: switching the HCI perspective from producing to introducing high-quality interactive software artifacts. *Interacting with Computers* **16**-6. Forthcoming.

11. Eco; U. (1979) *A theory of Semiotics*, Bloomington, IN: Indiana University Press.

12. Farkas, D.K. (1998) "Layering as a Safety Net for Mini-malist Documentation". In J.M. Carroll (ed.) *Minimalism Beyond the Nurnberg Funnel*. The MIT Press, Cambridge.

13. Fensel, D. (2001) Ontologies: a silver bullet for knowledge management and electronic commerce, Springer.

14. Gruber, T.R.(1993) "A translation approach to portable ontology specifications", *Knowledge Acquisition*, 5 (2): 199-220

15. Hendler, J.; McGuiness, D. (2000) "The DARPA agent Markup Language", *IEEE Intelligent Systems*, 16 (6), 2000. pp.67-73.

16. Kammersgaard, J. (1988) "Four different perspectives on Human-Computer Interaction", International Journal of Man-Machine Studies 28:343-362, Academic Press.

17. Kaplan, G.; Hadad, G.; Doorn, J.; Leite, J.C.S.P. (2000) "Inspección del Lexico Extendido del Lenguaje". *Proceedings of the Workshop de Engenharia de Requisitos, WER'00*. Rio de Janeiro, Brasil.

18. Leite, J.C.S.P.; Franco, A.P.M, (1992) "A Strategy for Conceptual Model Acquisiton". *Proceedings of the IEEE International Symposium on Requirements Engineering*, IEEE Computer Society Press, Pags. 243-246, San Diego.

19. McGuiness, D.; Harmelen, F. (2003) *OWL Web Ontology Overview*, W3C Working Draft 31 March 2003.

20. Mullet, K., and Sano, D. (1995) *Designing Visual Interfaces: Communication-Oriented Techniques*, SunSoft Press, Mountain View, CA.

21. Norman, D. e Draper, S. (eds., 1986) *User Centered System Design*. Hillsdale, NJ. Lawrence Erlbaum.

22. Oppermann, R. (1994) *Adaptive user support : ergonomic design of manually and automatically adaptable software*. Hillsdale, N.J. : Lawrence Erlbaum Associates.

23. Paternò, F. (2000) *Model-Based Design and Evaluation of Interactive Applications*, London, Springer-Verlag.

24. Peirce, C.S. (1931-55) *Collected Papers*. Cambridge, Ma. Harvard University Press. (excerpted in Buchler, Justus, ed., Philosophical Writings of Peirce, New York: Dover, 1955).

25. Prates,R.O., de Souza, C.S., Barbosa, S.D.J. (2000) “A Method for Evaluating the Communicability of User Interfaces”. *ACM Interactions*, 31–38, Jan-Feb 2000.

26. Preece, J., Rogers, Y., and Sharp, H. (2002) *Interaction design: beyond human-computer interaction*, John Wiley & Sons, New York, NY.

27. Schön, D. (1983) *The Reflective Practitioner: How Professionals Think in Action*, New York, Basic Books.

28. Sellen, A.; Nicol, A. (1990). Building User-Centered On-line Help. In Laurel, B. *The Art of Human-Computer Interface Design*. Reading: Addison-Wesley.

29. Silveira, M.S.; Barbosa, S.D.J.; de Souza, C.S. (2001) Augmenting the Affordance of Online Help Content. *Proceedings of IHM-HCI 2001*, Lille, Springer-Verlag.

30. Silveira, M.S.; Barbosa, S.D.J.; de Souza, C.S. (2004) Model-based design of online help systems. Proceedings of CADUI 2004.

31. Winograd, T. and Flores, F. (1986) Understanding Computers and Cognition: A New Foundation for Design, Addison-Wesley, Reading, MA.

Discussion

[Fabio Paternò] There is a tool that takes scenario and associates with objects and with tasks. Do you think that your method can be supported by a tool able to derive more structured information?

> [Simone D.J. Barbosa] The current approach is merely oriented for a designer analysis. We are not thinking about tool support.

[Philippe Palanque] Where does your taxonomy, presented at the beginning of the talk, comes from?

> [Simone D.J. Barbosa] This comes from work on help systems

[Philippe Palanque] So it does not come from a semiotic engineering analysis?

> [Simone D.J. Barbosa] No, but Semiotic Engineering would be useful to build this kind of taxonomy

[Ann Blandford] You said there is no such thing as a typical user. How do you deal with the usability across users?

[Simone D.J. Barbosa] What we are reasoning about is what is expected of users and how those expectations are communicated to them.

A Seamless Development Process of Adaptive User Interfaces Explicitly Based on Usability Properties

Víctor López-Jaquero[†‡], Francisco Montero[†‡], José P. Molina[†‡], P. González[†], A. Fernández-Caballero[†]

[†] Laboratory on User Interaction & Software Engineering (LoUISE)
University of Castilla-La Mancha, 02071 Albacete, Spain
{ victor | fmontero | jpmolina | pgonzalez | caballer }@info-ab.uclm.es

[‡] Belgian Laboratory of Computer-Human Interaction (BCHI)
Université Catholique de Louvain, 1348 Louvain-la-Neuve, Belgium
{ lopez | montero | molina}@isys.ucl.ac.be

Abstract. This work is aimed at the specification of usable adaptive user interfaces. A model-based method is used, which have been proved useful to address this task. The specification created is described in terms of abstract interaction objects, which are translated into concrete interaction objects for each particular platform. An adaptive engine is also proposed to improve the usability at runtime by means of a multi-agent system.

A seamless process for adaptation development

Currently different interaction paradigms are emerging due to several factors, such as ubiquitous access to information, the consideration of different user expertise levels, accessibility criteria or the wide range of interaction devices with specific capabilities (screen size, memory size, computing power, etc). In this paper a method is introduced for the specification of user interfaces of highly interactive systems with the capability of self-adapting to the changes in the context-of-use.

To fill the gap between model-based user interface development approaches and adaptive user interface frameworks, we propose enriching the usual model-based user interface development, to include, in a seamless manner, the development of the adaptation facilities required for adaptive user interfaces development. We propose a method for the development of adaptive user interfaces called *AL-BASIT* (Adaptive Model-Based User Interface Method), which extends usual model-based user interface development methods to support the development of adaptive user interfaces in a seamless way. Our proposal starts with requirements analysis to identify the tasks that will drive the design. Also user, physical environment and platform characteristics are collected to complete requirements analysis. In requirements analysis, use cases are used to identify the tasks and to establish a comprehensible channel of communication with the user, using an artefact understandable by the user and the designer. This stage is completed gathering the required data from the potential context-of-use for the application (user, platform and environment models). Analysis stage in aimed at the transformation of the requirements into a specification easier to handle, and usually in a more compact format. It also brings requirements analysis data closer to designer language. In our approach, we are using UML class diagrams to describe the domain model. To support human role multiplicity, we match each possible role a user can assume when using the user interface with the tasks they can perform. After analysis stage, design phase take place using the proposed tool. The design is based on the description of the identified tasks and their relationships with the domain elements they make use of. The task model is enriched describing the events to change from one task/action to another with the canonical abstract user interface tools [1]. From this data, an abstract user interface is generated which is independent of both modality and platform. Then, a translation is made to a concrete user interface (CUI) expressed in USIXML (http://www.usixml.org) user interface description language. The coordination between the CUI elements, the application functional core and the final running code is performed by means of connectors, as described in [2][3] This specification is adapted at runtime using a transformational approach. The adaptation engine reasons about the possible adaptation and preserves different usability properties according to the usability trade-off specified in terms of I* specification technique [4].

Conclusions

In this paper we have introduced a method for the development of adaptive user interfaces. It improves the usability of the system by adapting the user interface to the context-of-use at runtime. Thus, the user interface is adapted according to the changes in the context-of-use. For the design of adaptation engine, a multi-agent system is used. The goals of the agents in the multi-agent system are guided by the adaptation trade-off specified by the designer at design time using a goal-driven requirements notation: I*.

Acknowledgements

We gratefully acknowledge the support of the spanish PBC-03-003 grant and the SIMILAR network of excellence (http://www. similar.cc).

References

1. Constantine, L. *Canonical Abstract Prototypes for Abstract Visual and Interaction Design*. Proceedings of DSV-IS. Springer Verlag, LNCS 2844, 2003.
2. Lopez-Jaquero, V., Montero, F., Fernandez-Caballero, A. Lozano, M.D. *Towards Adaptive User Interface Generation: One Step Closer To People*. 5th International Conference on Enterprise Information Systems, ICEIS 2003. Angers, France, 2003.
3. Lopez-Jaquero, V., Montero, F., Molina, J.P., Fernandez-Caballero, A., Gonzalez, P. *Model-Based Design of Adaptive User Interfaces through Connectors*. DSV-IS 2003. Springer Verlag, LNCS 2844, 2003.
4. Yu, E. Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering' Proceedings of the 3rd IEEE Int. Symp. on Requirements Engineering (RE'97) Jan. 6-8, 1997, Washington D.C., USA. pp. 226-235.

Discussion

[Fabio Paternò] How do you specify the adaptive behavior of your system?

> [Victor Lopez-Jaquero] We use agents that exploit the specified rules selecting the more appropriate rules according to the current context of use. These agents include in their decision-making mechanism the XML specification of the UI.

[Willem-Paul Brinkman] You mention that you want to conduct user tests to evaluate your ideas. How do you envision you will do that?

> [Victor Lopez-Jaquero] Conducting a series of small experiments to study each individual issue separately.

[Willem-Paul Brinkman] This can become a very extensive task. Would you consider a case study instead?

> [Victor Lopez-Jaquero] We are considering a case study, of course, but you can just validate a small set of issues at a time, because otherwise, interdependecies can make evaluating the result an imposible task.

[Philippe Palanque] On one of your slides you said that you augmented CTT. Could you please tell us more about this augmentation?

> [Victor Lopez-Jaquero] We mainly added (canonical) actions to the transitions between the tasks in the task model to allow the specification of the dialogue.

More principled design of pervasive computing systems

Simon Dobson

Department of Computer Science, Trinity College, Dublin IE
simon.dobson@cs.tcd.ie

Paddy Nixon

Department of Information and System Sciences, University of Strathclyde, Glasgow UK
paddy@cis.strath.ac.uk

Abstract. Pervasive computing systems are interactive systems in the large, whose behaviour must adapt to the user's changing tasks and environment using different interface modalities and devices. Since the system adapts to its changing environment, it is vital that there are close links between the structure of the environment and the corresponding structured behavioural changes. We conjecture that predictability in pervasive computing arises from having a close, structured and easily-grasped relationship between the context and the behavioural change that context engenders. In current systems this relationship is not explicitly articulated but instead exists implicitly in the system's reaction to events. Our aim is to capture the relationship in a way that can be used to both *analyse* pervasive computing systems and aid their *design*. Moreover, some applications will have a wide range of behaviours; others will vary less, or more subtly. The point is not so much **what a system does** as how what it does **varies with context**. In this paper we address the principles and semantics that underpin truly pervasive systems.

1 Introduction

Pervasive computing involves building interactive systems that react to a wide variety of non-standard user cues. Unlike a traditional system whose behaviour may be proved correct in an environmentally-neutral state space, a pervasive system's behaviour is intended to change along with its environments. Examples include location-based services, business workflows and healthcare support, gaming, and composite access control policies.

Building pervasive computing systems currently revolves around one of two paradigms: (a) *event-handling systems*, where behaviour is specified in terms of reactions to events; and (b) *model-based systems*, in which rules are applied over a shared context model. The former leads to fragmented application logic which is difficult to reason about (in the formal and informal senses); the latter leaves a large number of rules whose interactions must be analysed, a situation known to be quite fragile. In addition, the majority of these approaches are premised on snapshot views of the environmental state.

A truly pervasive system requires the ability to reason about behaviours beyond their construction, both individually and in composition with other behaviours. This is rendered almost impossible when a system's reaction to context is articulated only as code, is scattered across the entire application, and presents largely arbitrary functional changes.

From a user perspective the design of pervasive computing systems is almost completely about interaction design. It is vitally important that users can (in the forward direction) predict when and how pervasive systems will adapt, and (in the reverse direction) can perceive why a particular adaptation has occurred. The hypothesis for our current work is that **predictability in pervasive computing arises from having a close, structured and easily-grasped relationship between the context and the behavioural change that context engenders**. In current systems this relationship is not explicitly articulated but instead exists implicitly in the system's reaction to events. Our aim is to capture the relationship in a way that can be used to both *analyse* pervasive computing systems and aid their *design*.

In this paper we describe our rationale for taking a more principled approach to the design of context-aware pervasive computing systems and outline a system that encourages such an approach, focusing on its impact on interaction. Section 2 presents a brief overview of pervasive computing, focusing on the difficulties in composing applications predictably. Section 3 explores pervasive computing from first principles to articulate the underlying motivations and factors influencing system behaviour. Section 4 describes a more principled design approach base on these factors and how they impact the interface functionality of systems, while section 5 concludes with some open questions for the future.

2 Pervasive computing

Pervasive computing can broadly be defined as *calm* technology that delivers the correct service to the correct user, at the correct place and time, and in the correct format for the environment[1]. Context, viewed alongside this definition, is all the information necessary to make a useful decision in the face of real-world complexity. More specifically, context is central to the development of several related trends in computing: the increasing pervasiveness of computational devices in the environment, the mobility of users, the connectivity of mobile users' portable devices and the availability to applications of relevant information about the situation of use, especially that based on data from physical sensors.

2.1 Context

Historically, the use of *context* grew from roots in linguistics [2]. The term was first extended from implying inference from surrounding text to mean a framework for communication based on shared experience [3]. The importance of a symbolic structure for understanding was embraced in other fields such as [4,5,6] and subsequently developed from a purely syntactic or symbolic basis to incorporate elements of action, interaction and perception.

[7] divides context into two broad classes: *primary* context is derived directly from sensors or information sources, while *secondary* context is inferred in some sense from the primary context. A typical example is when GPS co-ordinates (primary context) are converted into a named space (secondary context) through a look-up process (inference).

More recently, in the setting of pervasive computing, **context awareness** was at first defined by example, with an emphasis on location, identity and spatial relationships [8,9]. This has since been elaborated to incorporate more general elements of the environment or situation. Such definitions are, however, difficult to apply operationally and modern definitions [10] generalize the term to cover "any information that can be used to characterize situation". Current work in the field addresses issues including:

- developing new technologies and infrastructure elements, such as sensors, middleware, communication infrastructures to support the capture, storage, management and use of context.
- increasing our understanding of form, structure and representation of context;
- increasing our understanding of the societal impact of these new technologies and approaches and directing their application;

A more detailed retrospective of the academic history of context can be found in [10,11].

For this paper we conjecture that as we move away from the *define by example* notions of context there is an increasing demand to establish the foundational models for context. For pervasive computing systems there remains two fundamental problems. Firstly, the centrality of context to the progress in the field of pervasive computing demands new views on the theoretical underpinnings of context. For example there is no widely accepted operational theory or formal definition of context. There is also an immediate problem of providing to application developers ways in which they can describe the context needs of their applications in manner that is orthogonal to the application or business logic of the application. The programming primitives, frameworks, and tools are still in their infancy.

3 The semantics of a context-aware system

3.1 What *is* context?

By **context** we mean the environment in which an application is executing. This might include the identity of a user, their location, the locations of other users, the device they are using, the information, task workflows they are involved in, their goals, strategies and so forth.

The intention of making a system context-aware is to allow the detailed behaviour of the application to adapt to context while keeping the overall behaviour constant: a messaging application always delivers messages, but may deliver messages differently in different contexts. Interface modality [12] may not be purely a device issue: a system might adapt its mode of interaction on the same device for different circumstances (such as going from vision to voice on a handheld), or might choose to switch devices while maintaining the same interaction style (such as making use of a wall screens instead of a PDA for form input).

Context is not monolithic: a given context may be composed of a number of different facets. Moreover the facets available may change between different executions of a context-aware application, for example when a new location system is installed. This implies that context-aware systems have defaults for “missing” contextual parameters, and that there is some mechanism for making new parameters “useful” to a wide range of applications. We do not, for example, want a context-aware system to be tied to a particular kind of location system, but want the location systems available at run-time to be leveraged to their fullest extent. This is essential for incremental, open deployment.

3.2 Behaviour

As stated above, the *gross* behaviour of an application should remain the same - sorting algorithms remain sorting algorithms in whatever context they execute. However, the *detailed* behaviour may change with context - the sorting criteria, for example - and it is this detail, and the way **behaviour varies**, that we are seeking to capture when talking about the semantics of context-aware systems.

One way to view this is as follows. Behaviour can be captured as a function from inputs to outputs, with some of the inputs being captured during execution. Context provides additional inputs describing the environment in which the function is being evaluated. Two invocations of the same function with the same (external) inputs may result in different behaviours because of changes in context.

We can therefore regard contextual variation as changing the contextual inputs to an underlying “ordinary” function. In what follows, when we refer to “behaviour” and “behavioural change"”we mean this change in parameterisation rather than an explicit change in (the code of) the function being provided. (There is no loss of generality here as the parameter might encode a function description being passed to a universal evaluator.) From an implementation perspective this makes explicit the context on which the function's detailed behaviour depends.

3.3 Design

While much of the research on pervasive computing has its roots in the programming language and distributed systems communities, the chief design task is clearly one of interfacing - creating systems that are usable as part of a larger real-world activity. Moreover, the design task is both multimodal and dynamic.

Some pervasive computing systems will be unimodal, using a single device and interaction structure. However it is widely accepted that many will be multimodal, utilising a range of different devices across the lifetime of the interaction. This includes multiple users with different constraints.

If we consider the ability to deploy context-aware applications into a shared space, we must also deal with the interactions between these applications. This may involve negative aspects such as sharing device capabilities between applications, prioritising different (and possibly conflicting) decisions. However, there are also significant potentially positive aspects including the case where one application provides context for another that might not otherwise have been obtainable.

3.4 Behaviour variation

Some applications will have a wide range of behaviours; others will vary less, or more subtly. The point is not so much **what a system does** as how what it does **varies with context**.

Much of computer science has been devoted to the notion of *correctness* - that is, to ensuring that a system has a single behaviour, and that this is the behaviour the user wants. Context-aware systems attack the underlying assumption of a single behaviour that can be articulated, replacing it with the view that behaviour *should* change in different circumstances.

Arbitrary behavioural changes would be incomprehensible to users, and would make systems completely unusable. However, single behaviour is equally unattractive in that it prevents a system adapting to context. There is therefore a spectrum in the behavioural variation we are willing to accept (figure 1). In building a pervasive computing system we are looking for the "sweet spot" between adaptability and comprehensibility. However, this still leaves the issue of deciding *how* behaviour should change and *when* changes should occur.

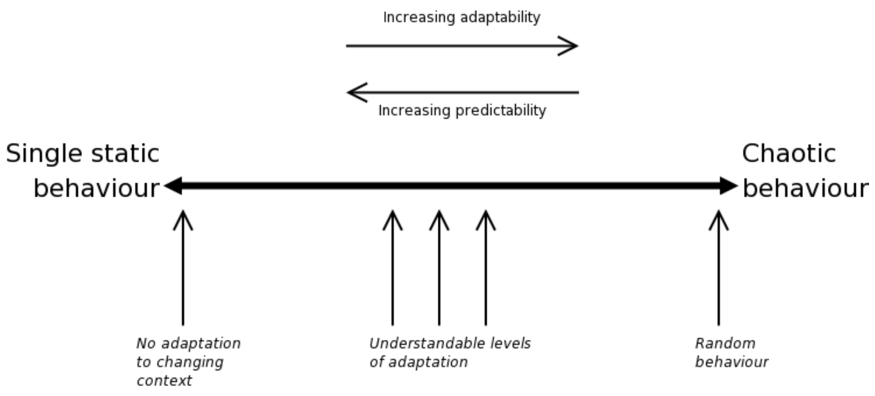

Increasing adaptability
Increasing predictability
Single static behaviour
Chaotic behaviour
No adaptation to changing context
Understandable levels of adaptation
Random behaviour

Fig. 1. The spectrum of behavioural variation.

An adaptive system adapts *to* something, and presumably adaptation happens when that something changes. Actually this turns out to be a little simplistic - adaptation may happen before or after a change - but the principle is valid. Since we are discussing context-aware systems, we can reasonably expect a system to adapt to changes in its context.

However, not all changes in context are significant or simple. A location-based service's behaviour will not typically be different at *every* different location, so not all location cues cause changes. Similarly location may not in itself be enough to define the system's behaviour without contributions from other aspects of context.

3.5 Describing the semantics

We might regard context as having a "shape" over which the system operates. The shape is multidimensional, defined by the various contextual parameters. The shape will also have identifiable "significant" points or areas that will have meaning to the user of the application, being perceived either as points where behaviour could (or should) change, or as areas in which behaviour could (or should) remain the same.

Not only do the significant points in the context define *when* behaviour can change, for a given application they will in many cases essentially define *what* new behaviour will be selected. To take a concrete example of a service providing tourist information, we expect the information being served both to change as we move and to remain relevant to the location we are in. The interface's adaptive behaviour of the system must therefore be closely related to the external world if that adaptation is to be intuitive.

This leads to our defining observation about developing a semantics for context-aware pervasive computing: in order for a pervasive computing system to be predictable to users, **the relationship between context and behaviour must be two-way and (largely) symmetric**. An application's behavioural variation should emerge "naturally" from the context that causes it to adapt, and that variation mandates that certain structures be visible in the model of context being used. It might only adapt to large-grained changes, placing it at the static end of figure 1; alternatively it may adapt to fine-grained changes, placing it at the dynamic end. The point is that the application's position in the spectrum is not selected *a priori* but emerges naturally from the shape of its context. If a context has a fine-grained structure it will support a highly adaptable application; conversely a highly adaptive application needs fine-grained context.

An *application*, in this view, consists of four elements:

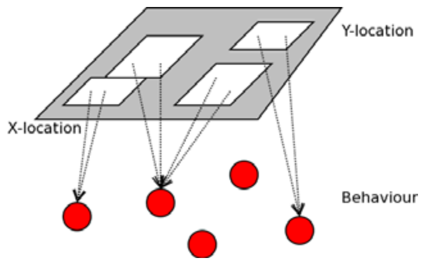
1. A baseline behaviour parameterised by a context
2. The context space with its significant points and shapes defined
3. The behavioural space with its own structures
4. A mapping matching changes in context to corresponding changes in behaviour

The first element is a standard program with adaptation hooks, and perhaps significant control structures for concurrency control and consistency maintenance. The third element describes the parameters used to control the program's adaptation. The second element describes the context expected by the application and the points at which this context forces or precludes adaptation. The fourth element describes the way in which the context adapts the program, matching significant changes in context to changes in behaviour.

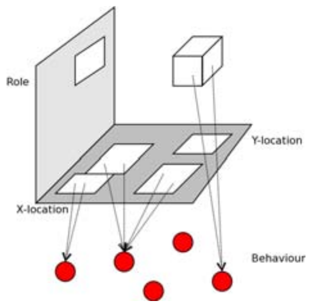
The issue of correctness reappears in another guise: instead of ensuring that a single behaviour is *implemented* correctly (and that the correct behaviour is implemented), we now need also to ensure that the behaviour *varies* correctly. The problem is not as bad as it might appear, however: if the underlying function is correct then the behaviour will be correct in *some* sense for each possible contextual parameter. The issue is one of the *appropriateness* of selecting a detailed behaviour in particular circumstances.

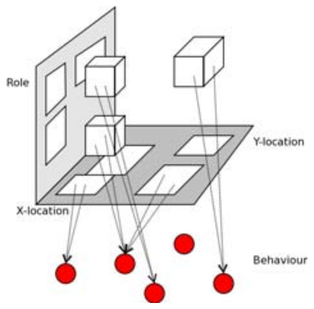
3.6 Towards more principled design

Making a function context-dependent essentially adds extra parameters to its definition. However, adding extra parameters in principle allows these additional degrees of freedom to affect the function's behaviour in arbitrary ways - a situation that is probably more general than is consistent with predictable variation. The challenge, then, is to provide additional parameters in such a way that their impact on the function's behaviour is constrained to be predictable, and follows (in some sense) the structure of the context.



(a) Location-dependent behaviour





(b) Adding role (c) Different roles in the same location

Fig. 2. Context dependence as parameter selection.

The essence of this problem is shown in figure 2. Figure 2(a) shows a function whose behaviour (the lower circles) depends on the location in which it is executed (the plane). Different regions of the plane map to the same behaviour, so the function observed by the user will be the same as they move within this region. Change in behaviour will only be observed when they move between regions.

Adding a extra contextual parameter, such as the person's role, adds another dimension to the behavioural space[12]. The behaviour may not vary in some locations for a change in role (figure 2(b)); alternatively there may be a change for some roles in some locations (figure 2(c)).

We claimed above that behaviour should only change "on cue" from context. This suggests that the change in role needs to be clear in the interface.

From a design perspective, it would also be attractive for the changed behaviour to depend structurally on the role and location: rather than making the change arbitrary, it should emerge naturally from the parameter space. This has three major advantages:

1. It simplifies the development of the adaptive controls by placing all adaptation functions in a single sub-system
2. It simplifies the development of the adaptive components by making the parameter space clearly defined and explicitly articulated
3. It provides a "closed form" of the system's context-aware behaviour for analysis

4 A mathematical model of principled design

The discussion above leads us to consider a model in which primary context conditions and constrains secondary context and behaviour. Formalising this notion leads to a semantics of context-aware systems.

We have adopted category theory as our semantic framework, for three reasons:

1. it is naturally extensible, so we can deal with an extensible collection of contextual parameters;
2. many of the well-known categorical structures suggest, at least intuitively, that they may be useful in structuring context awareness; and
3. our eventual goal is to develop programming abstractions for pervasive computing systems, and category theory's extensive use in language semantics may make this step easier.

However, our presentation here requires no understanding of the detailed mathematics of category theory: we focus here on the structural features of the approach and how it impacts the design and analysis of interface functionality. We refer the interested reader to [13] for a fuller treatment.

[12] Of course role is usually more complicated than this diagram suggests, but it will suffice for the purposes of illustration.

4.1 Modelling primary and secondary context

A **category** is a generalisation of the familiar approach of sets and functions between them. A category consists of a collection of **objects** and **arrows** between them. The most familiar category is the category of sets whose objects are sets and whose arrows are total functions between them. The arrows are constrained to be compositional and associative, and each object has an identity arrow.

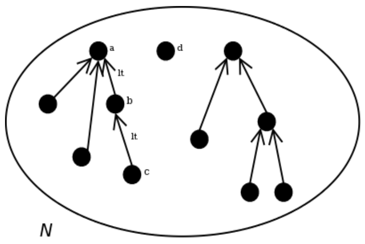


Fig. 3. Pointed structure within an object.

To each individual contextual parameter we assign an object in the category (*e.g.* a set) denoting the values the parameter can take. In a location system based on individual named spaces, for example, the "location" parameter would be represented by an object *N* whose points (elements in the case of a set) are the space names.
In many cases the elements of a parameter are themselves structured. A typical example (which occurs repeatedly) is a parameter structured as a partial order, pointed set or lattice, where each element can be "included" in at most one other (figure 3). For named spaces there is an arrow from the parameter object to itself, taking each space to its containing space or to itself if it is a "top" space. By repeatedly applying this operation we can navigate from a space up its container hierarchy. In figure 3 this means that the inclusion morphism *lt* takes space *c* to space *b*, space *b* to space *a*, and spaces *a* and *d* to themselves (we have omitted these arrows for clarity).

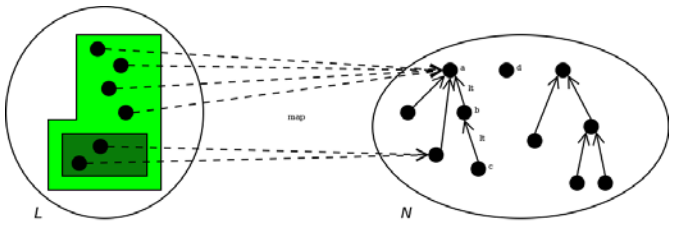


Fig. 4. Deriving secondary context.

Named spaces are probably secondary context, derived from a lower-level location system such as GPS. GPS can be modelled as an object *L* of GPS co-ordinate pairs. An obvious contextual constraint is the mapping between a GPS location and the named space containing it. We can represent this as an arrow *map:* $L \rightarrow N$ capturing the "map" (figure 4). It is important to realise that this is a *semantic* characterisation of what would implementationally be a lookup operation, the details which can be abstracted in the analysis.

Figure 4 makes clear the structural relationship between the two parameters; A region of L maps to an element of N in such a way that elements of the containing region in L must map to an element of N containing the original element. *map* is constrained to reflect the structure of one object in another, and it is this correspondence that preserves meaning in the interface.

4.2 Context as behaviour

Current context-aware systems are not uniform, in the sense that much of a system's behaviour is conditioned by information not held in a single context model. For the purposes of analysis it is simpler to regard context in the wider sense as the sole arbiter of behaviour: the system is functional with respect to its context. (We regard this as a sound implementation strategy too.)

The easiest way to accomplish this to include the "real" parameters to the external behaviour in the context. For a simple example, consider a wireless document system which delivers a set of documents depending on the user's location. The corpus of documents being managed can be represented as a contextual parameter (object) D whose elements are possible sub-sets of documents being served related by set inclusion.

We may now define an arrow *serve:* $N \rightarrow D$ which selects the set of documents to be served by the document system in each location. Although this arrow does not define behaviour in the normal sense of describing exactly what will happen, it *does* describe how the parameter passed to that behaviour will vary. We may therefore to some extent treat D as a proxy for the behaviour of the system and study how this "behaviour" changes with context.

4.3 Analysing the structure of behaviour

Even in this simple model there are a number of questions we may ask of the system. Key to these is an understanding of the way in which *different* contexts select the *same* behaviour. Using figure 4 as an example, there are a number of points in L that map to the same element of N. This is captured by the categorical notion of a **fibre**: given an element a of N the fibre of *map* lying over a is the sub-object of L that maps to a under *map*. Similarly the fibres of *serve* above represent the spaces in which the system will serve the same set of documents.

The significance of fibres is that they capture both those contexts in which the system will behave the same and the points at which that behaviour changes.

4.4 Compound context and behaviour

One of the advantages of category theory is that it has several strong notions of composition that can be used to create complex concepts by construction. A good example of this is the use of products of context and behaviour.

If C and D are contexts (objects) we can create a product context $C \times D$ whose elements are ordered pairs of elements from C and D respectively. Moreover there is an arrow between an element (i, x) and (j, y) if there is an arrow on C from i to j and an arrow on D from x to y.

Such products represent the compound state of the system: If we take N and another context P of people's identities, the compound context $P \times N$ represents a person in a named space. We can use this product contexts to contextualise behaviour in the normal way, by specifying an arrow *serve'*: $P \times N \rightarrow D$ defining how the documents available vary with identity and place. The risk here is that such behaviour will be arbitrary, in that there is no necessary relationship between the way behaviour changes with identity and the way behaviour changes with identity *and* location. In many cases we may wish to ensure that such a relationship is preserved.

If we have arrows *serveto*: $P \rightarrow D$ and *servein*: $N \rightarrow D$ we can model this by *constructing* the arrow *serve'* from the two more elementary arrows, in such a way that *serve'* preserves some of their features. For example, we might constrain *serve'* so that it always serves a set of documents that includes the set identified by *serveto* – location context may *broaden* the behaviour but always maintains the behaviour of *serveto* as a “core”. Conversely we might force *serve'* to never serve a larger set of documents than permitted by *serveto* – the underlying arrow specifies the “extent” of the behaviour. A third possibility is that location “adds nothing” to the behaviour, when *serve* defines the same behaviour as *serveto*. Similar arguments apply to *servein*.

These constructions allow us to potentially specify the constraints on complex behaviours in terms of simpler behaviours. This is important both for tackling the complexity of the system and ensuring its consistency. A user of *serve'* that preserves *serveto* as a core, for example, will be able to form a mental model in which (a) they can rely on a certain minimum behaviour everywhere, and (b) their location may add significant new documents. This consistency is vital to the usability of the system, and can be made a direct consequence of its categorical model.

Similar techniques can be used when contextualising a product context, where (for example) two behaviours B_1 and B_2 are combined to form a compound behaviour $B_1 \times B_2$ that specifies two aspects of the system independently. Again, composition of underlying arrows can be used to constrain the way in which behaviour varies.

4.5 Composition and conflict analysis

Pervasive computing almost implies dynamic composition, in that we expect mobile systems to be carried around by users and to “discover” resources as they move. This brings positive and negative possibilities: new capabilities may become available very easily, but systems may interact in undesired ways. A major challenge for analysis is to detect such conflicts.

In certain simple cases we can both detect conflicts and identify "safe" zones when two systems are composed. Suppose we have two systems with the same context and behaviour, described by two arrows $f,g : C \rightarrow D$: for the wireless document server these might be the public and private document servers. If we run both systems together, we may ask whether they will both serve the same document set for a given user and location. A categorical construction called an **equaliser** captures the sub-object C' of C in which f and g behave the same. If we can ensure that the system will remain in this region C', the systems may be composed safely; if it strays outside then the two systems diverge. Another possibility is to force g (for example) to serve as a core or extent of f.

In both cases the composition of systems is captured cleanly within the categorical model, and can be analysed using standard techniques. This may in turn lead to improved implementation techniques.

4.6 Designing "graspable" systems

Systems analysis, while important, is in many ways less interesting than systems design: we want to develop pervasive computing systems that are *usable and predictable by design*, using a model that both aids in this process and in the analysis of the results.

The fibre structure of arrows provides a powerful technique for designing systems as well as analysing them. Suppose we want to design our wireless document server so that it serves a set d_1 of documents in those places in the vicinity of a place n_1, and another set d_2 in the vicinity of n_2. If we constructed this system from scratch we would need to ensure that it responded to location events in the correct manner - an arduous testing process.

However, we can observe that the system behaves the same within a fibre - changes in context that remain within a fibre do not affect the behaviour. We need only ensure that all the places around n_1 lie in the fibre of d_1 to be convinced that the system will behave as required.

From a user perspective, in order to be predictable a change in behaviour must be accompanied by a perceptible change in the context that "makes sense" for the application at hand. Changes in behaviour occur when context moves between fibres. If we ensure that these changes correspond to external contextual cues that will convey the need for behavioural change to the user, then the user will be able to develop an appropriate mental model of the way in which the behaviour changes in response to context. The cues in the outside world are reflected exactly in the fibre structure of the model.

We claimed in section 4 that, in order for a pervasive computing system to be comprehensible, the relationship between context and behaviour needed to be largely symmetrical. It is this matching of fibre structure to external cues that captures this symmetry, either constructively (for design) or analytically (for analysis).

Although the matching of cues to fibre transitions is application-dependent and generally external to the model, it is sometimes possible to capture the cues within the structure of the category. If, for example, we can identify the context points at which behaviour should change, we can often identify the "internal" points where it should remain the same, corresponding to the fibre over the desired behaviour. These regions - sub-objects of the overall context - can have their behaviour described individually, with the "full" behaviour coming by composition in a way that will detect many conflicts automatically. This means that a user-centred design that identifies the adaptation points in the environment can be used directly to construct a mathematical description of the system being constructed, carrying usability concerns directly into the system model.

5 Conclusion

We have motivated using a more principled approach to the design and development of context-aware pervasive computing systems, and presented a formal approach that captures some of the essential driving forces in a natural and compositional way. We have shown how certain aspects of usability and predictability in the requirements for a pervasive computing system can be given a formal realisation within a system model suitable for use as a basis for analysis and design.

Perhaps more than any other potentially mainstream technology, pervasive computing requires that we take an automated approach to system composition and variation - the alternative would constrain deployment to constellations of devices and information sources that could be described *a priori*. This in turn means that we need to be able to state very precisely the way in which system behaviour varies. This is the point at which our work diverges from that in the ambient calculus[14] or bigraphs[15] - two very prominent and influential formal treatments of mobile systems - in that we sacrifice the precise characterisation of system behaviour in favour of broad-brush analysis. We also do not privilege location, regarding it as just one of the possible contextual parameters to be studied.

The obvious counter in this formulation is that the baseline behaviour needs to encapsulate all possible adaptations, which are then selected by context. While this is correct to an extent, we should differentiate between the abstract semantic model of a context-aware application and its concrete realisation. One would not necessarily pass context as a parameter to a function: it might be preferable to allow the function to access a shared context model, and provide some templated mechanism for this model to affect its behaviour. There are, however, serious engineering problems to be overcome in developing a programming model under this model.

Although we have not investigated it in this paper, a design approach such as we propose needs to be backed by an engineering methodology. In particular we have largely elided the way in which a designer would decide on the correct formulation for context and behaviour, or check that his choices relate correctly to the users' perceptions of the system. While traditional analysis and design methods can help address these problems, there is also a need to deploy detailed usability evaluations - possibly modified for pervasive computing - to inform the feedback loop. This is a subject that is outside our expertise but that we would be keen to explore further.

It seems unlikely that the techniques described are sufficient to address the full range of context-aware behaviours, so there is a major open question in the applicability of the techniques to real-world applications - something we are investigating at present. We are also addressing the limitation of the model to "immediate" context, where only the current situation (and not the past or possible future) affect behaviour. However, we believe that "closed form" expressions of context awareness are a key enabler for building the next generation of complex pervasive computing systems.

References

1. Weiser, M..The computer for the 21st century. Scientific American (1991)
2. Winograd, T. Architecture for context. Human Computer Interaction **16** (1994) 85-90
3. Minsky, M. A Framework for Representing Knowledge. In The Psychology of Computer Vision. McGraw Hill (1975)
4. Brooks, R. A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation **2** (1986)
5. A.Draper, B., Collins, R.T., Brolio, J., Hansen, A.R., Riseman, E.M. The schema system. International Journal of Computer Vision **2** (1989)
6. Bajcsy, R. Active perception. Proceedings of the IEEE **1** (1988) 996-1006
7. Salber, D., Dey, A., Abowd, G. The Context Toolkit: aiding the development of context-enabled applications. In Proceedings of the ACM Conference on Computer-Human Interaction, CHI'99. (1999) 434-441
8. Ward, A., Jones, A., Hopper, A. A new location technique for the active office. IEEE Personal Comunications **4** (1997) 42-27
9. Rodden, T., K.Cheverest, Davies, K., Dix, A. Exploiting context in HCI design for mobile systems. In Workshop on Human Computer Interaction with Mobile Devices. (1998)
10. Dey, A. Understanding and using context. Personal and Ubiquitous Computing **5** (2001) 4-7
11. Crowley, L., Coutaz, J., Rey, G., Reignier, P. Perceptual components for context aware computing. In Proceedings of Ubicomp 2002. (2002)
12. Calvary, G., Coutaz, J., Thevenin, D. A unifying reference framework for the development of plastic user interfaces. In Proceedings of EHCI'01. Volume 2254 of Lecture Notes in Computer Science., Springer Verlag (2001)
13. Dobson, S., Nixon, P. Towards a semantics of pervasive computing (just the category theory). Technical report, Department of Computer Science, Trinity College Dublin (To appear)
14. Cardelli, L., Gordon, A. Mobile ambients. In Nivat, M., ed. Foundations of software science and computational structures. Volume 1378 of LNCS.
Springer Verlag (1998)
15. Jensen, O.H., Milner, R. Bigraphs and mobile processes. Technical Report UCAM-CL-TR-570, University of Cambridge Computer Laboratory (2003)

Discussion

[Nick Graham] This is a semantic framework that is instantiated over a specific application. This seems to require the modeller to anticipate the possible contexts or compositions that may arise.

> [Simon Dobson] This is less a problem than with other approaches. In effect, we can define compositions without having to specify what kinds of things are being composed. This is sufficiently rich to allow interesting analyses.

There are a small set of composition operators that seem to recur frequently: although we have to select which operator to use when we encounter a new contextual parameter, we often don't need to know its details to do something meaningful.

[Helmut Stiegler] Category theory is all about commutative diagrams. You did not show any such examples, in which you can apply such diagrams. Do you have some ?

[Simon Dobson] Yes, we have them used. I suppressed them here on purpose. You will be able to find them in a technical report.

[Gerrit van Der Veer] How do the notions of "conflict" and "problem" relate to the framework ?

[Simon Dobson] These notions are not automatically specified, but have to be stated explicitly in order to reason about them.

Towards a new generation of widgets for supporting software plasticity: the "comet"

Gaëlle Calvary, Joëlle Coutaz, Olfa Dâassi, Lionel Balme, Alexandre Demeure

CLIPS-IMAG,
BP 53, 38041 Grenoble Cedex 9, France
{Gaelle.Calvary, Joelle.Coutaz}@imag.fr

Abstract. This paper addresses software adaptation to context of use. It goes one step further than our early work on plasticity [5]. Here, we propose a revision of the notion of software plasticity that we apply at the widget level in terms of comets. Plasticity is defined as the ability of an interactive system to withstand variations of context of use while preserving quality in use where quality in use refers to the ISO definition. Plasticity is not limited to the UI components of an interactive system, nor to a single platform: adaptation to context of use may also impact the functional core, it may have an effect on the nature of the connectors, and it may draw upon the existence of multiple platforms in the vicinity to migrate all or portions of the interactive system. A new reference framework that structures the development process of plastic interactive systems is presented to cover these issues. The framework is then applied at the granularity of widgets to provide the notion of a comet. A comet is an introspective widget that is able to self-adapt to some context of use, or that can be adapted by a tier-component to the context of use, or that can be dynamically discarded (versus recruited) when it is unable (versus able) to cover the current context of use. To do so, a comet publishes the quality in use it guarantees, the user tasks and the domain concepts that it is able to support, as well as the extent to which it supports adaptation.

1 Introduction

Mobility coupled with the development of a wide variety of access devices has engendered new requirements for HCI such as the ability of interactive systems to run in different contexts of use. By context of use we mean a triple <user, platform, environment> where the user denotes the archetypal person who is intended to use the interactive system; the platform refers to the hardware and software devices available for sustaining the user interaction; the environment describes the physical and social conditions where the interaction takes place. To master the diversity of contexts of use in an economical and ergonomic way, the *plasticity* property has been introduced [31]. Basically, plasticity refers to the adaptation to context of use that preserves the user's needs and abilities. For example, FlexClock [15] is a clock that expands or shrinks its user interface (UI) when the user resizes the window (Fig. 1). The time remains readable during and after the adaptation.

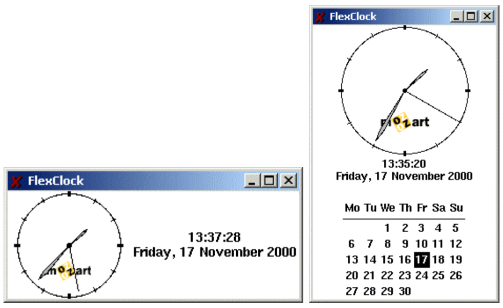


Fig. 1. FlexClock, an example of adaptation to the platform.

When applied at the widget level, the plasticity property gives rise to a new generation of widgets: the *comets* (COntext of use Mouldable widgETs). As a simple example, a set of radio buttons that shrinks into a combo box is a comet (Fig. 2).

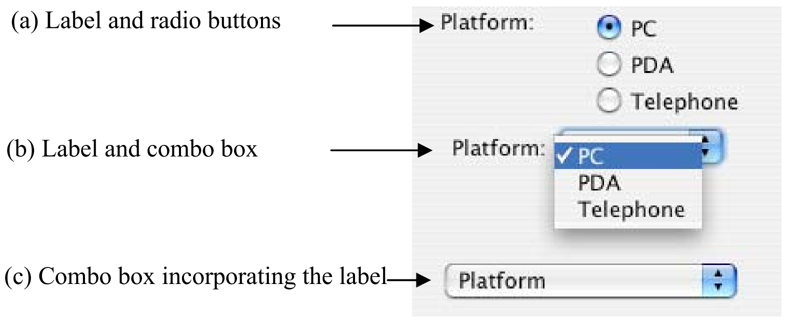


Fig. 2. Three graphical mockups supporting the same task "selecting one option among a set of options" through a) a label and radio buttons; b) a label and a combo box; c) a combo box incorporating the label. The example concerns the specification of the target platform (PC, PDA, telephone) for a centralized UI.

This paper presents our notion of comets. First we present new advances in plasticity to provide sound foundations for their elaboration. Then we focus on the comets per se considering both the design and run time perspective.

2 Foundations for comets: advances in plasticity

This section focuses on the lessons learned from experience that directly underpin the notion of comets. First, we propose a new definition for plasticity, then we examine the property from both a user and a system centered perspective.

2.1 A new definition of plasticity

Plasticity was previously defined as "the capacity of a user interface to withstand variations of context of use while preserving usability" [31]. Based on our experience, we have identified three reasons for revising the definition:

- In reality, plasticity is not limited to the UI components but may also impact the functional core. This occurs typically with services discovery. For example, because Bob has moved and is now in a place that makes a new service available, this service now appears on his PDA. The desktop is reshuffled (or tuned) to incorporate this new service and support an opportunistic interaction. Thus, the scope of the definition must be enlarged: plasticity must refer to the capacity of an *interactive system*, and not only to its UI, to adapt to the context of use;
- The current definition focuses on the preservation of usability only. As a result, utility is implicit. To make explicit the possibility to specify requirements concerning the preservation of functional (and not only non functional) properties (e.g., task accomplishment), the scope of the definition must be enlarged. To do so, we refer to *quality in use* instead of just usability. As defined by ISO [18], quality in use is based on internal and external properties (Fig. 3) including usability (Fig. 4);
- The definition is not operational enough. Due to ISO, the definition is now reinforced by a set of reference *characteristics* (factors), *sub-characteristics* (criteria) (Fig. 4) and metrics [19]. The framework QUIM (Quality in Use Integrated Map) [29] also contributes in this area by relating data, metrics, criteria and factors. A sound basis exists in HCI for usability ([1] [17] or more specifically [32] for dialog models).

Based on this new definition, an interactive system is said to be "plastic for a set of properties and a set of contexts of use" if it is able to guarantee these properties whilst adapting to cover another context of use.

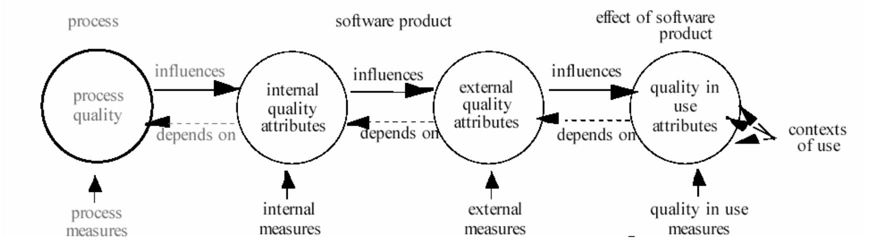
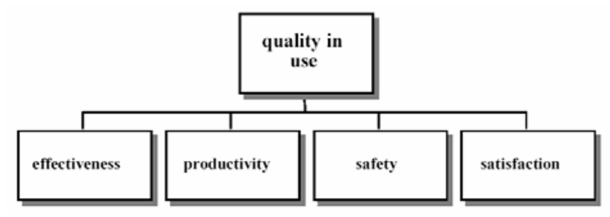


Fig. 3. Relationships between quality in use and internal and external qualities. Extracted from [18].

The properties are selected during the specification phase among the set of characteristics and sub-characteristics elicited by ISO (Fig. 4). Thus, plasticity is not an absolute property: it is specified and evaluated against a set of relevant properties (e.g., the latency and stability of the interactive system with regard to the "efficiency" characteristic, "time behavior" sub-characteristic).



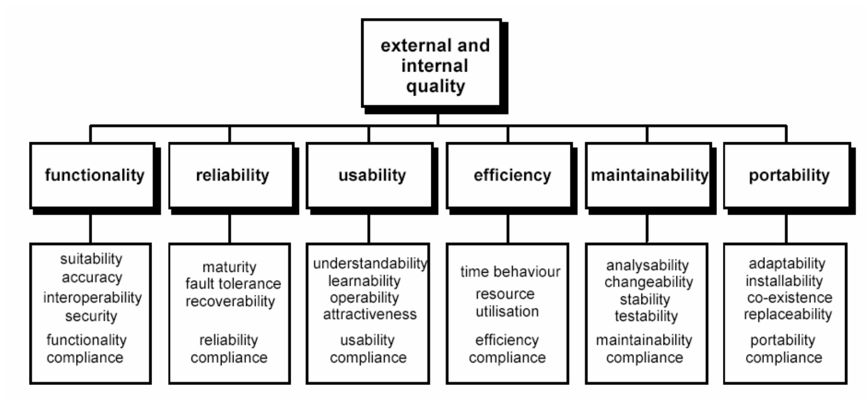


Fig. 4. Quality models for quality in use and internal and external qualities. These ISO models provide a sound basis for specifying and evaluating the extent to which an interactive system is supposed to be plastic. Extracted from [18].

The next section presents how to *plastify* an interactive system from a user centered perspective.

2.2 Plasticity from a user centered perspective

Whilst plasticity has always been addressed from a centralized perspective [5] (the UI was locally tuned as in FlexClock [15]), it is now obvious that ubiquitous computing favors the distribution of the interactive system among a set of platforms. As a result, two means are now available for adapting:

- Recasting the interactive system: this consists in reshuffling the UI, the functional core or the connector between both of these parts locally without modifying its distribution across the different platforms. Figure 1 provides an example of recasting;
- Redistributing the interactive system: it consists in migrating all (total migration) or part of (partial migration) the interactive system across the different platforms. Partial migration has been introduced by Rekimoto's painter metaphor [27] [4] and is now a major issue in HCI.

In ubiquitous computing, the notion of platform is no longer limited to an *elementary platform*, i.e., a set of physical and software resources that function together to form a working computational unit [7]. The notion of platform must definitely be seen as a *cluster*, i.e., a composition of elementary platforms that appear and disappear dynamically. For example, when Alice arrives in Bob's vicinity, her laptop extends the existing cluster composed of Bob's laptop, the PDA and the mobile phone. Bob's current interactive system can partially or fully migrate to Alice's laptop. Typically, to obtain a larger screen, it could be a good option to "bump" [16] the two laptops and split the interactive system between both of them (partial migration) (the *bumping* is illustrated in Figure 5 with two desktops). But when Bob's laptop battery is getting low, a full migration to Alice's laptop seems to be the best option as the screens of the PDA and mobile phone are too small to support a comfortable interaction.

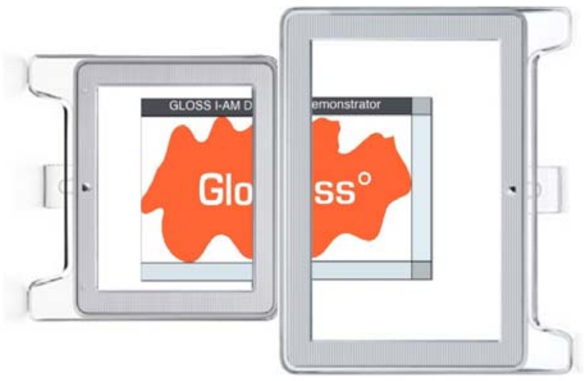


Fig. 5. A partial migration enabled by a top-to-top composition of the screens. Extracted from [9].

The granularity for distribution may vary from the application level to the pixel level [7]:

– At the *application level*, the user interface is fully replicated on the platforms of the target cluster. If the cluster is heterogeneous (e.g., is comprised of a mixture of PC's and PDA's), then each platform runs a specific targeted user interface. All of these user interfaces, however, simultaneously share the same functional core;
– At the *workspace level*, the user interface components that can migrate between platforms are workspaces. A workspace is an interaction space. It groups together a collection of interactors that support the execution of a set of logically connected tasks. In graphical user interfaces, a workspace is mapped onto the notions of windows and panels. The painter metaphor presented in Rekimoto's pick and drop [27] [4] is an example of a distribution at the workspace level: the palettes of tools are presented on a PDA whereas the drawing area is mapped onto an electronic white board. Going one-step further, the tools palette (possibly the drawing area) can migrate at run time between the PDA and the electronic board;
– At the *domain concept level*, the user interface components that can be distributed between platforms are physical interactors. Here, physical interactors allow users to manipulate domain concepts. In Rekimoto's augmented surfaces, domain

concepts, such as tables and chairs, can be distributed between laptops and horizontal and vertical surfaces. As for Built-IT [26], the topology of the rendering surfaces matters: objects are represented as 3D graphic interactors on laptops, whereas 2D rendering is used for objects placed on a horizontal surface;
– At the *pixel level*, any user interface component can be partitioned across multiple platforms. For example, in I-LAND [30], a window may simultaneously lie over two contiguous white boards (it is the same case in Figure 5 with two desktops). When the cluster is heterogeneous, designers need to consider multiple sources of disruption. For example, how to represent a window whose content lies across a white board and a PDA? From a user's perspective, is this desirable?

Migration may happen on the fly at run time or between sessions:

– *On the fly migration* requires that the state of the functional core is saved as well as that of the user interface. The state of the user interface may be saved at multiple levels of granularity: with regard to the functional decomposition promoted by Arch [3], when saved at the Dialogue Component level, the user can pursue the job from the beginning of the current task; when saved at the Logical Presentation or at the Physical Presentation levels, the user is able to carry on the current task at the physical action level, that is, at the exact point within the current task. There is no discontinuity;
– *Migration between sessions* implies that the user has to quit, then restart the application from the saved state of the functional core. In this case, the interaction process is heavily interrupted.

Recasting and redistribution are two means for adaptation. They may be processed in a complementary way. A full migration between heterogeneous platforms will typically require a recasting for fitting to a smaller screen. Conversely, when the user enlarges a window, a partial migration may be a good option to get a larger interaction surface by using a nearby platform. The next section addresses plasticity from a system's perspective.

2.3 Plasticity from a system centered perspective

The CAMELEON reference framework for plasticity [7] provides a general tool for reasoning about adaptation. It covers both recasting and redistribution. It is intended to serve as a reference instrument to help designers and developers to structure the development process of plastic interactive systems covering both the design time and run time.

The design phase follows a model-based approach [25] (Fig. 6). A UI is produced for a set of *initial models* according to a *reification process*:
– The initial models are specified manually by the developer. They set the applicative domain of the interactive system (concepts, tasks), the predicted contexts of use (user, platform, environment), the expected quality of service (a set of requirements related to quality in use and external/internal qualities) and the adaptation to be applied within as well as outside the current context of use (evolution, transition). The domain models are taken from the literature. Emerging

works initiated by [12] [28] deal with the definition and modeling of context of use. The *Quality Models* can be expressed with regard to the ISO models presented in section 2.1. The *Evolution Model* specifies the reaction to be performed when the context of use changes. The *Transition Model* denotes the particular *Transition User Interface* to be used during the adaptation process. A transition UI allows the user to evaluate the evolution of the adaptation process. In Pick and Drop [27], the virtual yellow lines projected on the tables are examples of transition UIs. All of these initial models may be referenced along the development process from the domain specification level to the running interactive system;

– The design process is a three-step process that successively reifies the initial models into the final running UI. It starts at the concepts and tasks level to produce the *Abstract User Interface* (Abstract UI). An abstract UI is a collection of related workspaces called *interaction spaces*. The relations between the interaction spaces are inferred from the task relations expressed in the task model. Similarly, connectedness between concepts and tasks is inferred from the concepts and tasks model. An abstract UI is reified into a *Concrete User Interface* (Concrete UI). A concrete UI turns an abstract UI into an interactor-dependent expression. Although a concrete UI makes explicit the final look and feel of the *Final User Interface* (Final UI), it is still a mockup that runs only within the development environment. The Final UI generated from a concrete UI is expressed in source code, such as Java and HTML. It can then be interpreted or compiled as a pre-computed user interface and plugged into a run-time infrastructure that supports dynamic adaptation to multiple targets.

At any level of reification:

– References can be made to the context of use. We identify four degrees of dependencies: whether a model makes hypothesis about the context of use; a modality; the availability of interactors; or the renderer used for the final UI. From a software engineering perspective, delaying the dependencies until the later stages of the reification process, results in a wider domain for multi-targeting. Ideally, dependencies to the context of use, to modalities and to interactors are associated with the concrete UI level (Fig. 7 a). In practice, the task model is very often context of use and modality dependent (Fig. 7b). As figure 7 shows, a set of four sliders (or stickers) can be used to locate the dependencies in the reification process. The movement of the stickers is limited by the closeness of their neighbour (e.g., in Figure 7b, the interactor sticker has a wide scope for movement between the concepts and tasks level and the final UI level, respectively corresponding to the position of the modality and renderer stickers);
– References can be made to the quality properties that have guided the design of the UI at this level of reification (cf. arrows denoted as “reference” in Figure 6);
– A series of abstractions and/or reifications can be performed to target another level of reification;
– A series of translations can be performed to target another context of use.

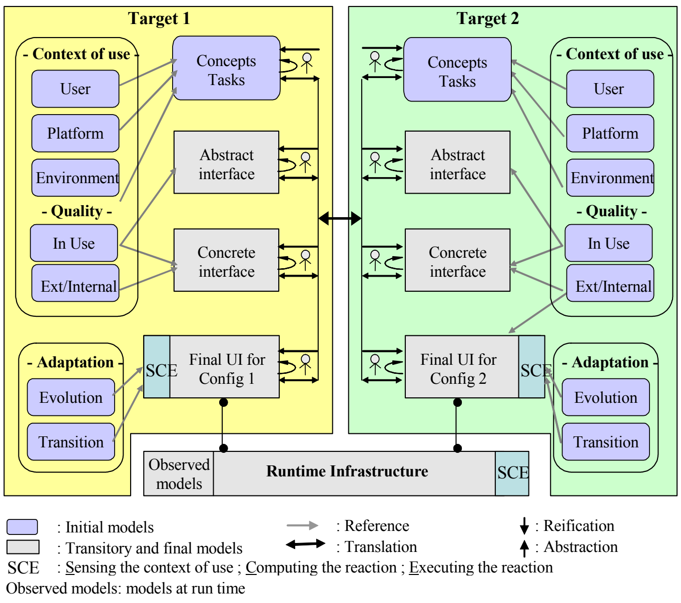


Fig. 6. The Reference Framework for supporting plastic user interfaces. The picture shows the process when applied to two distinct targets. This version is adapted from [7] where the quality models defined in 2.1 are now made explicit. Whilst reifications abstractions and translations are exhaustively made explicit, only examples of references are provided. In the example, the reference to the evolution and transition models is made at the latest stage (the final UIs).

Reifications and translations may be performed automatically from specifications, or manually by human experts. Because the automatic generation of user interfaces has not found wide acceptance in the past [23], the reference framework makes possible manual reifications, abstractions and translations (Fig. 6).

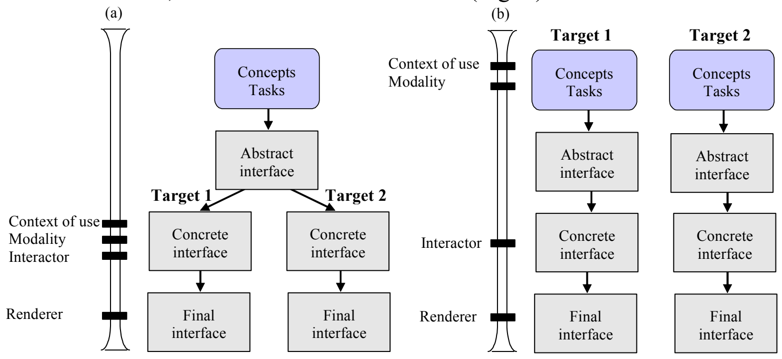


Fig. 7. Two instanciations of the design reference framework. The dependencies to the context of use, modalities, interactors and renderer are localized through stickers that constraint each other in their movement.

As for any evolutive phenomenon, the adaptation at run time is structured as a three-step process: sensing the context of use (S), computing a reaction (C), and executing the reaction (E) [6]. Any of these steps may be undertaken by the final UIs and/or an underlying run time infrastructure (Fig. 6). In the case of distributed UIs, communication between components may be embedded in the components themselves and/or supplied by the runtime infrastructure. As discussed in [24], when the system includes all of the mechanisms and data to perform adaptation on its own (sensing the context of use, computing and executing the reaction), it is said to be *close-adaptive*, i.e., self-contained (autonomous). FlexClock is an example of close-adaptive UI. *Open-adaptiveness* implies that adaptation is performed by mechanisms and data that are totally or partially external to the system. FlexClock would have been open-adaptive if the mechanisms for sensing the context of use, computing the reaction or executing the reaction had been gathered in an external component providing general adaptation services not devoted to FlexClock.

Whether it is close-adaptive or open-adaptive, dynamic reconfiguration is best supported by a component-connector approach [24] [11] [14]. Components that are capable of reflection (i.e., components that can analyze their own behavior and adapt) support close-adaptiveness [21]. Components that are capable of introspection (i.e., components that can describe their behavior to other components) support open-adaptiveness.

The next section applies these advances to the design and run time of comets.

3 The notion of comet

This section relies on the hypothesis that adaptation makes sense at the granularity of a widget. The validity of this hypothesis has not been proven yet, but is grounded in practice: refining an abstract UI into a concrete UI is an experimental composition of widgets with regard to their implicit functional (versus non functional) equivalence or complementarity. Basically, no toolkit makes explicit the functional equivalence of widgets (e.g., the fact that the three versions of Figure 2 are functionally but not non functionally equivalent: they support the same task of selecting one option among a set of options, but differ in many ways, in particular, in their pixels cost). Based on these statement and hypothesis, this paper introduces the notion of comet. It is first defined then examined from both a design and run time perspective. It is finally compared to the state of the art.

3.1 Definition

A comet is an introspective interactor that publishes the quality in use it guarantees for a set of contexts of use. It is able to either self-adapt to the current context of use, or be adapted by a tier-component. It can be dynamically discarded (versus recruited) when it is unable (versus able) to cover the current context of use.

The next section presents a taxonomy and a model of comets from a design perspective.

3.2 The comet from the design perspective

Based on the definition of comets and the advances in plasticity (section 2.3), we identify three types of comets (Fig. 8):

– *Introspective comets* refer to the most basic kind of comets, i.e. interactors that publish their functional and non functional properties (Fig. 9). The functional properties can include adaptation abilities (e.g., sensing the context of use, computing and/or executing the reaction), or be limited to the applicative domain (e.g., selecting one option among a set of options). For instance, the "combo box" comet (Figure 2) does not have to include the adaptation mechanisms for switching from one *form* to another one. It just has to export what it is able to do (i.e., single selection, the task it supports) and at which cost (e.g., footprint, interaction trajectory) to be called a comet;
– *Polymorphic comets* are introspective comets that embed (and publish because of their introspection) multiple versions of at least one of their components. The polymorphism may rise at the functional core level (i.e., the comet embeds a set of algorithms for performing the user task; the algorithms may vary in terms of precision, CPU cost, etc.), at the connector level between the functional core and the UI components (e.g., file sharing versus sockets), or at the UI level (e.g., functional core adaptor, dialog controller, logical or physical presentations with regard to Arch [3]). A comet incorporating the three versions of Figure 2 for selecting one option among a set of options would illustrate the polymorphism at the physical level. Polymorphism provides potential alternatives in case of a change in the context of use. For instance, Figure 2c is more appropriate than Figure 2a for small windows. The mechanism for switching from one *form* to another one may be embedded in the comet itself and/or supplied by a tier-component (e.g. the runtime infrastructure – see section 2.3);
– *Self-adaptive (*or *close-adaptive) comets* are comets that are able to self-adapt to the context of use in a full autonomous way. They embed mechanisms for sensing the context of use, computing and executing the reaction. The reaction may be based on polymorphism in case of polymorphic comets.

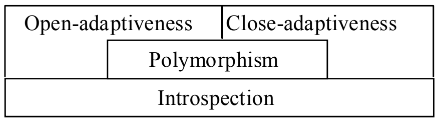


Fig. 8. A taxonomy of comets.

Introspection is the keystone capability of the comet. The properties that are published can be ranked against two criteria (Fig. 9): the type of the property (functional versus non functional) and the type of the service (domain versus adaptation). Examples of properties are provided in Figure 9. Recent research focuses on the notion of *continuity of interaction* [13]. The granularity of distribution and state recovery presented in section 2.2 belong to this area.

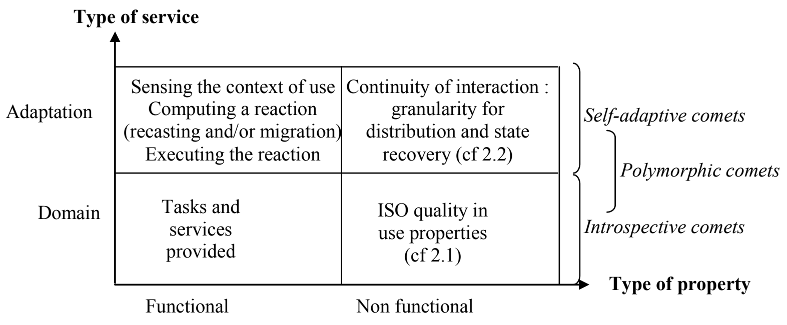


Fig. 9. A taxonomy of properties for structuring introspection.

Based on the nature of the domain task, a difference can be made between general comets that support basic tasks (i.e., those that are supported by classical widgets such as radio buttons, labels, input fields or sliders) and specific comets that support specific tasks. For instance, *PlasticClock* may be seen as a specific comet that simultaneously makes observable the time at two locations, Paris and New York (Figure 10). PlasticClock is polymorphic and self-adaptive. Its adaptation relies on two kinds of polymorphism, thus extending FlexClock:

– Polymorphism of abstraction: PlasticClock is able to compute the times in both an absolute and a relative way. The absolute version consists in getting the two times on web sites. Conversely, the relative way requests one time only and computes the second one according to the delay;
– Polymorphism of presentation: as shown in Figure 10, PlasticClock is able to switch from a large presentation format putting the two times side by side, to a more compact one gathering the two times on a same clock. Two hands (hours and minutes) are devoted to Paris. The third one points out the hours in New York (the minutes are the same). Allen's relations [2] provide an interesting framework for comparing these two presentations from a non functional perspective.

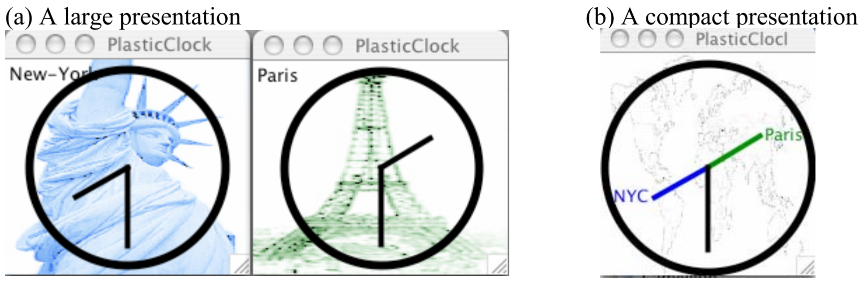


Fig. 10. PlasticClock.

The specific comets raise the question of the threshold between a comet and an interactive system. Should PlasticClock be considered as a comet or an interactive system? To our understanding, the response is grounded in software engineering: it depends on the expected level of reusability. As a result, comets can be designed as interactive systems. Figure 11 provides an UML class diagram obtained by applying both the reference framework and the taxonomy of comets for modeling a comet:

– A comet may be defined at four levels of abstraction. The most abstract one, called abstraction, is mandatory. This level may serve as starting point for producing abstract, concrete and final interaction objects (AIO, CIO, FIO) through a series of reifications and/or abstractions;

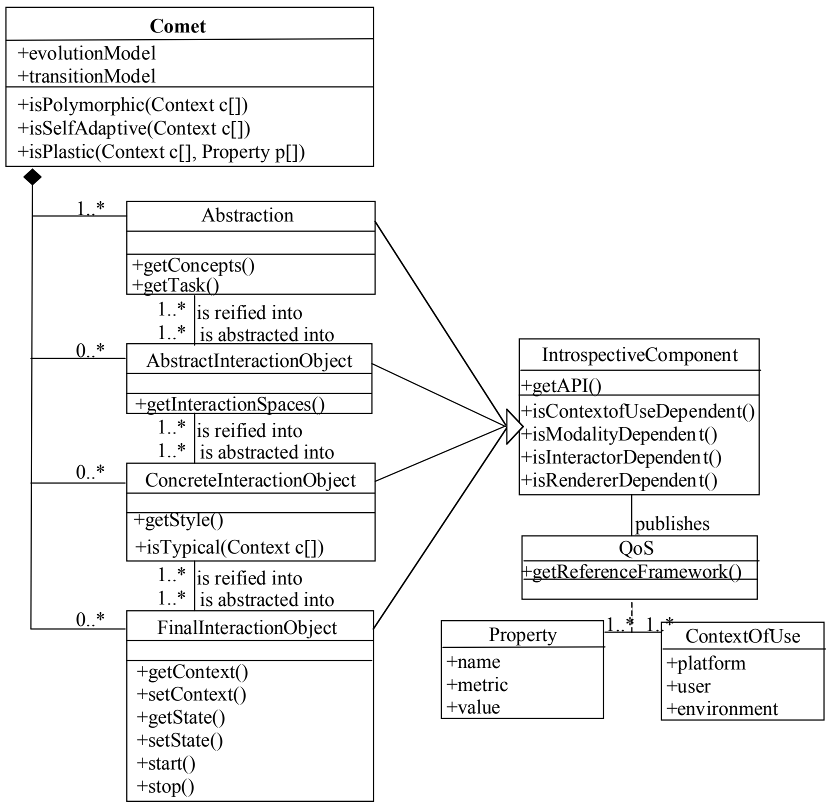


Fig. 11. A comet modeling taking benefit from both the reference framework and the taxonomy of comets.

The next section deals with the comets at run time.

– At any level of reification, comets are introspective, i.e., aware of and capable of publishing their dependencies and quality of service (QoS). The dependencies are expressed in terms of context of use, modality, interactor and renderer. The quality of service denotes the quality in use the comet guarantees on a set of contexts of use. It is expressed according to a reference framework (e.g. ISO) by a set of properties. In a more general way, introspective components publish their API;

- Specific information and/or services are provided at each level of reification. At the abstraction level, they are related to the concepts and task the comet supports; at the AIO level, the structure of the comet in terms of interaction spaces; at the CIO level, the style of the comet (e.g., the style "button") and its typicality for the given purpose (e.g., whether it is or not typical to use radio buttons for specifying the platform – Figure 2a); at the final level, the effective context of use and the interaction state of the comet. Managing the interaction state (i.e., maintaining, saving and restoring the state of the comet) is necessary for performing adaptation in a continuous way;
- The comets may embed an evolution and a transition model for driving adaptation. The comet publishes its polymorphism and self-adaptiveness capabilities for a set of contexts of use. Going one step further, it directly publishes its plasticity property for a set of properties P and a set of contexts of use C. It is plastic if any property of P is preserved for any context of C.

3.3 The comet from the run time perspective

This section addresses the execution of comets. It elicits a set of strategies and policies for deploying plasticity. It proposes a software architecture model for supporting adaptation.

We identify four classes of strategies:

- Adaptation by *polymorphism*. This strategy preserves the comet but changes its *form*. The change may be performed at any level of reification according to the three following cardinalities, 1-1, 1-N, N-1 depending on the fact that the original form is replaced by another one (cardinality 1-1), by N forms (cardinality 1-N) or that N forms, including the original form, are aggregated into an unique one (cardinality N-1). For instance, in Figure 2, when the comet switches from a to b, it performs a 1-1 polymorphism: the radio buttons are replaced with a combo box. When it switches from b to c, it performs a 2-1 polymorphism (respectively switching from c to b is a 1-2 polymorphism);
- Adaptation by *substitution*. Conversely to the adaptation by polymorphism, this strategy does not preserve the comet. Rather, it is replaced by another one (cardinality 1-1) or N comets (cardinality 1-N) or is aggregated with neighbor comets (cardinality N-1);
- Adaptation by *recruiting* consists in adding comets to the interactive system. This strategy supports, for instance, a temporary need for redundancy [1];
- Adaptation by *discarding* is the opposite strategy to the recruiting strategy. Comets may be suppressed because the tasks they support no longer make sense.

At run time, the strategies may be chosen according to the evolution model of the comet. The selected strategy is performed according to a policy. The policies depend on the autonomy of the comets for processing adaptation. We identify three types of policies:

- An *external non-concerted policy* consists in fully subcontracting the adaptation. Everything is performed externally by a tier-component (e.g. another comet or the runtime infrastructure) without any contribution of the comet. This policy is

suitable for comets which are unable to deal with adaptation. In practice, this is an easy way for guarantying the global ergonomic consistency of the interactive system. In this case, adaptation may be centralized in a dedicated agent (the tier-component);
- Conversely, the *internal non-concerted policy* consists in achieving adaptation in a fully autonomous way. Everything is performed inside the comet, without cooperating with the rest of the interactive system. The open issue is how to maintain the global ergonomic consistency of the interactive system;
- Intermediary policies, said *concerted policies*, depend on an agreement between the comet and tier-components. An *optimistic* version consists in applying the decision before it is validated by peers, whilst in a *pessimistic* version the comet waits for an authorization before applying its decision. The optimistic version is less time consuming but requires an undo procedure to cancel a finally rejected decision.

In practice, the policy decision will be chosen against criteria such as performance (c.f. the efficiency characteristic, time behavior sub-characteristic in section 2.1). The software architecture model Compact (COntext of use Mouldable PAC for plasticity) has been designed to take into account such an issue.

Compact is a specialization of the PAC (Presentation Abstraction Control) [8] model for plasticity. PAC is an agent-based software architecture model that identifies three recurrent facets in any component of an interactive system: an abstraction, a presentation and a control that assures the coherence and communication between the abstraction and the presentation facets. According to the “separation of concerns” principle promoted by software engineering, Compact splits up each facet of the PAC model in two slices, thus isolating a logical part from physical implementations in each facet (Fig. 12):
- Abstraction: as with the functional core adaptor in Arch, the logical abstraction acts as an API for the physical abstraction. It provides a framework for implementing the mechanisms to switch between physical abstractions (i.e., the functional core(s) of the comet; they may be multiple in case of polymorphism at this level). It is in charge of maintaining the current state of the comet;
- Presentation: in a symmetric way, as with the presentation component in Arch, the logical presentation acts as an API for the physical presentation part. It provides a framework for implementing the mechanisms to switch between presentations (they are multiple in case of polymorphism at this level);
- Control: the logical part of the control assumes its typical role of coherence and communication between the logical abstraction and the logical presentation. The physical part, called “Plastic” (Fig. 12), is responsible for (a) receiving and/or sensing and/or transmitting the context of use whether the comet embeds or not any sensors (i.e., the Sensing step of the Reference Framework), (b) receiving and/or computing and/or transmitting the reaction to apply in case of changes of context of use (i.e., the Computation step of the Reference Framework), and (c) eventually performing the reaction (i.e., the Execution step of the Reference Framework). The reaction may consist of switching between physical abstractions and/or presentations. The computation is based on a set of pairs composed of compatible physical abstractions and presentations. At any point in time, one or

many physical abstractions and/or presentations may be executed. Conversely, logical parts are only instanciated once per comet.

As in PAC, an interactive system is a collection of Compact agents. Specific canals of communication can be established between the plastic parts of the controls to propagate information in a more efficient way and/or to control ergonomic consistency in a more centralized way. Compact is currently under implementation as discussed in the conclusion. The next section analyses the notion of comet with regard to the state of the art.

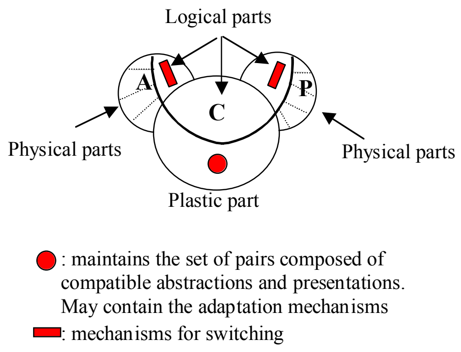

Logical parts
A
C
P
Physical parts
Physical parts
Plastic part
● : maintains the set of pairs composed of compatible abstractions and presentations. May contain the adaptation mechanisms
▬: mechanisms for switching

Fig. 12. The Compact software architecture model, a version of the PAC model (Presentation, Abstraction, Control) specifically mold for plasticity.

3.4 Comets and the state of the art

Plasticity is a recent property that has mostly been addressed at the granularity of interactive systems. The widget level has rarely been considered. We note that most of these works focus on the software architecture modeling. Based on the identification of two levels of abstraction (AIOs and CIOs) [33], they propose conceptual and implementational frameworks for supporting adaptation [22] [20] [10]. But adaptation is limited to the presentation level [20] [10]. They do not cover adaptations ranging from the dialog controller to the functional core.

We now have to go further in the implementation. We keep in mind the issue of legacy systems [20] and the need for integrating multimodality as a means for adaptation [10].

4 Conclusion and perspectives

Based on a set of recent advances in plasticity, this paper introduces a new generation of widgets: the notion of comets. A comet is an interactor mold for adaptation: it can self-adapt to some context of use, or be adapted by a tier-component, or be dynamically discarded (versus recruited) when it is unable (versus able) to cover the current context of use. To do so, a comet publishes the quality in use it guarantees, the user tasks and domain concepts it is able to support, as well as the extent to which it supports adaptation. The reasoning relies on a scientific hypothesis which is as yet unvalidated: the fact that adaptation makes sense at the widget level. The idea is to promote task-driven toolkits where widgets that support the same tasks and concepts are aggregated into a unique polymorphic comet. Such a toolkit, called "Plasturgy studio" is currently under implementation. For the moment, it focuses on the basic graphical tasks: specification (free specification through text fields, specification by selection of one or many elements such as radio buttons, lists, spinners, sliders, check boxes, menus, combo boxes), activation (button, menu, list) and navigation (button, link, scroll). This first toolkit will provide feedback about both the hypothesis and the appropriate granularity for widgets. If successful, the toolkit will be extended to take into account multimodality as a means for adaptation.

Acknowledment

This work is being supported by the European commission funded CAMELEON R&D project IST-2000-30104. The authors would like particularly to thank Jean Vanderdonckt and Quentin Limbourg, members of the project. Many thanks to Julie Dugdale for checking the paper.

References

1. Abowd, G.D., Coutaz, J., Nigay, L.: Structuring the Space of Interactive System Properties, Engineering for Human-Computer Interaction, Larson J. & Unger C. (eds), Elsevier Science Publishers B.V. (North-Holland), IFIP (1992) 113-126
2. Allen, J.: Maintaining Knowledge about Temporal Intervals, Journal Communication of the ACM 26(11), November (1983). 832-843
3. Arch: "A Metamodel for the Runtime Architecture of An Interactive System", The UIMS Developers Workshop, SIGCHI Bulletin, 24(1), ACM Press (1992)
4. Ayatsuka, Y., Matsushita, N. Rekimoto, J.: Hyperpalette: a hybrid Computing Environment for Small Computing Devices. In: CHI2000 Extended Abstracts, ACM Publ. (2000) 53–53
5. Calvary, G., Coutaz, J., Thevenin, D.: A Unifying Reference Framework for the Development of Plastic User Interfaces, Proceedings of 8th IFIP International Conference on Engineering for Human-Computer Interaction EHCI'2001 (Toronto, 11-13 May 2001), R. Little and L. Nigay (eds.), Lecture Notes in Computer Science, Vol. 2254, Springer-Verlag, Berlin (2001) 173-192
6. Calvary, G., Coutaz, J., Thevenin, D.: Supporting Context Changes for Plastic User Interfaces : a Process and a Mechanism, in "People and Computers XV – Interaction without Frontiers", Joint Proceedings of AFIHM-BCS Conference on Human-Computer Interaction IHM-HCI'2001 (Lille, 10-14 September 2001), A. Blandford, J. Vanderdonckt, and Ph. Gray (eds.), Vol. I, Springer-Verlag, London (2001) 349-363

7. Calvary, G., Coutaz, J., Thevenin, D., Bouillon, L., Florins, M., Limbourg, Q., Souchon, N., Vanderdonckt, J., Marucci, L., Paternò, F., Santoro, C.: The CAMELEON Reference Framework, Deliverable D1.1, September 3th (2002)
8. Coutaz, J.: PAC, an Object Oriented Model for Dialog Design, In Interact'87, (1987) 431-436
9. Coutaz, J. Lachenal, C., Barralon, N., Rey, G.: Initial Design of Interaction Techniques Using Multiple Interaction Surfaces, Deliverable D18 of the European GLOSS (Global Smart Spaces) project, 27/10/2003
10. Crease, M., Gray, P.D. & Brewster, S.A.: A Toolkit of Mechanism and Context Independent Widgets. In procs of the Design, Specification, and Verification of Interactive Systems workshop, DSVIS'00, (2000) 121-133
11. De Palma, N., Bellisard, L., Riveill, M. : Dynamic Reconfiguration of Agent-Based Applications . Third European Research Seminar on Advances in Distributed Systems (ERSADS'99), Madeira Island (Portugal), (1999)
12. Dey, A.K., Abowd, G.D.: Towards a Better Understanding of Context and Context-Awareness, Proceedings of the CHI 2000 Workshop on The What, Who, Where, When, and How of Context-Awareness, The Hague, Netherlands, April 1-6, (2000)
13. Florins, M., Vanderdonckt, J.: Graceful degradation of User Interfaces as a Design Method for Multiplatform Systems, In IUI'94, 2004 International Conference on Intelligent User Interfaces, Funchal, Madeira, Portugal, January 13-16, (2004) 140-147
14. Garlan, D., Schmerl, B., Chang, J.: Using Gauges for Architectural-Based Monitoring and Adaptation. Working Conf. on Complex and Dynamic Systems Architecture, Australia, Dec. (2001)
15. Grolaux, D., Van Roy, P., Vanderdonckt, J.: QTk: An Integrated Model-Based Approach to Designing Executable User Interfaces, in PreProc. of 8th Int. Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'2001 (Glasgow, June 13-15, 2001), Ch. Johnson (ed.), GIST Tech. Report G-2001-1, Dept. of Comp. Sci., Univ. of Glasgow, Scotland, (2001) 77-91. Accessible at http:// www.dcs.gla.ac.uk/~johnson/papers/dsvis_2001/grolaux
16. Hinckleyss, K.: Distributed and Local Sensing Techniques for Face-to-Face Collaboration, In ICMI'03, Fifth International Conference on Multimodal Interfaces, Vancouver, British Columbia, Canada, November 5-7, (2003) 81-84
17. IFIP BOOK: Design Principles for Interactive Software, Gram C. and Cockton G. (eds), Chapman & Hall, (1996)
18. ISO/IEC CD 25000.2 Software and Systems Engineering – Software product quality requirements and evaluation (SquaRE) – Guide to SquaRE, 2003-01-13 (2003)
19. ISO/IEC 25021 Software and System Engineering – Software Product Quality Requirements and Evaluation (SquaRE) – Measurement, 2003-02-03
20. Jabarin, B., Graham, T.C.N.: Architectures for Widget-Level Plasticity, in Proceedings of DSV-IS (2003) 124-138
21. Marangozova, V., Boyer, F.: Using reflective features to support mobile users. Workshop on Reflection and meta-level architectures, Nice, Juin, (2002)
22. Markopulos, P.: A compositional model for the formal specification of user interface software. Submitted for the degree of Doctor of Philosophy, March (1997)
23. Myers, B., Hudson, S., Pausch, R.: Past, Present, Future of User Interface Tools. Transactions on Computer-Human Interaction, ACM, 7(1), March (2000), 3–28
24. Oreizy, P., Tay lor, R., et al.: An Architecture-Based Approach to Self-Adaptive Software. In IEEE Intelligent Systems, May-June, (1999) 54-62

25. Pinheiro da Silva, P.: User Interface Declarative Models and Development Environments: A Survey, in Proc. of 7th Int. Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'2000 (Limerick, June 5-6, 2000), F. Paternò & Ph. Palanque (éds.), Lecture Notes in Comp. Sci., Vol. 1946, Springer-Verlag, Berlin, (2000) 207-226
26. Rauterberg, M. et al.: BUILT-IT: A Planning Tool for Consruction and Design. In Proc. Of the ACM Conf. In Human Factors in Computing Systems (CHI98) Conference Companion, (1998) 177-178
27. Rekimoto, J.: Pick and Drop: A Direct Manipulation Technique for Multiple Computer Environments. In Proc. of UIST97, ACM Press, (1997) 31-39
28. Salber, D., Abowd, Gregory D.: The Design and Use of a Generic Context Server, In the Proceedings of the Perceptual User Interfaces Workshop (PUI '98), San Francisco, CA, November 5-6, (1998) 63-66
29. Seffah, A., Kececi, N., Donyaee, M.: QUIM: A Framework for Quantifying Usability Metrics in Software Quality Models, APAQS Second Asia-Pacific Conference on Quality Software, December, Hong-Kong (2001) 10-11
30. Streitz, N. et al.: I-LAND: An interactive landscape for creativity and innovation. In Proc. of the ACM Conf. On Human Factors in Computing Systems (CHI99), Pittsburgh, May 15-20, (1999) 120-127
31. Thevenin, D., Coutaz, J.: Plasticity of User Interfaces: Framework and Research Agenda. In: Proc. Interact99, Edinburgh, A. Sasse & C. Johnson Eds, IFIP IOS Press Publ., (1999) 110–117
32. Van Welie, M., van der Veer, G.C., Eliëns, A.: Usability Properties in Dialog Models: In: 6th International Eurographics Workshop on Design Specification and Verification of Interactive Systems DSV-IS99, Braga, Portugal, 2-4 June (1999) 238-253
33. Vanderdonckt, J., Bodart, F.: Encapsulating knowledge for intelligent automatic interaction objects selection, In Ashlund, S., Mullet, K., Henderson, A., Hollnagel, E., White, T. (Eds), Proceedings of the ACM Conference on Human Factors in Computing Systems InterCHI'93, Amsterdam, ACM Press, New-York, 24-29 April, (1993) 424-429

Discussion

[Tom Ormerod] How much of the value of comet actually comes from the metaphor used at the interface ?

> [Gaëlle Calvary] The notion of comet is primary driven by the user task. In PlasticClock, when the screen size is enlarged, the date becomes observable because this task has been recognized as relevant for the user. It has been modeled in the task model. Conversely, if space is tight, then interaction is strictly reduced to the main tasks. So, the notion of comet is primary driven by functional aspects. Non functional properties are considered for selecting the most appropriate form. We will, for example, favor such or such metaphor. The problem is when no solution fits both functional and non functional requirements. Trade-offs are unavoidable. They are driven by strategies. This balance between functional and non functional properties is an interesting issue.

[Tom Ormerod] So, metaphor does not drive the design of the comet - the specification of tasks determines the appropriate metaphor.

> [Gaëlle Calvary] Yes. Of course, if the metaphor conveys an implicit task, then the task can be made explicit in a dedicated comet and the metaphor registered as possible presentation.

[Philippe Palanque] In the example of the plastic clock some tasks are not available anymore in the bigger clock such as provide the user with the precise time in Paris including minutes and seconds. Does Comet provide some help for checking such constraints ?

> [Gaëlle Calvary] First point, PlasticClock is just a demonstrator of plasticity. It has not been implemented as a collection of comets. Then, in practice, a comet is created if it is promising in terms of reusability. So, it is finely analyzed from a user-centered perspective in terms of accuracy, etc. Its adaptation rules are discussed with final users. Then, at run time, tradeoffs are performed to achieve an optimum. It can be global to the interactive system, or local to a comet. As a result, mismatches may appear between local and global interests. Strategies have to deal with such issues. So, in summary, a comet is designed in a local consistent way. But, when involved in an interactive system, adaptation must be solved in a global way.

[Jurjen Ziegler] Did you address some high-level adaptation strategies such as substituting agents by others in the run-time architecture ?

> [Gaëlle Calvary] Yes. We have elicited a functional decomposition of the runtime infrastructure that includes a component retriever and a configurator. The retriever is in charge of finding a component (or agent) in a repository that is then deployed by the configurator. Adaptation may be done at several levels of abstraction. Components may be retrieved at different levels of abstraction. Producing tools may be required to reify components that are not executable. Yet, adaptation is specified by rules. We are studying the appropriateness of Bayesian networks.

[Bonnie John] (to both Gaëlle and Simon Dobson) You are both offering different ways to think about the problem of contextual-aware systems. How do you evaluate whether your approach is a promising way to go forward ?

> [Gaëlle Calvary] Our approach is strongly coupled with software engineering. The validation lies in the cost/benefit ratio. Does a library of comets improve the productivity of engineers and/or the quality of service of the interactive system? We have to go further in the implementation to answer the question.
>
> [Simon Dobson] We have nothing to say about what adaptations are made. What we deal with are the situations in which adaptations should occur, and we can inform whatever mechanism is used to actually perform the adaptation. In terms of evaluation, our work should be evaluated as an aid to expression for designers and programmers: does it simplify the way in which adaptation occurs, does it improve analysis and the ability to develop correct

systems. "Correct" remains an external notion depending on the application being considered.

[Grigori Evreinov] For efficient adaptation and visualization of spatial events and/or widgets the right metaphor is very important. To validate the metaphor itself it could be interesting to apply the proposed approach for adapting temporal events, objects and widgets, that is, under time-pressure condition a spatial arrangement could be present more effectively.

[Gaëlle Calvary] Yes, we have to investigate time. Bayesian networks could be an option.

Using Interaction Style to Match the Ubiquitous User Interface to the Device-to-Hand

Stephen W. Gilroy and Michael D. Harrison[13]

Dependability Interdisciplinary Research Collaboration,
Department of Computer Science, University of York, York YO10 5DD, UK.
steveg@cs.york.ac.uk

Abstract. Ubiquitous computing requires a multitude of devices to have access to the same services. Abstract specifications of user interfaces are designed to separate the definition of a user interface from that of the underlying service. This paper proposes the incorporation of interaction style into this type of specification. By selecting an appropriate interaction style, an interface can be better matched to the device being used. Specifications that are based upon three different styles have been developed, together with a prototype Style-Based Interaction System (SIS) that utilises these specifications to provide concrete user interfaces for a device. An example weather query service is described, including specifications of user interfaces for this service that use the three different styles as well as example concrete user interfaces that SIS can produce.

1 Introduction

The increasing availability of personalized and ubiquitous technologies leads to the possibility that whatever the device-to-hand is, it becomes the way to access services and systems. Therefore, interfaces to services must be designed for a variety of different types of device from desktop systems to handheld or otherwise portable devices. Different styles of interaction often suit different devices most effectively. While the appearance of ubiquitous devices has brought forth a proliferation of innovative interactive techniques, the broad categories and aspects of style as, for example, identified by Newman and Lamming [1] can still be applied. While a *key-modal* interface may be appropriate for a mobile telephone, with its limited screen and restricted keypad, a *direct manipulation* (DM) interface may be appropriate for a device based around touch / pen interactive techniques, such as current models of palmtop or tablet PCs. Typically in such situations a different low-level interface will have to be designed separately for each device. It is possible that several interaction styles may have to be supported for different users or parts of the system on the same device. As new technologies evolve to meet the demands of ubiquitous computing additional styles will emerge.

[13]Mailing address: Informatics Research Institute, University of Newcastle upon Tyne, NE1 7RU, UK. michael.harrison@ncl.ac.uk

Style-specific design considerations normally take the form of guidelines, heuristics or ad-hoc rationalizations by designers [2]. Designs to support many devices may be facilitated by incorporating interaction style explicitly into an implementation. In this paper we demonstrate that incorporating style-level descriptions into a model of a user interface can give more flexibility than forcing a single user interface model on a heterogeneous selection of devices. This paper is concerned with an approach in which interaction with a service is bound to the features of the platform through a mediating style description. The aim is to support an interface that is appropriate given the technological constraints or opportunities afforded by the platform. In section 2 the approach to the style-based interaction system is contrasted with other approaches to platform independent service provision. In section 3 the interaction style approach is described in more detail. In section 4 an implementation of a style-based system and the specifications that drive it are described. In section 5 an example of a weather system is used to illustrate the idea. In section 6 the approach is discussed again in relation to other similar approaches and in section 7 the paper draws conclusions.

2. Modelling the Ubiquitous User Interface

Separating the user interface from application functionality [3] is a key theme in the delivery of interactive applications to multiple platforms. This is achieved by abstracting the interaction with a user interface from its presentation on a specific device. Model-based user interface development [4] provides useful tools to cleanly separate the parts of an application. However, its potential for easing cross-platform user interface development is less apparent when platforms differ in their support for styles.

The rise of ubiquitous computing and the proliferation of user appliances of widely differing capabilities and limitations have given new impetus to the need for cross-platform interface design. A provider of ubiquitous services typically wishes to target different users who may use devices of different capabilities, or a user or set of users who wish to migrate their use of services across several different devices.

Separation of application functionality and delivery via abstractly defined interfaces can be addressed in this broader context by the use of *service frameworks* [5] that organize and aggregate software functionality and data, and facilitate universal access to it. *Universal user interfaces* will provide interaction with services on a variety of devices, tailoring the interface to suit the device.

2.1 Service Frameworks

A service framework enables application functions to be delivered to devices whatever and wherever the devices are. The Web is an example of a framework for the delivery of many similar services through Hyper-Text Markup Language (HTML) files provided by web servers. Web services are delivered via Universal Resource Locators (URLs) that identify a particular service (usually requesting a single page of information). A user therefore makes the required service explicit by entering a URL into the browser manually, through a bookmark, or via a hyperlink. Other frameworks, e.g., XWeb [6], use similar approaches to existing web services and provide better support for diverse interaction.

2.2 Universal Interface Specification

An application's behaviour can be defined independently of platform, through the use of services. However, a mechanism is required to map that behavior to the specific interface components of a device. Model-based approaches map abstractions of interaction objects onto platform-specific implementations. The interactive components of the interface, for example a text box for inputting text or a drop-down list for making a choice, are abstracted and encapsulated in terms of a relatively small set of "interactors" [7]. Other approaches utilize several levels of abstraction that may include low-level "widgets", as well as more abstract components such as "group" or "choice". The sets of widgets available on different platforms may not intersect in terms of detail but as long as the abstraction can be fulfilled by a widget that *is* available on a particular platform then a concrete interface can be rendered.

2.3 Problems of Abstract User Interface Models

Abstract interface models [6, 8-12] are problematic when abstraction is such that there is no convenient implementation of the low-level interaction objects on a particular platform. A model must be defined to either restrict the set of objects to ones that are common across all platforms, or provide a wider set of objects to cover the variation in platform. In the former case, the interface becomes the "lowest common denominator" of all target platform capabilities, and is unsuitable if a new platform has interaction objects that do not exist in the available set. In the latter case, abstract objects are a union of available platforms. This gives rise to the two-fold problem of an ever-expanding library, or "toolkit", of widgets and an overly complicated mapping scheme to select the correct widgets for a platform.

Presenting a user interface for a UIML [11,12] specification on a specific platform involves more than selecting an appropriate widget representation. An interface structure that is defined *canonically* may fit one platform but not another. It is then necessary to have different specifications for cross platform structure variations, or alternatively a generic structure specification, which may be overridden when mapping the parts of the interface to actual platform elements. This defeats some of the point of a single structure definition. UIML also assumes a one-to-one mapping of parts to toolkit implementations. If a part in one interface implementation is needed it is added to the canonical definition of parts, even if it is not mapped to a particular platform.

XWeb [6], on the other hand, provides a higher-level formal specification of semantic interaction than a simple widget mapping. However, it still suffers from the "structure" problems of UIML in that it uses "grouping" interactors that arrange other interactors in a hierarchical structure, incorporating a canonical XView. An XView defines which elements of a data tree are manipulated by each interactor. While

XWeb allows designers to reuse a view specification across clients with no extra effort, designs have to combine the interactors into views that are suitable for all platforms. The designer can therefore either design one set of views that maps to all client devices, or create a different set of views for different client types, losing the advantage of a single specification. Even if this is done, a new client with new interactor implementations might have usability problems with existing views, a problem encountered when speech widgets were implemented in an XWeb client [6].

3. A Model of Interaction Style

A model that incorporates interaction style makes it possible to vary the structure or interface semantics applied across devices. User interface descriptions are defined on a per-style basis and a target device selects the description that best maps onto its capabilities. Hence, if a form-fill interaction style is most appropriate for the device in the context of a particular application then that style is bound to the application and mapped to the interactive components of the device. For another target device a dialogue style might be more appropriate and in this case, the same application software would be bound with this different style.

The number of styles supported in the model should be finite and small, to allow a designer to target the maximum number of devices with the minimum amount of effort. It should also be possible to add a completely new style by creating additional definitions for existing interfaces. Although a designer does not have to support all styles, compatibility will be lost if devices do not support the styles chosen.

Two distinguishing features of a style are the manner in which they guide the user to the desired task or function and how they gather required input from the user. There may be semantic relationships that are shared across styles but which manifest themselves in different ways.

The style-based interaction system described in section 4 incorporates support for three styles: form-fill, dialogue and menu. Although these three are considered “classic” styles that can be applied to desktop systems, they also apply equally to other kinds of device. The services provided may be targeted at both desktop and mobile devices. Form-fill would map onto a web-style interface on desktop type systems, dialogue for voice-based telephone systems and menu for mobile phones or embedded devices.

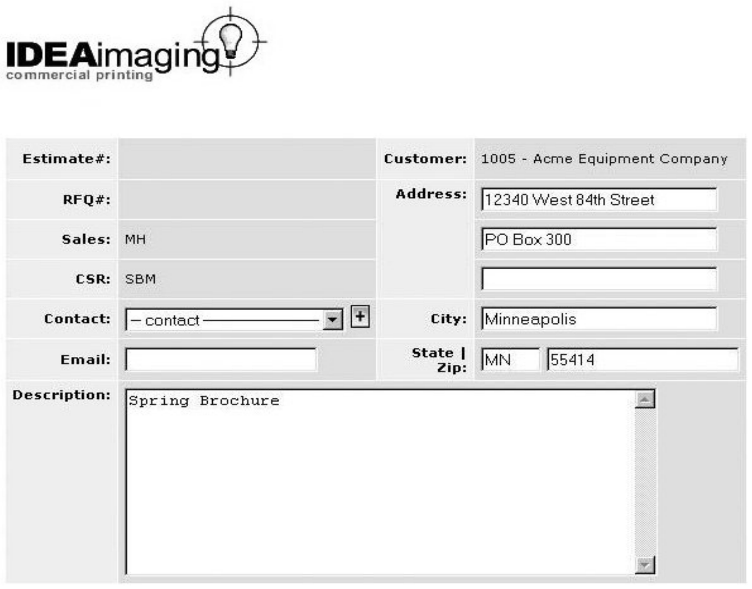
3.1 Form-Fill Style

Forms are two-dimensional rather than one-dimensional, so navigation is important. The organization of a form on the display of the device requires a logical structure so that it can be decomposed to suit different display capabilities [13].

Form elements have different interaction requirements. Simple elements just require text entry while complex elements involve groups of choices or data of a particular format and may be mandatory or optional. The relation between elements might mean that two elements are mutually exclusive, or that filling in an element makes other elements or form sections mandatory. In addition, the elements that are filled in might affect what actions are available with the form data.

When the form is filled in, an action must be chosen to process the information. This is usually done by special commands, or buttons. An action might specify a certain set of form elements from which it processes information or the action invoked by a command might depend on the value of certain form elements. Validation of elements could occur before processing or feedback given if the processing finds invalid information.

A typical example of a form-fill style is the web-based form illustrated in figure 1(a).



Estimate#:		Customer:	1005 - Acme Equipment Company
RFQ#:		Address:	12340 West 84th Street
Sales:	MH		PO Box 300
CSR:	SBM		
Contact:	-- contact --	City:	Minneapolis
Email:		State | Zip:	MN 55414
Description:	Spring Brochure		

Fig. 1(a). A Web-based Form Interface.

3.2 Dialogue Style

The key feature of this style is the structure of the dialogue with the user. As questions are posed, the user's answer determines the next question asked and that answer may be a piece of data that is gathered. A state-chart notation is useful in describing this interface. Each state is a mode of the interface, and the transitions between states are the available choices. On entering a state the appropriate prompt is displayed. Input and output in a question/answer interface is one-dimensional so, while it is limited in terms of interaction, it can be supported by devices without complex graphical capabilities and the conversational nature of interaction facilitates the use of speech. VoiceXML systems (figure 1(b)) are an example of a dialogue style of interface.

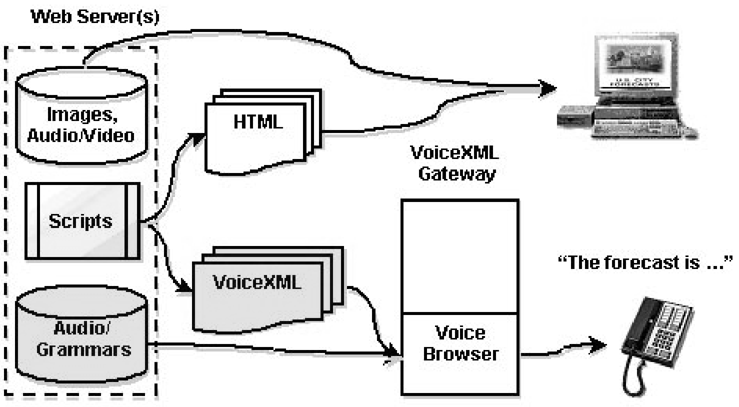


Fig. 1(b) A Voice XML Dialogue Interface.

3.3 Menu Style

The navigational structure of a menu style is governed by how best to partition the menu space to provide meaning to guide the user. Breadth is preferred over depth, as deep menus have the same orientation problems as dialogue structures. Devices that employ menu interfaces have a limited, customised input mechanism based around a small number of specialized buttons or keys. Input and navigation must be designed to facilitate easy mapping from an unknown layout of keys. Current generation mobile phones typically utilize a menu interface as shown in figure 1(c).

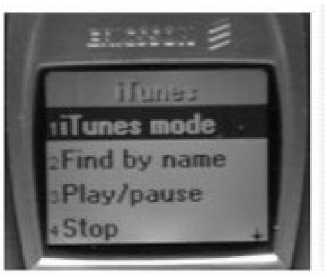


Fig. 1(c) A Mobile Phone Menu Interface.

4 Style-Base Interaction System (SIS) Framework

A prototype application framework supports interfaces using a variety of styles as outlined in section 3. The components of the framework are shown in figure 2. The framework consists of a runtime system that is configured by a set of eXtensible Mark-up Language (XML) specifications describing the service and style-based user interfaces of an application.

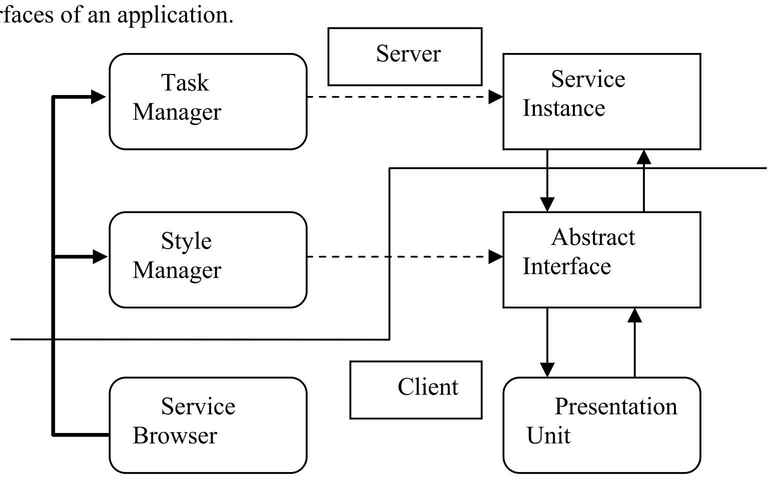


Fig. 2. SIS Framework.

SIS consists of both components that reside on a client appliance and those that can be managed on a remote server. Within a running ubiquitous application, this distinction is transparent. SIS is designed to switch easily between different style instantiations running on a single service instantiation. A user may thus migrate between different appliances without losing saved task-level information. It is feasible to swap a running style between different instances of the same service or two different services that both support the set of tasks required by the style definition.

The three components that deal with the initialization and management of an application are the *Service Browser* on the client, a *Style Manager* to look after styles and a *Task Manager* to look after the tasks required by services. Managers exist as separately running entities, possibly residing on remote servers, with their own resources and are configured using XML specifications of task and style. They use this configuration to generate the run-time components of the interface: *Service Instances* and an *Abstract Interface* for each style. Device specific *Presentation Units* provide concrete interface instantiations on each client. A weather service application is used to illustrate the approach.

4.1 Task Definition using Service Specifications

The XML specification of a service defines its tasks, required function and data storage. A task manager generates run-time instantiations of services called service instances from these specifications. A service instance provides the data storage for its component tasks and a list of all the tasks in the service. Task instantiations are shared between services that use them, and are maintained by the task manager. When a service instance needs a task, it calls the task using the manager that created it. Tasks are identified by a namespace scheme[14] to avoid clashes between tasks of the same name utilized by different services.

Functions. Service functions implement the tasks that are part of a service and "wrap" the logic implementation so that there is a consistent interface for use in SIS. SIS also allows external functions (utility functions) to manipulate data before it is used in a function call. An example service and utility function specification are shown in figure 3. The `class` and `method` attributes identify a function's Java implementation. The `<return>` and `<parameter>` elements identify the function's return type and required parameters respectively. Utility functions do not affect the state of the underlying application logic, but are assumed to perform some repeatable translation upon data. SIS therefore does not need to know the implementation of data types to be able to manipulate them.

```
<function class="WeatherService" method="getWeather"
name="GetWeather">
    <return type="weather">weatherData</return>
<parameter type="string">cityName</parameter>
</function>

<utility name="postalToCity" class="PostUtil"
method="postalToCity">
     <return type="alpha">cityName</return>
     <parameter type="string">postalCode</parameter>
</utility>
```

Fig. 3. Function Definitions: A Service Specification XML Fragment.

Tasks. A single task within a service represents the lowest level of interaction with an application that is understandable to the user. Tasks describe a flat pool of possible functions and define how they are invoked. Task parameters can be provided either by user input or by a stored value. In the case where a needed parameter is a stored value that is not initialized, that task can be defined as unavailable.

Each task can call on at most one service function to guarantee atomicity of tasks and avoid problems of sub-task ordering. The provision of utility functions is meant to encourage data representation issues to be separated from logic. Hence, logically

[14] A *namespace* is a unique identifier that labels a group of related items. Different groups can then use the same identifiers internally to label different items.

similar tasks may use the same underlying service function and use utility functions to manipulate the data they provide to that function.

An example task specification fragment is shown in figure 4. Note the definition of the mapping of input from the user (`<variable>` elements) to parameters of the service function (`<parameter>` elements). This mapping technique is described below.

```
<task name="Get City Weather" taskFunction="Get Weather">
    <variable type="simple">cityName</variable>
    <parameter type="alpha"
               source="task"
               store="lastCity">cityName</parameter>
</task>
```

Fig. 4. Task Definition: A Service Specification XML Fragment.

Mapping Tasks onto Functions. The data passed from tasks to their underlying function are defined in terms of input *variables* and function *parameters*. These are represented in task definitions by `<variable>` and `<parameter>` element tags. The types of parameters defined in the task exactly match the input parameters of the underlying service function. However, there need not be the same number of task parameters as variables. The manipulation of a variable to provide a parameter value is defined with the `<parameter>` element tag. It identifies the variable to be used, what mapping to perform and whether to store the generated parameter value for later use.

The default mapping, if no mapping is explicitly defined (as in figure 4), is no manipulation at all. Data is output as a parameter exactly as it is received as a variable.

```
<parameter type="alpha"
           source="task"
           mapping="utility"
           store="lastCity"
           name="postalToCity">
<parameter type="alpha"
           source="task">postalCode</parameter>
</parameter>
```

Fig. 5. Utility Mapping in a Task Parameter: A Service Specification XML Fragment.

A *utility mapping* (see figure 5) assigns a utility function to transform the data of a variable that defines a mapping from postcodes to city names. The `name` attribute identifies the utility function to use, and the nested `<parameter>` element tags describe the mapping for the utility function's parameters.

Extract mappings take an element of a record type and return one of the items within the record as specified in the parameter. (Figure 6 shows extraction of an ID value from an account record.)

```
<parameter type="alpha"
           source="task"
           mapping="extract">account
accID</parameter>
```

Fig. 6. Extract Mapping in a Task Parameter: A Service Specification XML Fragment.

Keeping Track of State. A task-based service keeps track of persistent state at a task level separately from any provision made by underlying logic. State therefore can be shared between tasks directly without the underlying logic. It is possible to support stateless implementations of the logic (such as with raw HyperText Transfer Protocol (HTTP) based systems). A task parameter can define a mapping from a state variable instead of a task variable. In figure 7, a state variable keeps track of the name of a city for which weather is requested and a task uses the name to give an update of that request.

```
<state> <variable type="string">lastCity</variable> </state>
...
<task name="Update Weather" taskFunction="Get Weather">
    <parameter type="alpha" source="store">lastCity</parameter>
</task>
```

Fig. 7. State Definition and Use in a Task Parameter: A Service Specification XML Fragment.

4.2 Interaction Style Specification

The key feature of the SIS approach is how tasks are implemented on different platforms. Each platform supports a set of presentation objects. Between the tasks and the presentation, each presentation style supports its own abstract user interface elements that gather input and display output to the user. These elements have their own distinctive way of navigating available tasks. No explicit layout or presentational information is contained in a style description; rather it is the semantic relationship between interface components that is described. It is the job of the presentation unit to resolve these relationships into an appropriate presentation.

Style instances are generated in the SIS client in order to facilitate fast user response. Therefore, events generated by presentation implementations are dealt with by style-specific, presentation-independent, objects that reside locally. The style manager generates each style instance from scratch locally on each client in order to customize a client's access to a common service.

Three styles are currently implemented but aim to provide a foundation for a potentially larger set.

4.2.1. Form-Fill Style

The style definition for a forms-based style involves: *field* elements for gathering user input, *actions* that can be invoked and a mapping from actions and fields to underlying tasks.

A field element is an abstract interactor that allows the user to enter a value to be used in a task, for example text entry, password entry, single choice, multiple choice, date entry, range entry and currency entry. Questions about whether a single choice entry would be represented by a drop-down list, radio buttons or some other selection method are deferred to platform implementation and depend on the actual data being selected and the layout constraints of the presentation. An example of a simple text field element and a single choice element are given in figure 8. The definition gives the type of the field element and the type of its value.

```
<field name="postalText" type="text"/>

-------------------------

<field name="accountChoice" type="choice" value="AccountType">
      <n-selection>1</n-selection>
      <selection-values source="utility">Get Accounts</selection-
values>
</field>
```

Fig. 8. Form-fill Style Specification: Example text field and single choice field element definitions.

Each style provides mechanisms for processing the data to produce an appropriate representation. Providers of services may specify functions that perform representational transformations. For example, in the form-fill style an output processor defines a set of items that can be extracted from a data type (see figure 9). Several output processors can be defined to work on the same types and used for different purposes.

```
<processor name="weatherOut" type="text">
     <input class="WeatherData">weatherData</input>
     <converter class="WeatherData">
        <item>
           <source>weatherData</source>
           <method>getWeatherText</method>
        </item>
     </converter>
</processor>
```

Fig. 9. Form-fill Style Specification: An example output definition.

A form is built out of fragments that map a set of fields to the inputs of a particular task. A fragment's task is only invoked if the requirements of the fields of that fragment are satisfied. A fragment also specifies an output processor that can extract information from the output of the task.

```
<form_fragment name="cityForm">
     <task>Get City Weather</task>
     <input req="mandatory">cityText</input>
     <output type="text">weatherOut</output>
</form_fragment>
```

Fig. 10. Form-fill Style Specification: An example form fragment definition.

This definition (figure 10) outlines a hierarchy of actions that may be invoked by a user and associates with each action a set of form fragments that are evaluated when that action is invoked. Typically an action would be invoked by the user pressing a submit button to indicate completion of the form ready for processing. An action is a semantic unit within the form. Trees of actions, together with form fragments allow a presentation to compose a form representation. The presentation decides whether fields are presented on several "pages" or on a single "page" and use different buttons to invoke different actions.

4.2.2. Dialogue Style

Dialogue style definitions are described by a set of grammars of input token combinations. Dialogue structures make use of these grammars to move between elements of the dialogue. A grammar used in a transition between states is called a *match set* and contains a list of match items that can be matched by a series of tokens in input. For example in figure 12 `<matchitem>` contains a main `<token>` whose contents must match the next input token and optionally a list of match items that can be matched after that token. Items are evaluated in list order. As soon as an item matches, no more items in a list are evaluated. An item only matches if its main token matches *and* one of its sub items matches. That a possibility is optional is supported by a special `<lambda>` match item that is matched if no other items in a list are matched.

```
<matchset name="CityMatch">
     <matchitem>
       <token>city</token>
       <matchitem>
       <token>name</token>
       </matchitem>
       <lambda/>
     </matchitem>
</matchset>
```

Fig. 11. Dialogue Style Specification: An example match set definition fragment.

The dialogue structure is a tree of states that has special task-invoking states as the leaf nodes in the tree (see figure 12). States are defined with `<dialogue-state>` element tags and contain possibly conditional prompts that are displayed if the dialogue stops at that state. A `transition` attribute identifies match sets or stored variables that a user's input must match. After a task is invoked, the dialog restarts at the root of the tree.

```
<dialogue-state>
      <prompt source="GetWeatherPrompt"/>
      <prompt source="GetUpdatePrompt">
       <condition task="Update Weather">
          <name>available</name>
          <value>true</value>
       </condition>
      </prompt>
       <dialogue-state transition="CityMatch">
           <prompt source="CityInput"/>
           <dialogue-state transition="$CITYVAR">
              <prompt source="CityWeather"/>
           </dialogue-state>
       </dialogue-state>
...
</dialogue-state>
```

Fig. 12. Dialogue Style Specification: An example dialogue tree definition fragment.

Task invocations are defined in special states that define the underlying task to be invoked, which dialogue variables to use, and the response to be generated with the output (figure 13).

```
<response name="weatherResponse" class="WeatherData">
   <output type="text">
     <method>getWeatherText</method>
   </output>
</response>
<task-state name="PostWeather">
   <task>Get Postal Weather</task>
   <parameter>$POSTVAR</parameter>
   <response>weatherResponse</response>
</task-state>
```

Fig. 13. Dialogue Style Specification: An example task state definition fragment.

Prompts can be either predefined questions or the response from a task invocation. Responses can also be shared between task instances. User variable input is transferred to the task states by use of a set of defined variables. The name of these variables can be used in place of a grammar match set in a transition between states.

4.2.3. Menu Style

A menu-based interface is specified by a tree of menu items (see figure 14). Each node representing an item has a label and an optional description of a task invocation. Only the leaves of the tree can have task invocations. Details of the task are wrapped into the menu item specification, with the name of the task and an output data extraction defined as usual, together with a list of inputs. Inputs can have a label to be displayed to the user when entering that input.

```
<menu-item>
    <label>Weather by PostCode</label>
    <task>Get Postal Weather</task>
    <input type="string">
        <name>postalCode</name>
        <label>Enter postal code</label>
    </input>
    <output class="WeatherData" method="getWeatherText"/>
</menu-item>
```

Fig. 14. Menu Style Specification: An example menu item definition.

This current version is limited to descriptions of simple menus, but as an aim of the specifications is to simplify interface definition for simple interfaces, the descriptions are also simple. It is envisioned that the specification will be extended to cope with more complicated menu semantics and user input.

4.3 Presentation

Presentation units run on the client device and prescribe a concrete user interface for style definitions. Each style will have a presentation unit tailored for it that runs on a particular device. A client presentation unit utilizes a reference to a remote service instance and the appropriate style instance. They give access to the internal object representations of tasks and the elements of styles. When a task is to be invoked, it passes the appropriate data to the service instance.

Current implemented presentation units use simple techniques to deal with physical layout and representational issues. An expansion of the presentation component in the future might include dealing with details of physical layout in an abstract way.

5 Creating Interfaces with Styles

An example weather service together with definitions of the three different styles of interfaces described above, and their rendering by presentation units is now described. The service provides a single function that returns a textual description of the weather for a given location supplied as a string.

5.1 The AnyWeather Service

The weather query service is described by a XML task specification for the service shown in figure 15. Three separate tasks perform the service:

1. Request the weather for a city by name (“`Get City Weather`”)
2. Request the weather for a city by postcode (“`Get Postal Weather`”)
3. Refresh the last weather request (“`Update Weather`”)

Requesting the weather for a city by name utilizes the underlying service function “`Weather Service`” directly, while a post-code based request requires the use of an external utility function, "`postalToCity`", to convert postcodes to city names. The "`Update Weather`" task utilizes a state store object to keep track of the last city for which weather was requested.

```
<service location="http://www-
users.cs.york.ac.uk/~steveg/weather/">
<function class="WeatherService" method="getWeather" name="Get
Weather">
<return type="weather">weatherData</return>
<parameter type="string">cityName</parameter>
</function>
   <utility name="postalToCity" class="PostUtil"
method="postalToCity">
       <return type="alpha">cityName</return>
       <parameter type="string">postalCode</parameter>
   </utility>
   <state>
      <variable type="string">lastCity</variable>
   </state>
<task name="Get City Weather" taskFunction="Get Weather">
<variable type="simple">cityName</variable>
        <parameter type="alpha"
                   source="task"
                   store="lastCity">cityName</parameter>
</task>
<task name="Get Postal Weather" taskFunction="Get Weather">
<variable type="simple">postalCode</variable>
        <parameter type="alpha"
                   source="task"
                   mapping="utility"
                   store="lastCity"
                   name="postalToCity">
<parameter type="alpha" source="task">postalCode</parameter>
</parameter>
</task>
     <task name="Update Weather" taskFunction="Get Weather">
         <parameter type="alpha"
source="store">lastCity</parameter>
     </task>
</service>
```

Fig. 15. AnyWeather task specification.

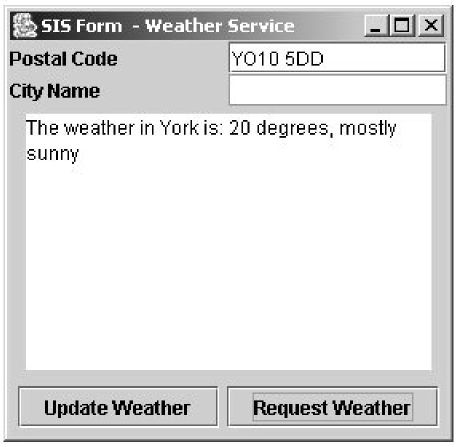
5.2 Form-Fill Interface

The specification of the form-fill style for the AnyWeather service is shown in figure 16. Two fields are defined, one to enter city names ("`cityText`") and one to enter postcodes ("`postalText`"). A processor ("`weatherOut`") extracts the description of the weather from a `WeatherData` output object. Three form fragments, for each of the three tasks, use the defined processor for output and the two fields as inputs. The `<sub-form>` definitions match the form fragments to an action and a single display.

```
<style type="form"
       location="http://www.users.cs.york.ac.uk/~steveg/weather">
    <field name="cityText" type="text" />
    <field name="postalText" type="text" />
    <processor name="weatherOut" type="text">
        <input class="WeatherData">weatherData</input>
        <converter class="WeatherData">
           <item>
              <source>weatherData</source>
              <method>getWeatherText</method>
           </item>
        </converter>
    </processor>
    <form_fragment name="cityForm">
         <task>Get City Weather</task>
         <input requirement="mandatory">cityText</input>
         <output type="text">weatherOut</output>
    </form_fragment>
     …
    <form>
        <display type="text">weatherDisplay</display>
        <action-set>
            <action-set name="getWeather">
               <action name="getCity"/>
               <action name="getPostal"/>
            </action-set>
            <action name="updateWeather"/>
        </action-set>
        <sub-form>
           <fragment>cityForm</fragment>
           <action>getCity</action>
           <display>weatherDisplay</display
        </sub-form>
…
    </form>
</style>
```

Fig. 16. AnyWeather form-fill style specification.

The form-fill presentation unit renders the form components on a single screen with two buttons representing the first sub-level of the action tree (see figure 17). The interface uses the requirements of the form fragments to evaluate which of the two user input tasks to invoke when the “Get Weather” button is pressed. The interface is told that “City Name” is mandatory for the “Get City Weather” task, but not required for the “Get Postal Weather” task, so if a city name is entered it can assume that the city task is required, and the button will invoke that task. In addition all non-required fields of that task will be disabled to help indicate which task has been chosen.



Postal Code: YO10 5DD
City Name:

The weather in York is: 20 degrees, mostly sunny

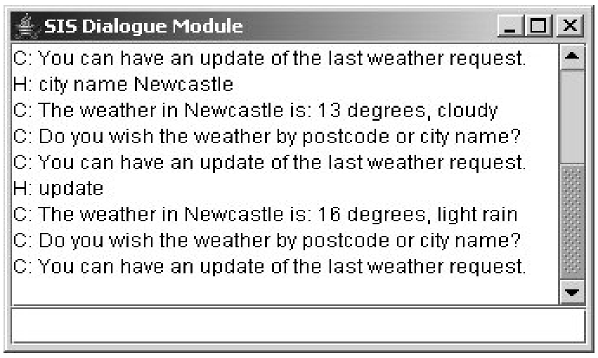
Fig. 17. Weather Service form-fill interface.

5.3 Dialogue Interface

The specification of the dialog style for the AnyWeather service is shown in figure 18. Prompts are defined for the initial dialog state and for requesting user input. A response extracts the weather description from a WeatherData object in much the same way as for the form-fill style. A task state for each of the available
tasks is assigned a response and an appropriate variable. Three match set grammars let a user enter a variety of phrases to select each of the tasks. For instance, a user can enter “postcode”, “postal code” or just “postal” to access the `Get Postal Weather` task. A dialogue with three paths leads to the three tasks. The paths to the user input tasks have two states, one of which prompts the user to enter the appropriate input if it is not already in the token string. The update task doesn't require user input so only requires one state transition to reach it. The presentation unit for the dialogue renders the interface shown in figure 19.

```
<style type="dialogue">
<question name="GetWeatherPrompt">...</question>
<question name="GetUpdatePrompt">...</question>
<question name="CityInput">...</question>
<question name="PostInput">...</question>
<response name="weatherResponse" class="WeatherData">
     <output type="text"><method>getWeatherText</method></output>
</response>
<task-state name="PostWeather">
     <task>Get Postal Weather</task>
     <parameter>$POSTVAR</parameter>
     <response>weatherResponse</response>
</task-state>
...
<matchset name="PostMatch">
    <matchitem>
       <token>postcode</token>
    </matchitem>
    <matchitem>
       <token>postal</token>
      <matchitem>
         <token>code</token>
      </matchitem>
      <lambda/>
    </matchitem>
</matchset>
...
<dialogue-state>
   <prompt source="GetWeatherPrompt"/>
   <prompt source="GetUpdatePrompt">
     <condition task="Update Weather">
            <name>available</name>
             <value>true</value>
     </condition>
   </prompt>
    ...
   <dialogue-state transition="PostMatch">
        <prompt source="PostInput"/>
        <dialogue-state transition="$POSTVAR">
            <prompt source="PostWeather"/>
        </dialogue-state>
   </dialogue-state>
   ...
</dialogue-state>
</style>
```

Fig. 18. AnyWeather dialogue style specification.



```
C: You can have an update of the last weather request.
H: city name Newcastle
C: The weather in Newcastle is: 13 degrees, cloudy
C: Do you wish the weather by postcode or city name?
C: You can have an update of the last weather request.
H: update
C: The weather in Newcastle is: 16 degrees, light rain
C: Do you wish the weather by postcode or city name?
C: You can have an update of the last weather request.
```

Fig. 19. Weather Service dialogue interface.

```
<style type="menu"
       location="http://www-users.cs.york.ac.uk/~steveg/weather">
       <menu>
          <title>Weather Service Menu</title>
          <menu-item>
             <label>Weather by City</label>
             <task>Get City Weather</task>
             <input type="string">
                <name>cityName</name>
                <label>Enter a city name</label>
             </input>
             <output class="WeatherData" method="getWeatherText"/>
          </menu-item>
          <menu-item>
             <label>Weather by PostCode</label>
             <task>Get Postal Weather</task>
             <input type="string">
                 <name>postalCode</name>
                 <label>Enter postal code</label>
             </input>
             <output class="WeatherData" method="getWeatherText"/>
          </menu-item>
          <menu-item>
              <label>Update Weather</label>
              <task>Update Weather</task>
              <output class="WeatherData" method="getWeatherText"/>
          </menu-item>
       </menu>
</style>
```

Fig. 20. AnyWeather menu style specification.

5.4 Menu Interface

The specification for the menu style of interface for AnyWeather is shown in figure 20. All three tasks are available from the main menu, one item per task. The two tasks requiring user input have inputs fields rendered as separate entry screens in a menu presentation implementation as shown in figure 21.

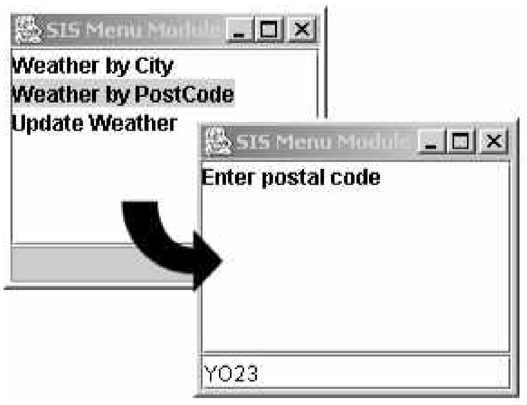


Fig. 21. Weather Service menu interface.

6 Discussion

The specifications in SIS separate the specification of the functionality of a ubiquitous application from the specification of its interface and provide a selection of different styles of interface so that an interface can more closely match the capabilities and limitations of a device. Both achievements are consistent with the original requirements of User Interface Management Systems (UIMS). Having a clean separation of function and interface has particular advantages when providing a selection of interface descriptions. It is clearly less important when providing a single "canonical" interface as in the case of XWeb and UIML (as discussed in section 2.3) or a UIMS vision based around a single type of device.

SIS achieves this separation by making the abstraction of functionality very simple. Any semantic relationships between the tasks must occur at the style level. In the AnyWeather service the relationship of tasks in the form-fill style (figure 16) is different from the dialogue style (figure 18), and this would be the case however systematically the layering was achieved.

Style specifications do not dictate how a presentation unit displays the information conveyed in the style. Presentation units on different devices display a style in different ways to fit that device even though the style definition is the same on each device. Applications can therefore use native applications on devices by having a presentation unit that renders interfaces in a way that is consistent with them. For instance a presentation unit could choose to display the AnyWeather form-fill actions as three separate buttons, rather than two, or indeed display the three sub-forms on different screens.

Although AnyWeather is designed to be simple to illustrate the basic ideas, more features can be added to each of the different styles. A further application of these features demonstrating SIS is based around an internet banking scenario. In this case more complex data types need to be supported, and this requires development of a richer type system. List and record types can be implemented to help support more complex applications as well as user-defined custom types (similar to those in XWeb).

The relative size of dialogue style definitions might be said to be in conflict with the requirements for definitions for simple interfaces to be simple themselves. However, the benefit of having a clear, extensible specification means that the parsing engine of the system can be much simpler and allows for better integration with simple tools. In future, size might be alleviated without affecting the parsing engine by using transformations from more concise specifications into the current versions.

7 Conclusion

A model of interaction style has been devised that can be used to provide a range of possible interfaces to be presented on a device. Basing a single interface specification on simple (yet still abstract) concepts can work, but is limited if target devices are too diverse in their interactive capabilities. Conversely, tying the specification too closely to the capabilities of any one device leads to the situation of having a different specification for each device. Having a finite set of styles specifications can be complex enough to make fuller use of devices capabilities yet different and flexible enough to work on a wide range of devices. Interaction styles have potential to be viable for defining interfaces for ubiquitous interactive systems on many devices. Additional applications will provide the impetus for expanding the features of SIS, and demonstrate its potential and flexibility.

References

1. Newman, W., Lamming, M: Interactive System Design. Addison-Wesley (1995) 293—322
2. Shneiderman, B: Designing the User Interface, 3rd edition. Addison Wesley Longman (1998) 71-74
3. Edmonds, E.: The emergence of the separable user interface. In Edmonds, E., ed.: The Separable User Interface. Academic Press (1992) 5-18
4. Vanderdonckt, J.: Current trends in computer-aided design of user interfaces. In Vanderdonckt, J., ed.: Computer-Aided Design of User Interfaces Proc.of CADUI '96. Namur University Press (1996) xiii-xix
5. Abowd, G., Schilit, B.N.: Ubiquitous computing: The impact on future interaction paradigms and HCI research. In: CHI97 Extended Abstracts. (1997)
6. Olsen, D.R., Jefferies, S., Nielsen, S.T., Moyes, W., Fredrickson, P.: Cross-modal interaction using XWeb. UIST 2000. (2000) 191-200
7. Myers, B.A.: A new model for handling input. ACM Transactions on Information Systems (TOIS) **8** (1990) 289-320
8. Ponnekanti, S.R., et~al.: ICrafter: A service framework for ubiquitous computing environments. In: Proceedings of Ubicomp 2001. LNCS 2201 (2001) 56-75
9. Eisenstein, J., Vanderdonckt, J., Puerta, A.: Applying model-based techniques to the development of UIs for mobile computers. In: IUI01:2001 International Conference on Intelligent User Interfaces. (2001) 69—76

10. Muller, A., Forbrig, P., Cap, C.H.: Model-based user interface design using markup concepts. In: DSV-IS. Volume 2220 of Lecture Notes in Computer Science, Springer (2001) 16-27
11. Phanouriou, C.: UIML: A Device-Independent User Interface Markup Language. PhD thesis, Virginia Tech (2000)
12. Abrams, M., Phanouriou, C., Batongbacal, A., Williams, S.: UIML: an appliance-independent XML user interface language. In: Computer Networks. Volume 31. (1999) 1695-1708
13. Turau, V.: A framework for automatic generation of web-based data entry applications based 0on XML. In: ACM Symposium on Applied Computing (SAC 2002). (2002)

Discussion

[Gerrit van Deer Veer] You did not mention/elaborate interaction styles “direct manipulation” nor “command language”. DM requires complex representation of n-dimensional interaction space and n-degrees of freedom user act to, command language seem completely upprite(?). Also, in envisioning scenarios of companies like Philips, NTT, Sun (“starfire”) these styles are mixed.

> [Stephen Gilroy] We did not elaborate DM: it’s very complex. We considered mixed styles (?). their analysis / Specification would be separate/unconnected.

[Ann Blanford] Walk-up-and-use isn’t just device or just context – it’s a tuple of device, context, user, task(s). i.e. There are combinations that work together and often that don’t. Can these combinations make style selections simpler ?

> [Stephen Gilroy] Yes.

[Kevin Schneider] Within your categorization of interaction styles, are there different styles for each device ? For example, would there be a different interaction style for filling in a form on a PC versus filling in a form on a PDA.

> [Stephen Gilroy] No, it would be the same style. The device would handle the different presentations.

Supporting Flexible Development of Multi-Device Interfaces

Francesco Correani, Giulio Mori, Fabio Paternò

ISTI-CNR
56124 Pisa, Italy
{francesco.correani, giulio.mori, fabio.paterno}@isti.cnr.it
http://giove.isti.cnr.it

Abstract. Tools based on the use of multiple abstraction levels have shown to be a useful solution for developing multi-device interfaces. To obtain general solutions in this area it is important to provide flexible environments with multiple entry points and support for redesigning existing interfaces for different platforms. In general, a one-shot approach can be too limiting. This paper shows how it is possible to support a flexible development cycle with entry points at various abstraction levels and the ability to change the underlying design at intermediate stages. It also shows how redesign from desktop to mobile platforms can be obtained. Such features have recently been implemented in a new version of the TERESA tool.

1 Introduction

Model-based approaches [10, 13] have long been considered for providing support to user interface design and development. Recently, such approaches have received further attention because of the challenges raised by multi-device environments [1, 4, 6, 13]. The use of tools based on logical abstractions enables adapting the interfaces under development to the characteristics of the target devices. This can simplify the work of designers who do not have to address a proliferation of devices and related implementation details.

The potential logical descriptions to consider are well identified, and their distinctions are clear [3]: task models represent the logical activities to perform in order to reach users' goals; object models describe the objects that should be manipulated during task performance; abstract user interfaces provide a modality independent description of the user interface in terms of main components and logical interactors; concrete user interfaces provide a platform-dependent description identifying the concrete interaction techniques adopted, and lastly the user interface implements all the foregoing.

Various approaches have benefited from this logical framework, and tools supporting it have started to appear. In particular, there are tools that implement a forward engineering approach, which take an abstract description and generate more refined ones until the implementation is obtained; or tools supporting reverse engineering approaches, which instead take an implementation and aim to obtain a corresponding logical description. Examples of forward engineering tools are Mobi-D [13] and TERESA [6]. They both start with task models and are able to support user interface generation, though by applying different rules and additional models. TERESA is the tool for the design of multi-device interfaces developed in the EU IST CAMELEON project. It introduces the additional possibility of adapting the transformation process to the platform considered. A platform is a set of devices that share a similar set of interaction resources. Another example of tool for forward engineering is ARTstudio [4], which also starts with the task model and supports the editing of abstract and concrete user interface, but, contrary to TERESA, it generates Java code instead of Web pages and is not publicly available. Examples of different support for reverse engineering are Vaquita [2] and WebRevEnge [8]. The first one provides the possibility of rebuilding the concrete description of Web pages, whereas the latter reconstructs the task model corresponding to the Web site considered. In both cases one limitation is the lack of support for the reverse engineering of Web sites implemented using dynamic pages.

The needs and background of software developers and designers can vary considerably, and there is a need for more flexible tools able to support various transformations in the logical framework mentioned. To this end, we have designed and implemented a new version of the TERESA tool, aiming to provide new possibilities with respect to the original version [6]. In particular, the new version that is presented in this paper supports multiple entry points in the development process and the redesign of a user interface for a different platform.

In the paper we first recall the basic design criteria of the original version of the TERESA tool and then we dedicate one section to describing how multiple entry points can be supported and one for the transformation for redesign from desktop to mobile. We then show examples of applications of such new features and, lastly, we draw some conclusions and indications for future work.

2 The initial TERESA environment

The TERESA tool was originally designed to support the development of multi-device interfaces starting with the description of the corresponding task model. In order to facilitate such a development process the main functionality of the CTTE tool [7], supporting editing, analysis, and interactive simulation of task models, have been integrated into the new tool. So, once designers have obtained a satisfying task model, they can immediately change mode and use it to start the generation process. The tool provides automatic transformation of the task model into an abstract user interface structured into presentations. For each presentation, the tool identifies the associated logical interactors [11] and provides declarative indications of how such interactors should be composed. This is obtained through composition operators that have been defined taking into account the type of communication effects that designers aim to achieve when they create a presentation [8].

The composition operators identified are:

- Grouping (G): indicates a set of interface elements logically connected to each other;
- Relation (R): highlights a one-to-many relation among some elements, one element has some effects on a set of elements;
- Ordering (O): some kind of ordering among a set of elements can be highlighted;
- Hierarchy (H): different levels of importance can be defined among a set of elements.

In addition, navigation through the presentations is defined taking into account the temporal relations specified among tasks. The abstract user interface description can then be refined into a concrete user interface description, whereby a specific implementation technique and a set of attributes are identified for each interactor and composition operator, after which the user interface implementation can be generated. Currently, the tool supports implementations in XHTML, XHTML mobile device, and VoiceXML (one version for multimodal user interfaces in X+V and one version for graphical direct manipulation interfaces are under development).

3 Support for Flexible Forward Engineering

Interface design is complex. Often, as designers go through the various steps in order to develop suitable solutions for the current abstraction level, they would like to reconsider some of the choices made earlier in an iterative process. Furthermore, the actual results of automatic transformations may not be precisely those expected and thus would need to be refined. Lastly, the need to provide relevant support to a flexible methodology requires the ability to offer different entry points.

The original version of the TERESA tool provided a concrete solution to the issue of supporting development of multi-device interfaces through various levels of automation. However, when designers selected the completely automatic solution sometimes it happened that what they get was rather different from what they wanted (Figure 1 shows an example [12]). Thus, there was a need for providing designers with better support for tailoring the transformations to their needs.

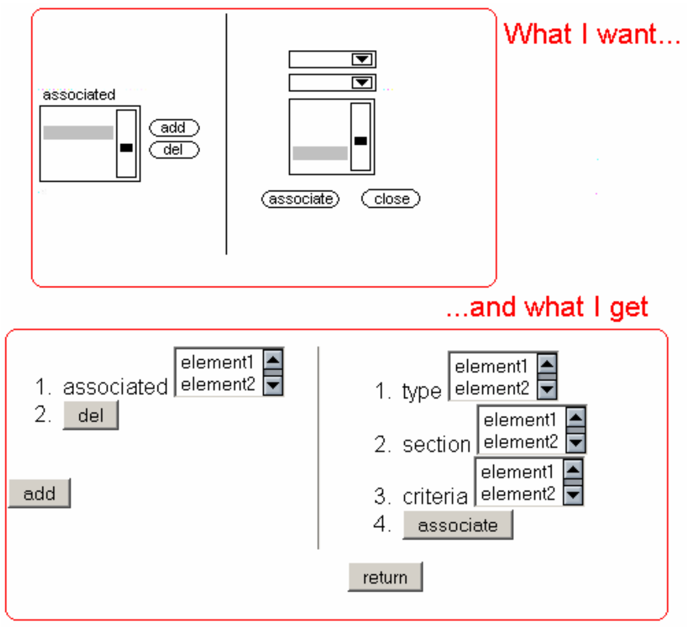


Fig. 1. Example of mismatch between designer's goals and result of automatic generation.

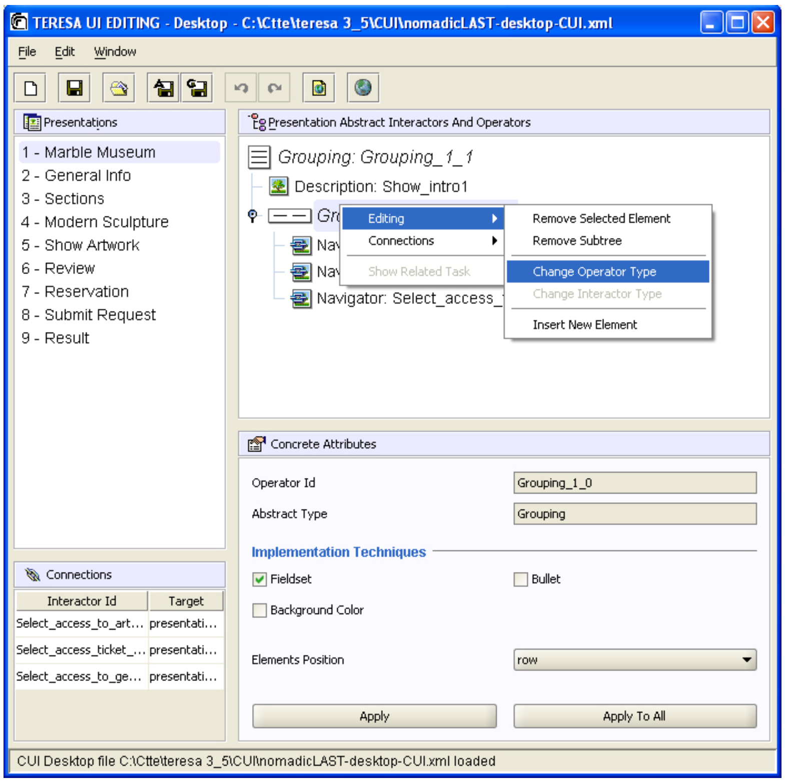
Once a suitable description of the abstract user interface has been obtained from a given task model, it is important that its properties be adjusted to increase usability for the generated presentations. Designers may also decide to start defining the abstract interface from scratch, bypassing the task modelling phase.

In order to deal with all these issues we decided to extend TERESA functionalities by adding new features, in particular, enabling changes, even radical ones, in the properties of abstract user interface elements and the ability to develop an abstract user interface from scratch.

Once an abstract user interface has been created, there are various levels of modifications that can be possible:

- *Modifying the structure of a presentation without changing the associated interactors*. This can be performed in different ways: moving the orders of the interactors within a composition operator, changing, adding or removing composition operators;
- *Modifying the association between interactors and presentations without changing existing interactors*. This can be performed by merging or splitting existing presentations or moving one interactor from one presentation to another.
- *Modifying the set of available interactors*, this means changing the type of interactors, adding or removing interactors (this can be done by either working on single interactors or adding or removing groups of interactors or entire presentations).

In order to avoid confusing designers the editing features have to be explicitly enabled. Then, to ease the use of these functionalities, a number of features have been introduced. The type of an interactor is explicitly represented through an icon (as are the task categories in the task model) and modifications to the interactors order within a presentation can be performed through a drag and drop function. The result of a completely automatic transformation from the task model to the abstract user interface is a set of presentations (which are listed on the left side of the control panel, see Figure 2) and the related connections defining navigation through them. When one presentation is selected then its logical structure in terms of interactors and composition operators is shown in the central part. Designers can select either composition operators or interactors and the corresponding attributes are shown in the bottom part. The position of an interactor in the presentation can be moved through drag and drop interactions. If editing has been enabled it is also possible to change the type of operators and interactors. For example, in Figure 2 there is a change of a Grouping operator.



1 - Marble Museum
2 - General Info
3 - Sections
4 - Modern Sculpture
5 - Show Artwork
6 - Review
7 - Reservation
8 - Submit Request
9 - Result

Grouping: Grouping_1_1
Description: Show_intro1
Gr
Nav
Nav
Navigator: Select_access_

Interactor Id	Target
Select_access_to_art…	presentati…
Select_access_ticket_…	presentati…
Select_access_to_ge…	presentati…

Operator Id	Grouping_1_0
Abstract Type	Grouping

Implementation Techniques

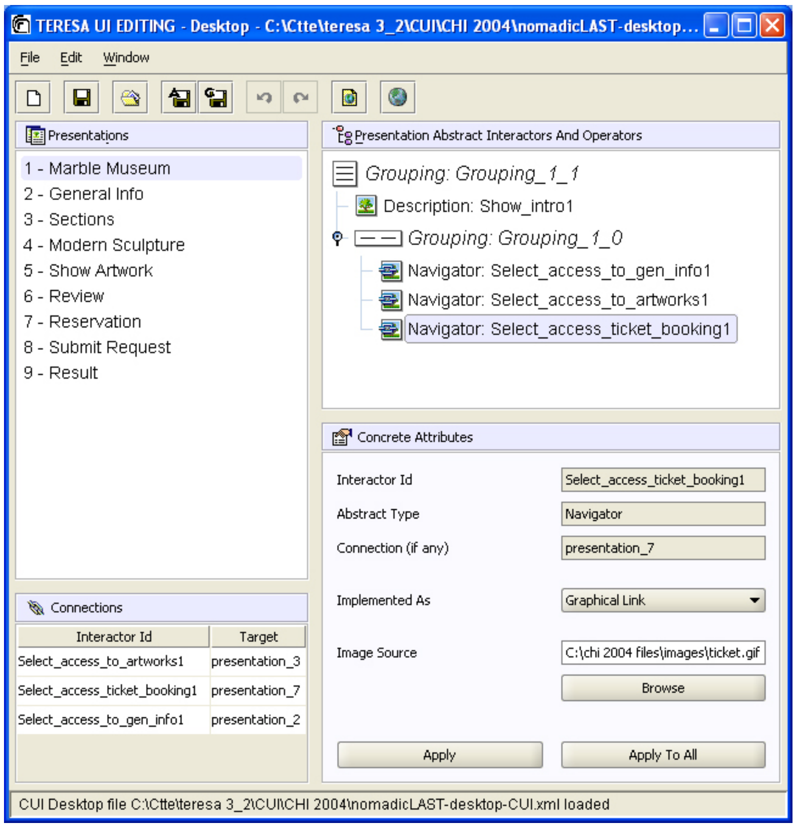
[✓] Fieldset [ ] Bullet
[ ] Background Color

Elements Position: row

CUI Desktop file C:\Ctte\teresa 3_5\CUI\nomadicLAST-desktop-CUI.xml loaded

Fig. 2. Example of change of composition operator.

The editor of the abstract user interface (see Figure 3) provides designers with a view on various aspects that can be modified. One panel indicates the list of presentations defined so far. The logical structure of the currently selected presentation is shown as well. It can be represented either showing the logical structure in a tree-like manner or through the list of the elements composing it. The concrete aspects of the currently selected interactor are displayed in a separate panel. For example, in the figure a navigator interactor has been selected and its identifier, type, concrete implementation (in this case through a graphical link) and related attributes (in this case the image) are shown in the associated panel. Even the navigation through the various presentations is represented and can be edited: it is defined by a list of connections, each one defined by the interactor that triggers the change and the target presentation. The tool also provides the possibility of showing the corresponding XML-based specification and the logs of the designer interactions with the tool.


TERESA UI EDITING - Desktop - C:\Ctte\teresa 3_2\CUI\CHI 2004\nomadicLAST-desktop...
File
Edit
Window
Presentations
Presentation Abstract Interactors And Operators
Concrete Attributes
Connections
Browse
Apply
Apply To All

1 - Marble Museum
2 - General Info
3 - Sections
4 - Modern Sculpture
5 - Show Artwork
6 - Review
7 - Reservation
8 - Submit Request
9 - Result

Grouping: Grouping_1_1
Description: Show_intro1
Grouping: Grouping_1_0
Navigator: Select_access_to_gen_info1
Navigator: Select_access_to_artworks1
Navigator: Select_access_ticket_booking1

Interactor Id	Select_access_ticket_booking1
Abstract Type	Navigator
Connection (if any)	presentation_7
Implemented As	Graphical Link
Image Source	C:\chi 2004 files\images\ticket.gif

Interactor Id	Target
Select_access_to_artworks1	presentation_3
Select_access_ticket_booking1	presentation_7
Select_access_to_gen_info1	presentation_2

CUI Desktop file C:\Ctte\teresa 3_2\CUI\CHI 2004\nomadicLAST-desktop-CUI.xml loaded

Fig. 3. Tool support for editing the abstract and the concrete user interface.

Lastly, a preview of the associated interface can be provided in order to allow designers to get a more precise idea of the resulting interface. Figure 4 shows the interface corresponding to the abstract/concrete presentation in Figure 3. Three navigator interactors are implemented through graphical links to other points in the application, and are grouped on the same row. In turn, this group is included in an additional group arranged vertically together with a description element that is implemented through images and text.

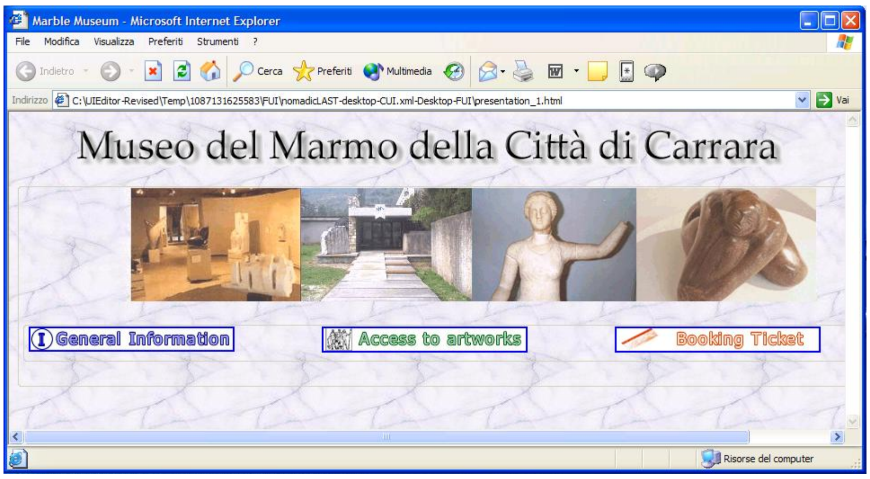

Marble Museum - Microsoft Internet Explorer
File Modifica Visualizza Preferiti Strumenti ?
Indietro Cerca Preferiti Multimedia
Indirizzo C:\UIEditor-Revised\Temp\1087131625583\FUI\nomadicLAST-desktop-CUI.xml-Desktop-FUI\presentation_1.html
Vai
Museo del Marmo della Città di Carrara
General Information
Access to artworks
Booking Ticket
Risorse del computer

Fig. 4. The user interface corresponding to the concrete interface obtained through preview.

4 Support for Redesign

Nowadays many devices provide access to Web pages: computers, mobile phones, PDAs, etc. Often there is a need for redesigning the user interface of an application for desktop systems into a user interface for a mobile device. Some authors call this type of transformation graceful degradation [5]. One main difference between such platforms is the dimension of the screen (a mobile phone cannot support as many widgets as a desktop computer in a presentation), so the same page will be displayed differently or through a different number of pages on different devices. Transcoding techniques (such as those from HTML to WML) are usually based on syntactical analysis and transformations, thus producing results which are poor in terms of usability because they tend to propose the same design in devices with different possibilities in terms of interaction resources.

In this section we describe the solution adopted to transform pages written for a desktop computer into pages for a mobile phone. In our transformation we have classified the type of mobile phones based on the screen size and other parameters, which determine the number of widgets that can be supported in a presentation. We thus group such devices into three categories: large, medium or small. In the transformation we consider that a Web page for a specific device can display a limited number of interactors [11] that depends on the type of platform. Obviously, the number of interactors supported in a desktop presentation will be greater than the number of interactors contained in a mobile phone presentation, so a desktop Web presentation will be divided into many mobile phone presentations to still support interactions with all the original interactors.

In our transformation we consider the user interface at the concrete level. This provides us with some semantic information that can be useful for identifying meaningful ways to split the desktop presentations along with the user interface state information (the actual implemented elements, such as labels, images, …). We also consider some information from the abstract level (see Figure 5): in particular the abstract level indicates what type of interactors and composition operators are in the presentation analysed. The redesign module analyses such inputs and generates an abstract and concrete description for the mobile device from which it is possible to automatically obtain the corresponding user interfaces. The redesign module also decides how abstract interactors and composition operators should be implemented in the target mobile platform. Thus, settings and attributes should change consequently depending on the platform. For example, a grouping operator can be represented by a field set in a desktop page but not in a page for a small mobile phone.

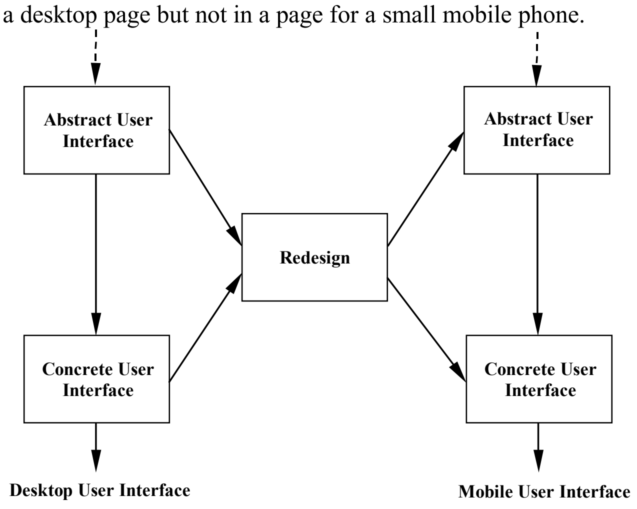
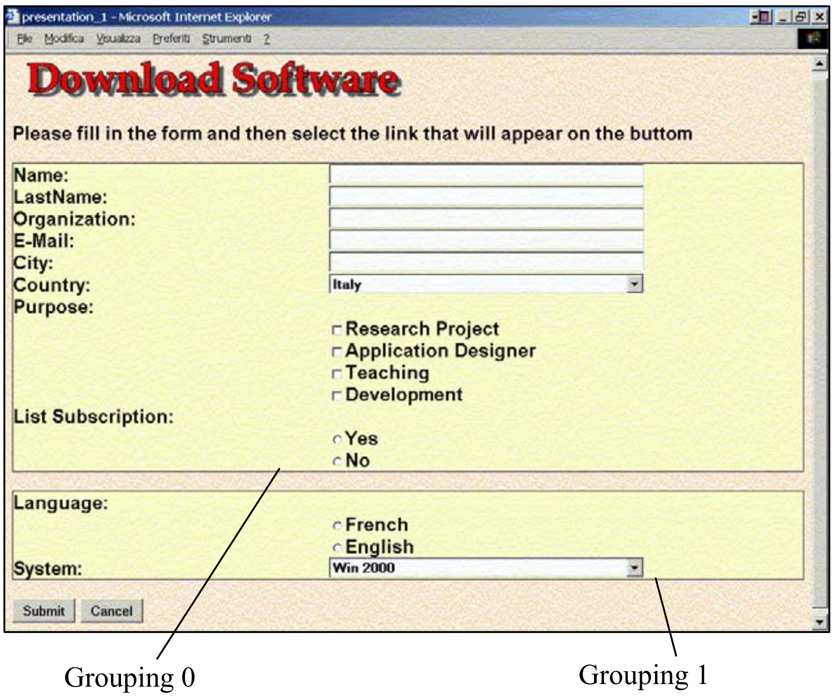


Fig. 5. The architecture of the redesign feature in TERESA.

In order to automatically redesign a desktop presentation for a mobile presentation we need to consider the limits of the available resources and semantic information. If we only consider the physical limitations we could divide large pages into small pages which are not meaningful. To avoid this, we also consider the composition operators indicated in the presentation specification. To this end, the algorithm tries to maintain groups of interactors (that are composed through some operator) for each page, thus preserving the communication goals of the designer. However, this is not always possible because of the limitations of the target platform. In this case, the algorithm aims to equally distribute the interactors into presentations of the mobile device. For example if the number of interactors supported for a large mobile presentation is six, and a desktop presentation contains a *Grouping* with eight interactors, this can be transformed into two mobile presentations, each one containing respectively a Grouping of four interactors. Since the composition operators capture semantic relations that designers want to communicate to users, this seems to be a good criterion for identifying the elements that are logically related and should be in the same presentation. In addition, the splitting of the pages requires a change in the navigation structure with the need of additional navigator interactors that allow the access to the newly created pages. The transformation also considers the possibility of modifying some interface elements. For example, the images are either resized or removed if there is no room for them in the resulting interfaces.


presentation_1 - Microsoft Internet Explorer
File Modifica Visualizza Preferiti Strumenti ?
Submit
Cancel
Grouping 0
Grouping 1

Download Software

Please fill in the form and then select the link that will appear on the buttom

Name:	
LastName:	
Organization:	
E-Mail:	
City:	
Country:	Italy
Purpose:	
	Research Project
	Application Designer
	Teaching
	Development
List Subscription:	
	Yes
	No

Language:	
	French
	English
System:	Win 2000

Fig. 6. Example of desktop Web user interface.

In order to explain the transformation we can consider a specific example of a desktop Web site and see how one of its pages (Figure 6) can be transformed using our method. The automatic transformation starts with the XML specification of the Concrete Desktop User Interface and creates the corresponding DOM tree-structure. The concrete user interface contains *interactors* (such as text, image, text_edit, single_choice, multiple_choice, control, etc) and *composition operators* (grouping, ordering, hierarchy or relation) which define how to structure them. A composition operator can contain other interactors and also other composition operators. Figure 7 represents the tree-structure of the XML file for the *desktop_ Download* presentation shown in Figure 6.

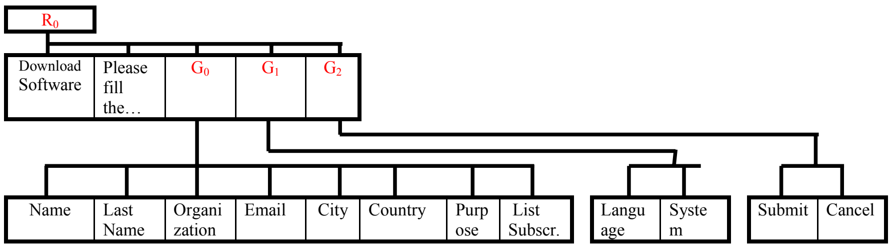


Fig. 7. Tree-structure of XML file for the "desktop_Download" *presentation*.

The resulting structure contains the following elements:

- composition operator R_0 , contains 2 interactors ("Download Software", "Please fill the form…") and 3 groupings (G_0, G_1, G_2);
- composition operator G_0 , contains 8 interactors (Name, Lastname, Organization, Email, City, Country, Purpose, List Subscription);
- composition operator G_1 , contains 2 interactors (Language, System);
- composition operator G_2, contains 2 interactors (Submit,Cancel);

The *relation* operator involves all the elements of the page: the elementary description interactor "Download Software", the elementary text interactor "Please fill in the form…" and the elements made up of the three aforementioned grouping operators. In general, the relation operator identifies a relation between the last element and all the other elements involved in the operator. In this case, the last element is represented by the composition operator G_2 which groups the "Submit" and "Cancel" buttons. In Figure 7 we can see the names of the interactors used in the *desktop_Download* presentation. There are also two *grouping* operators (G_0 and G_1) representing the two fieldsets in the user interface in Figure 6 and a grouping operator (G_2) involving the two buttons "Submit" and "Cancel".

Overall, this desktop presentation contains 14 interactors, which are too many for a mobile phone presentation. We assume that a presentation for a large mobile phone (such as a smartphone) can contain a maximum number of six interactors. Our transformation divides the "desktop_Download" presentation of the example into four presentations for mobile devices. Considering the tree structure of the XML specification of the Concrete User Interface in Figure 7, the algorithm makes a depth first visit starting with the root, and generates the mobile presentations by inserting elements contained in each level until the maximum number of widgets supported by the target platform is reached.

The algorithm substitutes each composition operator (in the example G_0 and G_1) that cannot fit in the presentation with a link pointing to a mobile presentation containing their first elements. In this case the two links point to the *mobile_Download2* and *mobile_Download4* presentations, which contain the first elements of G_0 (i.e., "Name") and the first elements of G_1 (i.e., "Language"), respectively.

So looking at the example, the algorithm begins to insert elements in the first "*mobile_Download1*" presentation and when it finds a composition operator (such as G_0), it starts to generate a new mobile presentation with its elements; so we obtain:

mobile_download1 = {**R(**"Download Software", "Please fill the form…", G_0, ….**)**}

The composition operator for the elements in mobile_Download1 is the Relation R_0. Continuing the visit, the algorithm explores the composition operator G_0. It has 8 elements but they cannot fit in a single new presentation. Thus, two presentations are created and the algorithm distributes the elements equally between them. We obtain:

mobile_Download2 = {**G(**Name, Lastname, Organization, Email**)**}
mobile_Download3 = {**G(**City, Country, Purpose, List Subscription**)**}

The composition operator for these two mobile presentations is grouping because the elements are part of G_0. The depth first visit of the tree continues and reaches G_1. It inserts a corresponding link in the mobile_Download1 presentation, which points to the new generated mobile_Download4 presentation where it inserts the elements of G_1.

Finally, we obtain:

mobile_Download1 = { **R(**"Download Software", "Please fill the form…", G_0, G_1, G_2) }
mobile_Download2 = {**G(**Name, Lastname, Organization, Email**)**}
mobile_Download3 = {**G(**City, Country, Purpose,List Subscription**)**}
mobile_Download4 = {**G(**Language, System**)**}

The entire last element of a Relation should be in the same presentation containing the elements composed by a Relation composition operator because it is the element that defines the association with the others elements. When the last element is another composition of elements (such as G_2), it is inserted into the presentation completely.

Thus, mobile_Download1 presentation becomes:

mobile_Download1 = { **R(**"Download Software", "Please fill the form…", "Form – part 1", "Form – Part 2", **G(**Submit,Cancel**))** }

Figure 8 shows the resulting presentations for the mobile device.

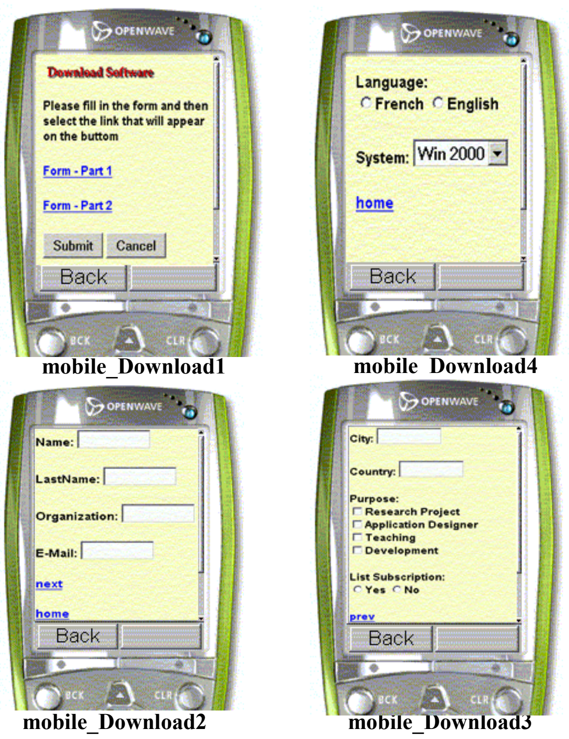


Fig. 8. Result of example desktop page transformed into four mobile pages.

4.1. Connections

The XML specifications of concrete and abstract interfaces also contain tags for connections (*elementary_connections* or *complex_connections*). An *elementary_connection* permits moving from one presentation to another and is triggered by a single interactor. A *complex_connection* is triggered when a Boolean condition related to multiple interactors is satisfied.

The transformation creates the following connections among the presentations for the mobile phone:

- original connections of desktop presentations are associated to the mobile presentations that contain the interactor triggering the transition. In the example the connection associated with the "Submit" button is asociated with the *mobile_Download1* presentation. The destination for each of these connections is the first mobile presentation obtained from the splitting of the original desktop destination presentations;

- composition operators that are substituted by a link introduce new connections to presentations containing the first interactor associated with the composition operators. In the example, we have two new links "Form - Part 1" and "Form – Part 2" which support access to the pages associated with the first interactor of G_0 and the first interactor of G_1 respectively:

  mobile_Download1 ===== Form – Part 1 ======> mobile_Download2

  mobile_Download1 ==== Form – Part 2 ======> mobile_Download4

- when a set of interactors composed through a specific operator has been split into multiple presentations we need to introduce new connections to navigate through the new mobile presentations. In the example *previous* and *next* links have been introduced automatically by the tool and we obtain the following connections:

  mobile_Download2 ===== next ======> mobile_Download3

  mobile_Download3 ===== prev ======> mobile_Download2

  the connections above, are useful to navigate between presentations "*mobile_Download2*" and "*mobile_Download3*" which contain the results of the splitting of the G_0 elements.

  mobile_Download2 ===== home ======> mobile_Download1
  mobile_Download4 ===== home ======> mobile_Download1

  the connections above are the corresponding connections for going back from presentations containing the first elements to presentations containing the links to the newly created pages. In the example, we have the "Form – Part 1" link, which is contained in "*mobile_Download1*" presentation. Likewise, we have the "Form – Part 2" link contained in "*mobile_Download1*" presentation. Thus, we need two home links that allow going back to mobile_Downolad1 from mobile_Download2 and mobile_Download4.

- complex desktop connections may need to be split into elementary connections if the associated interactors are included in different mobile presentations (in the example of Figure 6 there are no complex connections).

4.2. Other considerations

Our transformation addresses a number of further issues. Attributes for desktop presentations must be adapted to mobile presentations. For example, the maximum dimension for a font used in a desktop presentation different from the maximum for a mobile device, and consequently large fonts are resized. The transformation of desktop presentations containing images produces mobile presentations also containing images only if the target mobile devices support them. Because of the dimension of mobile screens, original desktop images need to be resized for the specific mobile device. In our classification, images are only supported by large and medium mobile phones.

In consideration of the screen size of most common models of mobile phones currently on the market, we have calculated two distinct average screen dimensions: one for large models and another for medium size. From these two average screen dimensions (in pixels), we have deduced the reasonable max dimensions for an image in a presentation for both large and medium devices. The transformed images for mobile devices maintain the same aspect ratio as those of the original desktop interface. In *mobile_Download1* presentation we have an example of resize of image "Download Software".

Interactors often do not have the same weight (in terms of screen consumption) and this has consequences on presentations. From this viewpoint, *single_selection* and *multiple_selection* interactors can be critical depending on their cardinality. For example, a single_selection composed of 100 choices can be represented on a desktop page through a list, but this is not suitable for a mobile page because users should scroll a lots of items on a device with a small screen. A possible solution could be dividing 100 choices in 10 subgroups in alphabetical order (a-c, d-f,.. ...w-z) and each subgroup is connected to another page containing a pull-down menu only composed of the limited number of choices associated with that subgroup and not of all the original 100 choices. For example, the menu for selection of a Country present in desktop presentation can be transformed as shown in Figure 9.

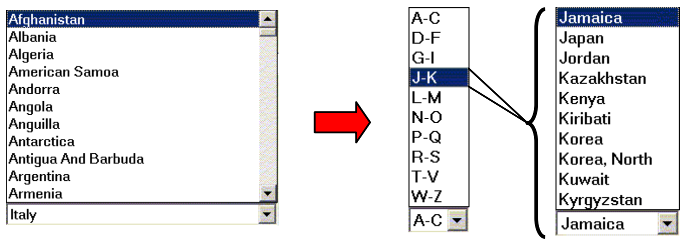


Fig. 9. Transformation of a single selection interactor for desktop system into one interactor for mobile presentations.

In the previous example of Figure 8 another simple solution has been applied, substituting the country pull-down menu of *desktop_Download* presentation with a text edit in the *mobile_Download3* presentation.

In general, the problem of redesigning and transforming a set of presentations from a platform to another is not easy and often involves many complex aspects related to user interface design.

5. Conclusions and Future Work

We have presented an approach to flexible multi-user interface design. The approach is supported by the new version of the TERESA tool, which is publicly available at http://giove.isti.cnr.it/teresa.html.

It provides designers with multiple entry points to the design process (which can be the task, abstract, or concrete user interface level) in order to change the results of automatic transformations from the task to the lower levels, and support redesign for different platforms. This last feature has also been considered in the CAMELEON project where the Vaquita tool has been used for reverse engineering of the design of a desktop Web interface. Its results are then input into the TERESA tool for redesigning for a mobile platform.

Future work will be dedicated to integrating natural interaction techniques in this environment in order to allow even people with little programming experience to easily use it in the design of multi-device interfaces. We also plan to add a feature in TERESA so that when a description at a lower level is modified, then such modifications are reflected into the description at the upper levels.

Acknowledgments

This work has been supported by the CAMELEON EU IST Project (http://giove.isti.cnr.it/cameleon.html). We also thank our colleagues in the project for useful discussions.

References

1. Abrams, M., Phanouriou, C., Batongbacal, A., Williams, S., Shuster, J. UIML: An Appliance-Independent XML User Interface Language, Proceedings of the 8th WWW conference, 1999.
2. Bouillon, L., Vanderdonckt, J., Retargeting Web Pages to other Computing Platforms, Proceedings of IEEE 9th Working Conference on Reverse Engineering WCRE'2002 (Richmond, 29 October-1 November 2002), IEEE Computer Society Press, Los Alamitos, 2002, pp. 339-348.
3. Calvary, G. Coutaz, J. Thevenin, D. Limbourg, Q. Bouillon, L. Vanderdonckt, J., "A Unifying Reference Framework for Multi-target User interfaces", Interacting with Computers Vol. 15/3, Pages 289-308, Elsevier.
4. G. Calvary, J. Coutaz, D. Thevenin. A Unifying Reference Framework for the Development of Plastic User Interfaces. IFIP WG2.7 (13.2) Working Conference, EHCI01,Toronto, May 2001, Springer Verlag Publ., LNCS 2254, M. Reed Little, L. Nigay Eds, pp.173-192.

5. Florins M., Vanderdonckt J., Graceful degradation of user interfaces as a design method for multiplatform systems, Proceedings ACM IUI'04, Funchal, ACM Press.
6. G. Mori, F. Paternò, C. Santoro, Design and Development of Multi-Device User Interfaces through Multiple Logical Descriptions, IEEE Transactions on Software Engineering, August 2004, Vol.30, N.8, pp.507-520, IEEE Press.
7. G. Mori, F. Paternò, C. Santoro, "CTTE: Support for Developing and Analysing Task Models for Interactive System Design", IEEE Transactions on Software Engineering, pp. 797-813, August 2002 (Vol. 28, No. 8), IEEE Press.
8. Mullet, K., Sano, D., *Designing Visual Interfaces*. Prentice Hall, 1995.
9. Paganelli, L., Paternò, F. A Tool for Creating Design Models from Web Site Code, International Journal of Software Engineering and Knowledge Engineering, World Scientific Publishing 13(2), pp. 169-189 (2003).
10. Paternò, F., Model-Based Design and Evaluation of Interactive Application. Springer Verlag, ISBN 1-85233-155-0, 1999.
11. Paternò, F., Leonardi, A. A Semantics-based Approach to the Design and Implementation of Interaction Objects, Computer Graphics Forum, Blackwell Publisher, Vol.13, N.3, pp.195-204, 1994.
12. Pribeanu C., Personal Communication, 2004.
13. Puerta, A., Eisenstein, J., Towards a General Computational Framework for Model-based Interface Development Systems, Proceedings ACM IUI'99, pp.171-178.
14. Puerta, A., Eisenstein, XIML: A Common Representation for Interaction Data, Proceedings ACM IUI'01, pp.214-215.

Discussion

[Stephen Gilroy] How do you deal with mis-match between interactor support on desktop and mobile platforms?

> [Fabio Paternò] The tool implements design criteria that take into account the features of the target platforms when it generates the corresponding concrete user interface. The next trasformation generates the final implementation in a language that depends on the platform. For example, it can generate XHTML for a desktop interface or XHTML Mobile Profile for a mobile interface. In case we want to support further implementation languages, such as WML, we only need to add a transformation from the concrete description for mobile devices to such implementation language. This transformation has to take into account the specific features of the new implementation language considered but it is easy to implement it because there is little distance in terms of levels of abstractions between the concrete description and the implementation language.

[José Macías] If I understand well, Teresa does the forward engineering and WebRevEnge does the reverse engineering one. Have you thought of combining both tools to obtain the whole cycle?

> [Fabio Paternò] Yes, this is the natural evolution of this research, and we think it will be very interesting to have a single tool able to suppport various levels of forward and reverse engineering.

[José Macias] How can you get the task model from an HTML page in WebRevEnge?

> [Fabio Paternò] We have analysed the most usual tasks of web applications and then we have built a tool that it is able to analyze the HTML code and identify first the corresponding basic tasks, next the tasks that are semantically related and consequently can be considered sub-task of a common higher level task, and then the temporal relations among tasks supported in one page or across multiple pages. Following this type of approach we have identified a good number of rules that are supported by the WebRevEnge tool, which is publicly available and documented in a journal publication.

[Jürgen Ziegler] Can the tool decide when a model is too complex to map to a mobile device?

> [Fabio Paternò] Not automatically; one needs to go back to the task model in order to identify tasks not suitable for a mobile device.

[Jürgen Ziegler] Can you create alternative presentations for mobile phones instead of those used on desktops?

> [Fabio Paternò] The tool generates new presentations for mobile devices according to the rules described in the paper. To this end the content for the desktop version is used and, in some cases, transformed. Future work will be dedicated to make more flexible the content transformation.

[Robbie Schaefer] Regarding the page splitting algorithm: Do you see a danger that user interfaces are generated which are processed in the wrong order by the user? What about a sequential approach?

> [Fabio Paternò] Our transformation provides results in which users have some flexibility in the choice of the order to follow when accessing the mobile pages. Users may be reluctant to process long sequences of pages on mobile phones. User evaluation has to show whether our design decision is the most appropriate.

The Software Design Board: A Tool Supporting Workstyle Transitions in Collaborative Software Design

James Wu and T.C.N Graham

School of Computing, Queen's University
Kingston, Ontario, CANADA
{wuj,graham}@cs.queensu.ca

Abstract. Software design is a team activity, and designing effective tools to support collaborative software design is a challenging task. Designers work together in a variety of different styles, and move frequently between these styles throughout the course of their work. As a result, software design tools need to support a variety of collaborative styles, and support fluid movement between these styles. This paper presents the Software Design Board, a prototype collaborative design tool supporting a variety of styles of collaboration, and facilitating transitions between them. The design of Software Design Board was motivated by empirical research demonstrating the importance of such support in collaborative software design, as well as activity analysis identifying the lack of support in existing tools for different styles of collaboration and transitions between them.

1 Introduction

The design of large, complex software systems is a team activity. A study by DeMarco and Lister found that developers working on large projects spend up to 70% of their time collaborating with others [6], while Jones found that team activities account for 85% of costs in large scale development projects [18]. This degree of interactivity between team members has necessitated the development of tools that can support collaboration within the software design process.

Designing effective collaborative design tools is a challenging task. In addition to technical and implementation issues associated with concurrent and/or distributed work, designers are hampered by a lack of data on how groups work together in software design. Collaborative applications are too often developed based on the individual experience of the designer, rather than on detailed study of the target user group and target tasks. This can result in tools that are neither useful nor usable. Even when user-centred design techniques are applied, the results are often tailored to the needs of single users, without sufficient support for collaborative work [10].

To better support collaborative work, software design tools need to support a variety of *workstyles* for collaborative interaction, as well as support fluid transitions between these workstyles. A workstyle is a characterization of the style of interaction employed by a group of collaborators, or supported by an interactive tool [36]. For example, co-located collaborators working at a whiteboard are engaged in an entirely different workstyle than distributed collaborators asynchronously sharing a document stored in a repository. In earlier work, we have shown that members of collaborative groups interact with each other through a variety of workstyles, and move frequently between different workstyles throughout the course of their interactions [37].

In this paper, we present a prototype collaborative software design tool, the Software Design Board. Software Design Board supports a variety of workstyles appropriate to the early stages of software development, and facilitates transitions between them. The functional requirements of the tool are informed by studies of existing design tools and by results of empirical research into collaborative software design activities. In presenting Software Design Board, we begin with a brief examination of related tools in the domain. As Software Design Board is primarily intended for use with an electronic whiteboard, these related tools are those that support software design through the use of informal media. Next, we present the empirical research that motivated the importance of supporting transitions in workstyle in collaborative design. We then introduce a model for characterizing styles of collaborative work, and show how this model is used to identify mismatches between collaborative activities and existing tool support. Finally, we introduce the Software Design Board and show how it supports a variety of important workstyles and workstyle transitions.

2 Tools Supporting Collaborative Software Design through Informal Media

People often carry out design work using informal media such as paper or whiteboards [20]. Particularly in the early stages of design, informal media are appropriate as they allow design diagrams to be quickly and fluidly sketched [34]. Computational analogues of such informal media include electronic whiteboards, data tablets and stylus input for computers. Tools supporting interaction with informal media attempt to extend the free form, fluid interaction afforded by physical informal media to these computational counterparts.

The main advantage of informal media tools is that they support a natural working style without imposing significant cognitive overhead on the user through heavyweight interaction mechanisms. They allow users to use the tool transparently, without having to think about the tool itself. The drawback of many of these tools is the limited, or non-existent, support for movement towards more formal, structured work. This lack of support may limit development as a design evolves and begins to require more formal treatment. Also, many of these tools are intended to be general-purpose, and lack features that may be useful in the early stages of software design.

We identify three subcategories of these tools. In each, we consider an archetype tool that is typical of the subcategory, and identify other similar tools.

- *Informal CASE Tools:* These are software design tools that support interaction through informal media. Ideogramic UML [15] is a commercial tool that evolved from the Knight research project [5]. IdeogramicUML is intended to support the “agile” use of UML [1], meaning effective and lightweight use of UML. It supports a wide variety of interaction devices, including PCs, tablets, Tablet PCs and electronic whiteboards. This tool supports gesture based modeling in UML, as well as free hand diagramming with no gestural interpretation. Furthermore, IdeogramicUML only supports co-located collaboration using electronic whiteboards, and requires additional tool support to be used by distributed teams. Other similar tools include UML Recognizer [21] and Tahuti [13].

- *Enhanced Electronic Whiteboards:* These are electronic whiteboard applications that attempt to replicate and extend the functionality of physical whiteboards using electronic whiteboards such as a Smartboard [28]. Flatland [24] is an augmented whiteboard application designed to support informal office work. Flatland provides various stylus-appropriate techniques for interaction and space management on an electronic whiteboard. Furthermore, it provides the ability to apply different behaviors to define application semantics. Flatland allows different segments on the board to respond differently to stylus input based on the applied semantics. However, it does not specifically support design tasks, but is intended to support for informal work in an office environment and as such can be appropriate in early software design tasks. Furthermore, Flatland does not support distributed collaboration, but only facilitates teamwork in a co-located setting. Other similar tools include Tivoli [25], Dolphin [30], and MagicBoard [4].
- *Shared Drawing Tools:* These tools support collaborative sketching or drawing tasks such as often found in early design work [31, 16] without providing support for any specific notation. ClearBoard [16] is a shared drawing program that allows two remote users to simultaneously draw in a shared space while providing awareness information such as hand gestures and gaze. It is based on the metaphor of 'talking through, and drawing on, a big transparent glass board' [16]. Clearboard also provides additional functionality such as simple stroke manipulations, recording of working results, as well as the ability to integrate generic files into the drawing space. Other similar tools include Commune [3], GroupSketch [11], and VideoWhiteboard [32].

Tools supporting interaction through informal media support collaboration in software design by facilitating unstructured interaction in a way appropriate to the early, creative design stages. They support an informal style of work that allows users to interact naturally and to use the tool transparently without imposing unnecessary overhead. Informal media tools support a small group of designers, and rely on social protocol to mediate group interaction. They typically produce informal artifacts of unbound semantics and free-form syntax. Most importantly for our purposes, informal media tools are typically independent of synchronicity or location, i.e. they support synchronous and asynchronous, as well as distributed and co-located interactions. This means they can support transitions in workstyle between synchronous/asynchronous and co-located and distributed styles of interaction.

3 Importance of Workstyles in Collaborative Software Design

We now present the empirical research that motivated the importance of supporting transitions in workstyle in collaborative design. We have performed extensive empirical studies into the nature of collaboration in software design [37]. We followed 5 development groups at a large software company over a 6-week period. Our research illustrated that not only is significant time spent collaborating within the design process, but also significant time and effort is spent in transitions between different collaborative styles of work. For example, team members may move frequently between asynchronous and synchronous workstyles, or between co-located and distributed workstyles, throughout the course of a single workday. These observations highlight the need for collaborative design tools that provide support for performing transitions between the various activities and working styles in which designers engage. Although some existing tools facilitate transitions in software designers' workstyles [7, 21, 12], most provide only basic communication facilities. More importantly, existing support for workstyle transitions is not commensurate with the frequency with which designers change between collaborative work styles [37, 38].

During our study, team members were observed to be highly interactive, spending on average more than two hours per day on communication tasks. Communication was predominantly face to face or via telephone or email. Also, team members often changed various aspects of their interaction such as location, synchronicity or modality of communication. These results provide evidence regarding the importance of collaboration and communication in software design, and motivate the need to support these activities in software design tools.

We also found that developers change locations frequently in order to collaborate, showing that on average, developers collaborated in more than 6 locations per day. According to interviews, this was due to a strong preference to work face-to-face. Many designers felt it was simpler, quicker and generally more efficient to use standard communication, including meeting face-to-face, than to establish remote interaction though tools. This often meant that people would walk up and down multiple flights of stairs numerous times each day to meet in person rather than use a telephone or another collaboration tool. These changes in location further indicate the frequency of workstyle transitions in collaborative software design.

Designers were also observed to frequently change the way in which they communicate, and to carry on multiple, simultaneous threads of collaboration. We found that it is typical for designers to attend a face-to-face meeting on a topic, then follow up with email, ask a supplementary question by telephone, follow up with more email, and so forth. Within individual threads of collaboration, we observed that designers change the mechanism by which they communicate more than once per day on average. These changes often involve a change in synchronicity (e.g. a change from telephone to email involves a change from synchronous to asynchronous interaction). Moreover, developers on average carried out more than three simultaneous threaded interactions in the course of a single day. All of these changes, between communication modalities, synchronicity and collaboration groups, reflect transitions in workstyle.

The results of this study have clear implications for the design of tools supporting team-based software design in large companies. These results show the importance of flexibility with respect to how a tool supports collaboration. Changes in physical location, synchronicity and communication modality are frequent, and tools should be designed to support such changes. Current tools do not sufficiently support such changes, if at all. In most existing tools, changes in synchronicity and location require a change in modality (e.g. from face-to-face to telephone) as well, imposing additional overhead on designers that choose to use them. More information on these empirical results can be found in the full study [37].

3 Understanding Workstyles

The *Workstyle Model* [36] allows us to characterize styles of collaborative work, either those employed by a group or supported by a tool. We can use these characterizations to identify mismatches between common activities and available tool support. These mismatches highlight areas where additional tool support is needed within a domain. Workstyle modeling complements task modeling [8] with supplemental information about how people communicate and coordinate their activities, and about the nature of the artifact to be produced. We have applied this model to the evaluation of how software designers collaborate, the forms of collaboration a wide variety of software design tools support, and to the design of the Software Design Board application itself. The development of the model itself was informed by the empirical study, presented in the previous section, as well as by informal laboratory studies of tools and designers.

In order to understand the relevance of workstyle analysis, consider the task of creating a design in some formal diagrammatic notation. A task model can identify the activities involved in creating such a design: drawing and labeling nodes, connecting them with relations, editing and reformatting diagram elements, and so forth. This model of design activities might lead to the development of a tool similar to Rational Rose [26] permitting mouse-based structural editing of design diagrams. However, in addition to the tasks that need to be performed, it is important to understand the users' preferred workstyle before committing to a design. Designers may be working in a brainstorming style, or may be recording precise documentation from which a system is to be built. A brainstorming workstyle is well supported by a whiteboard, which provides sufficient space for small groups to work, and supports a fluid style of interaction where multiple designers may interact with the design artifact in parallel. Alternatively, recording of precise documentation is well supported by a traditional Computer-Aided Software Engineering (CASE) tool. It is important to note that, though both tools support the activities identified in the task model, they do so in different ways that are appropriate to entirely different styles of work. The workstyle model helps in the analysis of peoples' goals and tasks by helping to understand their preferred style of work.

The *Workstyle* model characterizes a working style as an eight dimensional space that addresses the style of collaboration and communication between designers and the properties of the artifacts that are created during the collaboration. The functionality of collaborative design tools can be plotted in this space to specify the set of workstyles that they can support. It then becomes possible to compare designers' preferred workstyles to those supported by available tools and to identify potential task/tool mismatches. These mismatches can be used to guide the design of new tools that are more appropriate to particular design activities. Figure 1 depicts a graphical representation of the axes of Workstyle Model on which workstyle analyses are plotted

3.1 Dimensions Describing Collaboration Style

The first four dimensions of the model describe the nature of the collaboration in which a group is engaged, or that can be supported by a tool. They are defined as follows:

- *Location:* The location axis refers to the distribution of the people involved in the collaboration. As people become more geographically distributed, supporting some collaborative workstyles becomes increasingly difficult [27].
- *Synchronicity:* The synchronicity axis describes the temporal nature of the collaboration. People may work together at the same time (*synchronously*) or at different times (*asynchronously*).
- *Group Size:* The group size axis captures the number of people involved in the collaboration. Support for larger groups typically comes at the expense of intimacy in the interaction between collaborators.
- *Coordination:* This axis describes how users' activities are coordinated, whether by the choice of tools they are using or through the adoption of some coordination model [22].

3.2 Dimensions Describing Artifact Style

The remaining four dimensions describe the nature of the artifacts produced by the group, or able to be produced by a tool. They are defined as follows:

- *Syntactic Correctness*: The artifact being produced may be required to follow a precise syntax. This requirement may inhibit progress in early stages of design by forcing initially abstract designs to conform to a predetermined syntax [20, 35].
- *Semantic Correctness:* An artifact is considered to be semantically *sound* if its meaning is unambiguous and free of contradiction. The production of semantically sound artifacts facilitates automated analysis and evolution.
- *Archivability:* Archivability represents the difficulty of saving an artifact so that it can be used at a later time. For example, word processing documents have high archivability, as they can be saved to disk and retrieved later.
- *Modifiability:* This axis represents the ease with which an artifact can be modified. For example, small modifications to a whiteboard drawing are simply performed by erasing and redrawing.

3.3 Applying the Workstyle Model

The Workstyle Model can be applied to assess tools and/or the interaction style of users. The model can be used to evaluate the support provided by individual tools for various working styles, or applied to users to evaluate their working styles while accomplishing various tasks with preferred tools. To do so, values for each property are plotted on a two-dimensional representation of the model, as seen in Figure 1. A single workstyle is represented as a point in an eight dimensional space, while a range of workstyles is represented as a region in this space. Support for a single value in a particular property is indicated by a line intersecting the related axis, while a region over the axis represents support for a range of values in that property. So a plot that consists of a single line with no expanded areas can represent a tool or set of tools that supports a single, rigid workstyle. Similarly, if applied to users, the plot may represent a particular style of work used to accomplish some particular task. Conversely, a plot that covers an area of the graph may represent a tool or set of tools that supports a range of workstyles and transitions between them. Similarly, if applied to users, it may represent a change in the style of interaction that has occurred over a period of time. Once plotted, differences in the workstyles supported by various tools become visually apparent. These plots can be compared to workstyle plots of users accomplishing the tasks supported by those tools in their preferred manner. Mismatches between these plots identify tools that are not providing sufficient usability for their supported tasks. More detail and examples of applying the Workstyle Model can be found in [36, 38].

3.3.1 Workstyle Example – UML Design Tools

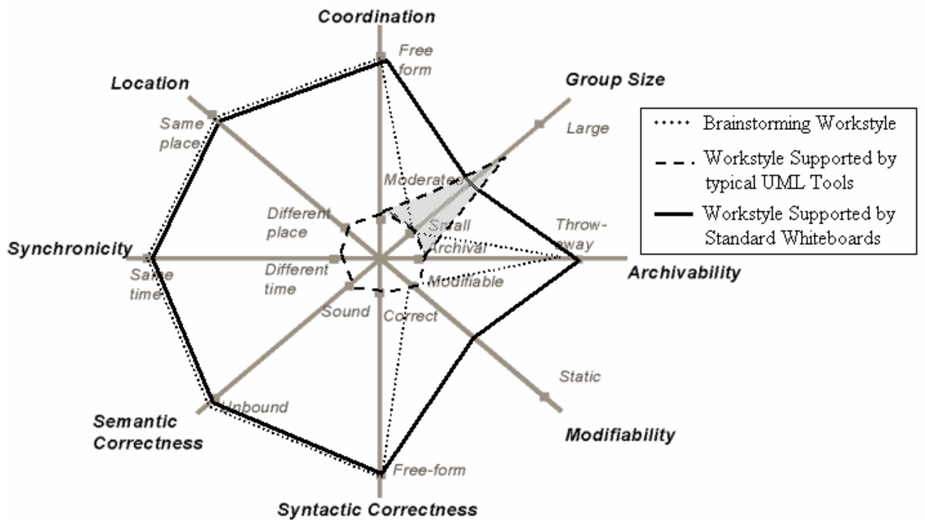


Fig. 1. A Workstyle comparison between UML tools and standard whiteboards in support for typical brainstorming activities.

It is useful to consider the workstyle supported by popular UML design tools such as Rational Rose [26]. Design tools such as these are a good fit with workstyles where little real-time communication with other designers is required, and where the goal is to create precise, archival designs. However, these design tools provide poor support for the early stages of design, such as brainstorming. During these phases, designers spend significant time on communications tasks.

The inappropriateness of UML design tools for early stages of design can be clearly shown by examining the brainstorming workstyle. As shown in Figure 1, brainstorming is typically carried out by small groups working face to face, using free-form coordination and social protocols to determine who gets to speak or write next. In brainstorming, designers do not wish to be distracted by requirements to be syntactically correct, or even semantically sound [2, 31]. Modifiability is important as early designs evolve rapidly, and archivability is important to allow early designs to be migrated to more formal designs.

Figure 1 clearly shows that while UML design tools may support the core tasks of the early stages of design, they do not support the workstyle of early design (brainstorming). The emphasis on asynchronous, moderated work with strong emphasis on syntactic correctness and semantic soundness is incompatible with the free-form brainstorming workstyle. A better match to the workstyle of early design is the workstyle supported by standard whiteboards. These tools support small groups of co-located users working synchronously, and rely upon social protocol to mediate user interaction. They impose no requirements on syntax, nor do they interpret any semantic meaning from the input.. The main incompatibility of these tools to the brainstorming workstyle is the limited ability to easily archive artifacts created on the board.

In this example, we have seen how workstyle analysis can be applied to a tool and compared to the workstyle of the collaborative activities in which it may be used. Such comparisons can highlight incompatibilities between a tool and the way in which it will be used within a particular context. Through this mechanism, tools can be selected for use in particular contexts to provide better usability to users carrying out their tasks.

4 Software Design Board: Supporting Workstyle Transitions in Software Design

Based on the findings from our empirical study into collaboration in software design, as well as workstyle analyses revealing inadequacies of existing design tools, we developed the Software Design Board to facilitate transitions between some common working styles as described by the Workstyle Model. This is achieved through the integration of informal media and flexible collaboration mechanisms, as well as support for migration between different software tools, devices and collaborative contexts. These facilities are intended to support fluid transitions between the some of the different styles of work in which designers are frequently engaged, specifically synchronous/asynchronous and/or co-located/distributed collaboration, and more generally, formal/informal interactions.

4.1 Functional Requirements

The functional requirements for the Software Design Board evolved from workstyle analyses of existing tools and of developers in the early stages of software design. For example, workstyle analyses of existing tools for collaborative software design revealed that each support only a single or limited set of collaborative workstyles. Furthermore, the empirical studies described in Section 3 revealed a variety of behavioral patterns in which developers frequently engage. Most importantly, the study found that team members regularly changed the nature of their interactions with each other in terms of synchronicity, location and modality. The results have specific implications on tool design; tools should be designed to support these frequent changes in workstyle.

All of these findings reveal some open problems in the area of tool support for collaborative software design, and motivated the functional requirements driving the design of Software Design Board. Specifically, the following are aspects of collaborative design that are poorly supported in existing tools:

- *Unsupported Workstyles:* Workstyle analyses of existing tools revealed that some workstyles are not supported by any individual class of tools. For example, large groups of synchronous collaborators, whether distributed or co-located, are not well supported by any available tool. This may be a result of hardware restrictions, or the limited applicability of such workstyles in practice. Additionally, no existing tools allow free-form interaction while supporting the creation of syntactically and semantically refined artifacts. Even informal CASE tools such as IdeogramicUML [15] employ a gesture-based syntax that places restrictions on free-form interaction.
    - **Functional Requirement 1**: Support the freehand creation of syntactically correct UML diagrams.
- *Lack of Support for Workstyle Evolution:* Workstyle analysis of existing tools revealed that individual tools only support a single or limited set of workstyles, and provide little or no support for movement between workstyles. However, our empirical investigations found that designers frequently move between synchronous/asynchronous and collocated/distributed styles of interaction. Additionally, transitions between workstyles often involve changes between interaction devices. For example, moving from an informal to a more formal workstyle may involve switching from an electronic whiteboard to a PC. Available tools do not sufficiently support migration between devices.
    - **Functional Requirement 2**: Support transitions between synchronous and asynchronous styles of collaboration.
    - **Functional Requirement 3:** Support transitions between collocated and distributed styles of collaboration.
    - **Functional Requirement 4**: Support transitions between physical devices.

- *Lack of Support for Multiple Collaborative Contexts:* In addition to frequently changing their collaborative workstyle, the results of the study presented in Section 3 show that individual designers also switch amongst a number of concurrent collaborative contexts. This means that they frequently move between multiple interactions with different groups. For example, a given designer may be participating in a number of concurrent projects or tasks, and may frequently switch their focus from one project to another. Furthermore, designers may participate concurrently in multiple collaborative contexts.
  - **Functional Requirement 5**: Support transitions between collaborative contexts.
- *Limited of Support for Integration of Existing Applications:* Current meta-tools that support sharing of existing applications, such as Netmeeting [23], impose significant restrictions on collaboration that can be inappropriate to many of the important workstyles found identified during the empirical study. Mechanisms for integrating existing tools into a variety of collaborative workstyles would allow designers to collaborate on wide variety of tasks without giving up their preferred tools for accomplishing those tasks.
  - **Functional Requirement 6**: Support integration of existing applications into all supported workstyles.

4.2 Overview of the Software Design Board

The Software Design Board (SDB) is a shared whiteboard application with additional functionality that supports collaborative software design. As seen in Figure 2, user interaction with this tool is similar to a typical interaction with a standard whiteboard.

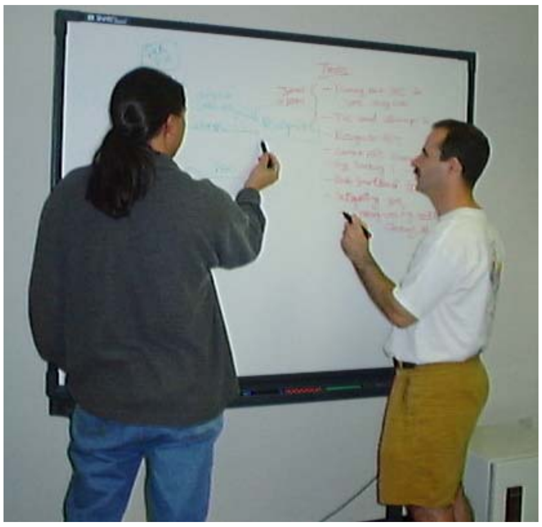


Fig. 2. Using Software Design Board.

Typical sessions using the tool via different devices are depicted in Figure 3. When used on a PC, the interface supports drawing using a typical structured drawing tool. Functionality is accessed through typical drop-down menus. When used on an electronic whiteboard or tablet PC, the user interface supports unstructured pen input of stroke information for freehand data such as diagrams, annotations, notes and lists.

This feature is in partial support of Functional Requirement 1 (*Support the freehand creation of syntactically correct UML diagrams*). Unstructured stylus-based input also provides the basis for lightweight user interaction with the tool. Furthermore, an integrated structure recognizer [9] supports automated translation of freehand diagrams into a more structured format appropriate for interpretation as UML or any other box-and-arrow notation. This functionality is similar to other tools [5, 21]. An example of this recognition functionality applied to a simple diagram is depicted in Figure 4.

In addition, objects can be placed on the board in and amongst the free hand data. These objects can include design documents or diagrams that may be browsed and annotated, or external programs that can execute other functionality. For example, a design document may be embedded into some area of the board allowing it to be communally browsed and annotated within the context of the other data on the board. This document is opened and displayed within the tool with which it was created, and all of that tool's functionality is accessible through the SDB's interface. This functionality supports Functional Requirement 6 (*Support integration of existing applications into all supported workstyles*). A typical session with an embedded design artifact is depicted in Figure 5.

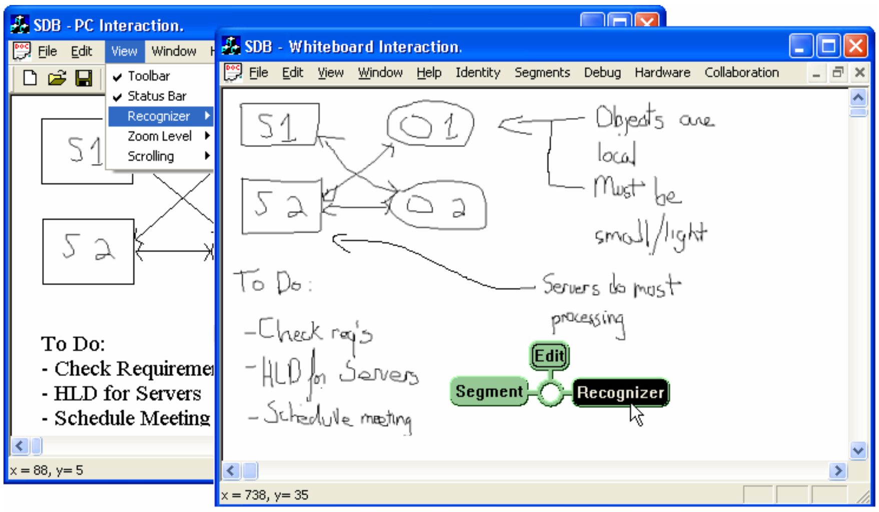


Fig. 3. Typical single-user sessions in Software Design Board. A PC user manipulates structured drawing elements and text, and interacts through drop-down menus. A whiteboard user draws free hand, and interacts through pie menus and gesture-based commands.

In order to support collaboration, the tool integrates communication and sharing mechanisms. For example, gesture transmission is supported within the context of synchronously shared whiteboard space. Voice communication mechanisms are planned, but not yet implemented. Additionally, any OLE-based communication tool can be integrated into the whiteboard space.

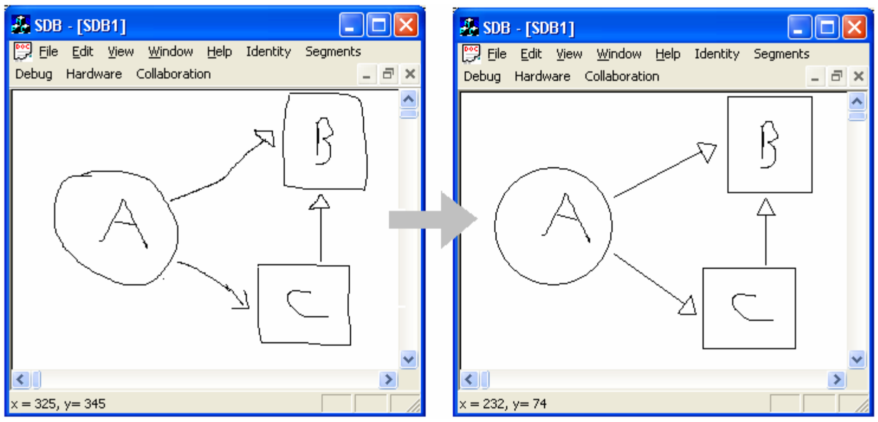

SDB - [SDB1]
File Edit View Window Help Identity Segments
Debug Hardware Collaboration
A
B
C
x = 325, y= 345
SDB - [SDB1]
File Edit View Window Help Identity Segments
Debug Hardware Collaboration
A
B
C
x = 232, y= 74

Fig. 4: Applying the syntax recognizer to a freehand diagram. Hand drawn elements such as circles, squares and arrows are recognized and converted into structured drawing elements.

These communication objects are embedded and manipulated directly within the context of the board, and are maintained with the rest of the data on the board. For example, external applications such as web browsers or media streams may be embedded in the board space and used for communication. These communication mechanisms support Functional Requirement 2 (*Support transitions between synchronous and asynchronous styles of collaboration*) and Functional Requirement 3 (*Support transitions between co-located and distributed styles of collaboration*) by allowing the simultaneous use of functionality supporting all of these styles of interaction within a single application.

The whiteboard space can be divided into any number of *segments*. These segments allow data to be shared in different ways. Generally, a segment is an area in the board containing contextually related data. As with a regular whiteboard, a user explicitly specifies the segmentation of data in the board through delineating strokes, e.g. a surrounding box or circle. Segments can be shared with others to allow users of other SDB clients to connect and synchronously interact with each other and share data. To share segments asynchronously, another client connects and copies the content of the segment to his/her local client. This data can then be manipulated without affecting the data in the original segment. Diverging copies of segments may be manually or automatically reconciled, if possible. When shared synchronously, data in a shared segment is viewed in decoupled WYSIWIS [29] fashion. Furthermore, at any time a user can change the way in which segments are shared. Synchronously shared segments can be easily detached and shared asynchronously, and vice versa. Gesture information is automatically transmitted between synchronously shared segments via telepointers. This functionality also supports Functional Requirement 2 (*Support transitions between synchronous and asynchronous styles of collaboration*) and Functional Requirement 3 (*Support transitions between co-located and distributed* *styles of collaboration*), by providing the mechanism by which users can freely and fluidly move between (synchronously or asynchronously) shared and private data.

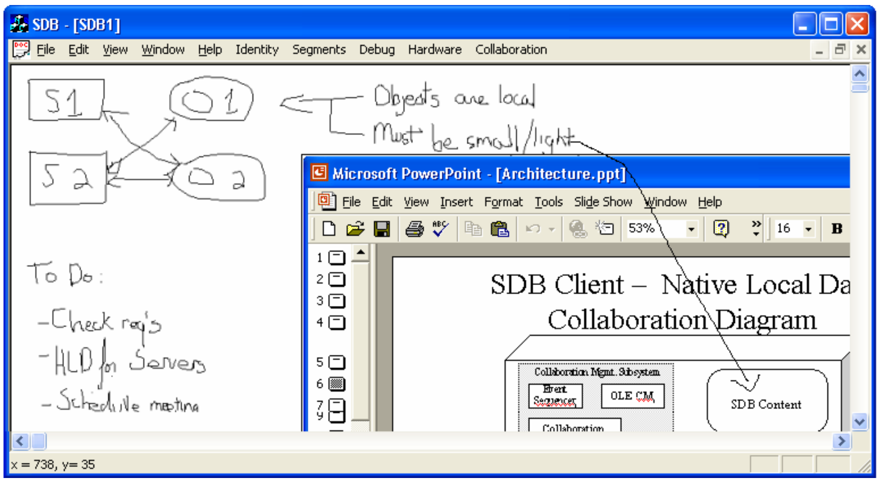

SDB - [SDB1]
File Edit View Window Help Identity Segments Debug Hardware Collaboration
S1
O1
S2
O2
Objects are local
Must be small/light
To Do:
-Check req's
-HLD for Servers
-Schedule meeting
Microsoft PowerPoint - [Architecture.ppt]
File Edit View Insert Format Tools Slide Show Window Help
SDB Client – Native Local Da
Collaboration Diagram
Collaboration Mgmt. Subsystem
Event Sequencer
OLE CM
SDB Content
Collaboration
x = 738, y= 35

Fig. 5. A design document embedded in a Software Design Board session.

Furthermore, on any SDB client, different segments may be shared concurrently and in different ways, between different groups. This functionality supports Functional Requirement 5 (*Support transitions between collaborative contexts*), by allowing users to move freely between different collaborative interactions contained within each segment. A typical session involving segment sharing is depicted in Figure 6.

Software Design Board implements a plastic interface [33] that can be used on different hardware devices. While the main platform for this application is an electronic whiteboard, it can also be accessed from a PC with or without an associated tablet. Widget-level plasticity supports appropriate interaction through each type of device [17]. For example, whiteboard users can use pie-menus and gesture based commands that are more appropriate to their stylus-based interfaces, while PC clients can use traditionally structured pull-down menus systems. There is also the potential to develop clients that facilitate access from a PDA or any other appropriate device. The interaction allowed by each interface is appropriate to the specific device. For example, interaction through a PDA would be greatly limited as compared to an interaction at a SmartBoard, and drawing facilities on a mouse-based PC client may be more structured than those on the SmartBoard, in order to accommodate the associated input mechanism. This functionality is in support of Functional Requirement 4 (*Support transitions between physical devices*).

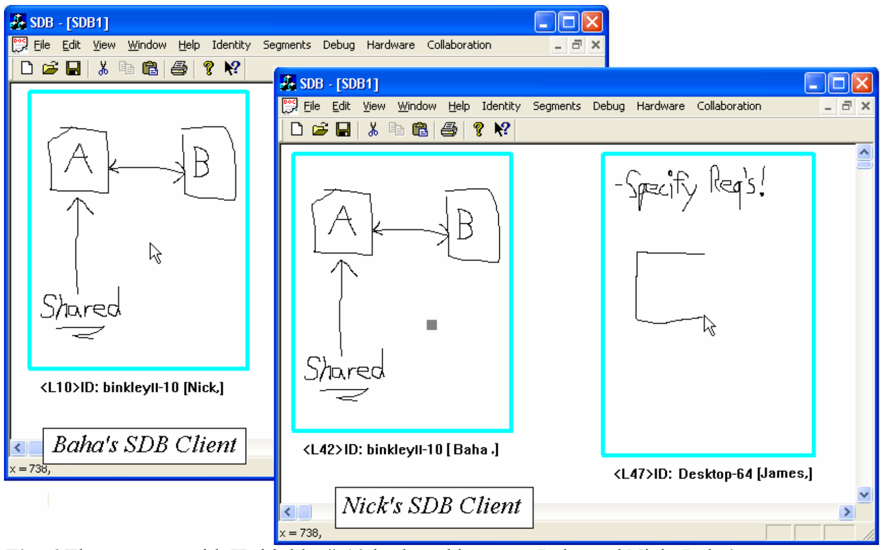

SDB - [SDB1]
File Edit View Window Help Identity Segments Debug Hardware Collaboration
A
B
Shared
-Specify Req's!
<L10>ID: binkleyII-10 [Nick.]
<L42>ID: binkleyII-10 [Baha .]
<L47>ID: Desktop-64 [James.]
Baha's SDB Client
Nick's SDB Client
x = 738,

Fig. 6.The segment with ID binkley||-10 is shared between Baha and Nick. Baha's mouse pointer appears as a telepointer on Nick's client. Nick is concurrently sharing a different segment, with ID Desktop-64, with James.

Device appropriate interfaces allow users to interact with the application through any available or preferred hardware, and freely migrate between device types, as long as the limitations of the hardware are accepted. Migration between tools and devices is further supported by the segmentation of data. Segmentation facilitates data plasticity, wherein types of data within a segment can be manipulated appropriately in the context of a given device or application. If a segment is known to contain data of a particular type, then it can be interpreted or formatted appropriately for any specific device or tool. For example, if a segment is known to contain a UML diagram, then it can be interpreted and migrated via XML into an appropriate UML-based CASE tool.

In addition to the functionality described above, a variety of additional features are integrated into the user interface to facilitate interaction with the Software Design Board. Unlike a regular whiteboard, a session in the SDB can be essentially unbounded in size. To facilitate navigation, the interface to the workspace is scrollable and zoomable. If a more structured input mechanism is desired at the whiteboard, a floating keyboard and/or structured drawing palette can be made available through menu options. These options can be accessed from context sensitive and device appropriate menu systems. Finally, all functionality is available through both context sensitive pull-down menus and pie-menus that facilitate gesture-based commands. This allows advanced users to use the tool more effectively by bypassing the menu structure.

4.3 Workstyle Transitions in Software Design Board

We now consider some simple scenarios that illustrate how Software Design Board can be used to perform some common transitions between workstyles. This is not intended as a set of instructions for performing the indicated transition, but rather as examples of how such transitions are supported within the tool. Additionally, it is intended to demonstrate the ease with these transitions can be performed within the tool.

- ***Distribution Transitions***: A group of co-located collaborators works together around an electronic whiteboard (a co-located workstyle). They want to share their work with a remotely located group. They draw a box around their current work in order to define a segment, and use a simple gesture command to share that segment with the remote group. The availability of the remote group is indicated via the context-sensitive pie menus [14, 19] that structure the gesture. At the remote site, a change in the entry structure of the menu system indicates the availability of a newly shared segment. The remote group creates a local segment in their workspace, and uses a similar gesture to attach their segment to that which was newly shared with them. Synchronized copies of the original data now appear in both group's segments, and telepointers appear to provide a sense of awareness of the actions of each group to the other. The two groups now collaborate in this distributed workstyle.
- ***Synchronicity Transitions:*** A group of users interacts synchronously with data contained in a shared segment (a synchronous workstyle). Each user performs updates that are immediately reflected in every other user's view of the data. They decide to work separately so that each user may concentrate on a particular aspect of the data. Each user detaches his/her segment from the shared session, and is left with a local copy of the data to which asynchronous updates can be performed. Now each user interacts with the data in their local copy (an asynchronous workstyle).
- ***Device Transitions***: A user is drawing a design on an electronic whiteboard. Using the piemenu structure and gesture commands, he invokes the recognizer and converts the freehand design to a structured drawing. He then creates a shared segment containing the diagram on the whiteboard. He moves to his PC and starts the Software Design Board client. Using the traditional pull-down menu structure, he creates a segment, attaches it to the shared segment he previously created at the whiteboard. He continues to work on that diagram from the PC, manipulating the structured elements in a manner appropriate for mouse-based interaction.
- ***Context Transitions***: A user maintains two different shared segments in his Software Design Board workspace. Each segment is shared between a different group of colleagues with whom he collaborates, and therefore each segment maintains completely different data (each maintains a different work context). Through the course of the day he scrolls the workspace back and forth between those segments in order to interact with the different groups as required.
- ***Syntax Transitions***: A group of co-located users are brainstorming and free hand drawing a design on a whiteboard. Eventually, the drawing becomes too large and convoluted to easily manipulate in this manner. Some elements consume a disproportionate amount of board space; others overlap due to the freeform

development of the diagram. The designers want to move the work into a structured drawing editor to clean up the drawing and continue work. They use a gesture command to select all relevant drawing elements, then another gesture to invoke the syntax recognizer. The drawing is automatically converted to discrete, structured drawing elements such as boxes, circles and arrows. A third gesture is used to invoke a 'Send To…' command, which causes the newly structured elements to be opened within a structured drawing editor. The group now restructures their drawing, and continues to work.

- ***Semantic Transitions***: A group of users has completed a freehand design diagram on a whiteboard. The users invoke the syntax recognizer to structure their drawing, as described above. Next, they use a gesture command to reselect all drawing elements, and another gesture to invoke the UML semantic interpreter. The structured drawing is automatically interpreted as a simplified UML class diagram– boxes are converted to classes, open arrows as generalizations, closed arrows as aggregations. A third gesture is used to invoke a 'Send To…' command, which causes the newly structured elements to be opened within a UML editor for further manipulation.

4.4 Current Status of the Implementation

The Software Design Board application is currently a functional research prototype. Most of the functionality described in the previous sections exists, either wholly or partially, though some core functionality remains to be implemented. Functionality for moving, resizing and copying freehand elements still remains to be implemented, and structured drawing functionality and other PC-based interaction techniques are less developed. Distributed, synchronous sharing is currently limited to drawing data; synchronous application sharing functionality is only partially implemented and not yet functional. The functionality for implementing syntax transitions is not fully implemented. An XML DTD has been developed to describe these recognized free-hand diagrams, and standalone code for writing and reading these XML documents exists. However, this code has not yet been integrated with the Software Design Board application. Finally, only limited work has been done toward supporting semantic transitions, i.e. applying a semantic interpretation to the syntactic structure of the drawing described by the XML document. This work has been limited by the limited implementation supporting syntax transitions. As the functionality evolves to more completely support the syntax transition, so too will the functionality supporting the semantic transition.

5 Conclusions

In this paper, we have introduced a prototype collaborative software design tool, the Software Design Board. Software Design Board supports a variety of workstyles important in the early stages of software development, and facilitates transitions between them. The functional requirements for the tool evolved from workstyle analysis of existing design tools and from results of empirical research into collaborative software design activities.

The need to support workstyle transitions in tools for collaborative software design stems from the fact that designers switch amongst numerous collaborative styles throughout the course of the their work. Many factors influence the style in which they may choose to work (their *workstyle*), including the task at hand, availability of tools, distribution of collaborators, and personal preferences. These influences change frequently, thus designers often migrate between workstyles in response to such changes. Unfortunately, there are obstacles to such transitions. These may include having to recreate work artifacts in the format of a new tool, interruption of the flow of work, or physical relocation. Such obstacles may prove sufficiently burdensome that designers choose to continue to work in a style that is inappropriate for their current context. These obstacles exist because the variety of workstyles and workstyle transitions in which designers engage are not well supported by most existing design tools. Most of these tools are designed to support a single or limited set of workstyles, and their architectures are generally not capable of handling the dynamic changes in workstyle that are typical of collaborative design.

Software Design Board was developed to address some of these shortcomings and to support designers in some of the common workstyles and transitions in workstyle in which they frequently engage. Specifically, Software Design Board supports designers working synchronously/asynchronously, distributed/collocated and more generally, formally/informally. It supports the creation of syntactically bound or free-from artifacts, can be used through a variety of physical devices, and facilitates collaboration in multiple, concurrent contexts.

References

1. AgileAlliance, http://www.agilealliance.org
2. Bly, S., A. (1988). "A Use of Drawing Surfaces in Different Collaborative Settings". Conference on Computer-Supported Cooperative Work, Portland, OR.
3. Bly, S.,A. and S. Minneman (1990). "Commune: A Shared Drawing Surface." SIGOIS Bulletin: 184-192.
4. Crowley, J., Coutaz, J., Berard, F. (2000). "Things that See." Communications of the ACM **43**(3): 54-64.
5. Damm, C. H., Hansen, K. M., Thomsen, M. (2000). "Tool Support for Object-Oriented Cooperative Design: Gesture-Based Modelling on an Electronic Whiteboard". Proceedings of Conference on Human Factors and Computing Systems. The Hague, Netherlands.
6. DeMarco, T. and T. Lister (1987). Peopleware. New York, Dorset House.
7. Dewan, P. Choudary, R. (1991). "Flexible user interface coupling in collaborative systems". CHI ' 91, New Orleans, LA, ACM.
8. Diaper, D. (1989) Task analysis for human computer interaction, Ellis Horwood,.
9. Fonseca, M.,J., Pimentel, C., and Jorge, J., A. (2002). "CALI: An Online Scribble Recognizer for Calligraphic Interfaces**"**, Proceedings of the 2002 AAAI Spring Symposium - Sketch Understanding. Palo Alto, USA. pp51-58
10. Francik, E., Rudman, S. E., Cooper, D., and Levine, S. (1991). Putting innovation to work: adoption strategies for multimedia communication systems. *Communications of the ACM*, 34(12), pp. 52-64.

11. Greenberg, S. and R. Bohnet (1991). "GroupSketch: A Multi-user Sketchpad for Geographically Distributed Small Groups". Proceedings of Graphics Interface, pp 207-215.
12. Grundy, J. C., Mugridge, W.B, Hosking, J.G., Apperley, M. (1998). "Tool Integration, Collaboration and User Interaction Issues in Component-based Software Architectures". TOOLS '98, Melbourne, Australia, IEEE.
13. Hammond, T. and R. C. Davis (2002). "Tahuiti: A Geometrical Sketch Recognition System for UML Class Diagrams". Sketch Symposium, Stanford University, Palo Alto, CA.
14. Hopkins, D. (1991) "The Design and Implementation of Pie Menus", Dr. Dobb's Journal, CMP Media. December 1991.
15. Ideogramic – IdeogramicUML, http://www.ideogramic.com
16. Ishii, H. and M. Kobayashi (1992). "ClearBoard: A seamless medium for shared drawing and conversation with eye contact". Conference on Human Factors in Computing Systems, Monterey, CA, ACM.
17. Jabarin, B., and Graham, T.C.N. (2003) "Architectures for Widget-Level Plasticity", Proceedings of DSV-IS 2003 Portugal, June 11-13. pp. 124-238
18. Jones, T. C. (1986). Programming Productivity. New York, McGraw-Hill.
19. Kurtenbach, G. and Buxton, W. (1991) "Issues in Combining Marking and Direct Manipulation Techniques" In Proceedings of ACM UIST'91. pp. 137--144.
20. Landay, J. A. and B. A. Myers (1995). "Interactive Sketching for Early Stages of Design". CHI '95, Denver, CO, ACM Press.
21. Lank, E., Thorley, J.S., Chen, S.J. (2000). "An Interactive System for Recognizing Hand Drawn UML Diagrams". CASCON2000, Toronto, ON.
22. Malone, T. W. and K. Crowston (1990). "What is coordination theory and how can it help design cooperative work systems?". Proceedings of Conference on Computer-Supported Cooperative Work. ACM Press. pp. 357-370
23. Microsoft Corp. – Netmeeting, http://www.microsoft.com
24. Mynatt, E. D., Igarashi, T., Edwards, W.K. LaMarca, A. (1999). "Flatland : New Dimensions in Office Whiteboards". CHI '99, Pittsburgh, PA, ACM.
25. Pederson, E. R., McCall, K., Moran, T.P., Halasz, F. G. (1993). "Tivoli: An Electronic Whiteboard for Informal Workgroup Meetings". INTERCHI '93. Amsterdam, Netherlands. April.
26. Rational Corp. – Rose, http://www.rational.com
27. Seaman, C.B. and Basili, V.R. (1997) "Communication and Organization in Software Development: An Empirical Study". IBM Systems Journal 36(4).
28. SMART Technologies, Inc. – SMARTBoard, http://www.smarttech.com
29. Stefik, M., Bobrow, D.G., Foster, G., Lanning, S., and Tatar, D. (1987) "WYSIWIS revised: early experiences with multiuser interfaces", ACM Transactions on Office Information Systems, 5(2), pp.147-167
30. Streitz, N. A., J. Geißler, Haake, J. M., Hol, J. (1994). "DOLPHIN: integrated meeting support across local and remote desktop environments and LiveBoards". Conference on Computer Supported Cooperative Work, Chapel Hill. NC.
31. Tang, J., C. (1991). "Findings from Observational Studies of Collaborative Work." International Journal of Man-Machine Studies. 34(2), pp. 143-160
32. Tang, J. C. and S. Minneman (1991). "VideoWhiteboard: Video Shadows to Support Remote Collaboration". Conference on Human Factors and Computing Systems, New Orleans, LA.
33. Thevenin, D., and Coutaz, J., (1999). "Plasticity of User Interfaces: Framework and Research Agenda" Proceedings of Interact '99 Edinburgh, Scotland. pp 110-117.

34. Wang, W., Dorohonceanu, B., Marsic, I. (1999). "Design of the DISCIPLE Synchronous Collaboration Framework". Internet, Multimedia Systems and Applications, Nassau, Bahamas, IASTED Press.
35. Wong, Y.Y. (1992) "Rough and ready prototypes: Lessons from graphic design". Short Talks Proceedings of CHI '92: Human Factors in Computing Systems, pp. 83-84, Monterey, CA,
36. Wu, J., Graham, T.C.N, Everitt, K., Blostein, D. and Lank, E. (2002) "Modeling Style of Work as an Aid to the Design and Evaluation of Interactive Systems". Proceedings of CADUI'02. Valenciennes, France.
37. Wu, J., Graham, T.C.N., Smith, P. (2003) "A Study of Collaboration in Software Design" ISESE 2003, Rome, IT. Sept 29-Oct 1.
38. Wu, J. (2003) "Tools for Collaborative Software Design" Queen's University, School of Computing. Technical Report 2003-462, Queen's University, Kingston, Ontario, Canada, January 2003.

Discussion

[Philippe Palanque] As you use the work style axes as a mean for evaluating the adequacy between tool and a work style do you not need more detailed information for each axes?

> [Nick Graham] All the axes are continuous and we use them more as an informational tool - we worked on making the axes more precise but we did not find it to be more useful.

[Jürgen Ziegler?] Are the dimensions independent or are there interrelationships between eg. modifiability and degree of semantic correctness?

> [Nick Graham] I think we can come up with examples for each pair of axes where you could be at either extreme and if you think of each pair of axes that the extremes are presented as cross products of all four possible positions, then we can come up with examples of all four positions for all the axis pairs, so we are quite confident that axes are orthogonal.

[Grigori Evreinov] Did you think of using parallel coordinate systems?

> [Nick Graham] No, that would be interesting; do you think that would be better?

[Grigori Evreinov] Yes, we have Information Visualization Research Group in our Department (http://www.cs.uta.fi/~hs/iv/) and the parallel coordinates system is presented on their site, so you can try it! or ask about the author Harry Siirtola

> [Nick Graham] That would be interesting!

[Jörg Roth] Your work style model reminds me of the Denver model from 1996 (they have 2 diagrams with 5 axes each instead of your 8)?

> [Nick Graham] There are similar in the sense that they are both related to groupware and presented as "quiviant diagrams". Beyond that the axes are actually very different to my recollection! I have compared to the Denver

model, but to give you a proper answer I would have to look at the Denver model again, because I cannot remember the axes exactly!

[Michael Harrison] One of the interesting things about collaborative work is that, just like we have had this conference I will go away to a room and do some work and maybe have some ideas and produce some notes. Next time we have a collaborative meeting I may want to fold that back in to the collaboration and I was not sure how that kind of continuity could be achieved. This characterises different collaborative models whereas that is not essentially a collaboration model, but it is essential to the process of collaboration.

[Nick Graham] That would be considered a tool transition, so one tool is pen and paper and the other your designed word software. We are very interested in that, so one approach is to say it would be wonderful if you had electronic paper that you had been scrip ling on and that could be imported right in to the tool, a poor mans approach to that would be to scan it, a really poor mans approach would be to sit and type it in. So those are examples of how transitions can be easy or hard. The whole goal is certainly to find ways of making the transition easier so that people are more likely to do them.

[Hong-Mei Chen, University of Hawaii] The Work style model you presented here seems to be domain-specific to software design in your empirical case studies and not applicable to other kind of collaborative work. For instance, some brain storming tasks (as studied in Group Decision Support Systems - GDSS) consider important factors such as social cues and anonymity to be important.

[Nick Graham] I agree with you that there are many other axes that we could put in and we have actually studied it in IFIP WG 2.7/13.4 and discussed the kind of transitions that would come up, e.g. with respect to privacy. An example could be a situation where you start out in a context where privacy is not important to you and the all of a sudden you are asked to enter your credit card information and privacy becomes very important to you. This just to say, that these are also important issues and we do not claim to have solved every issue in the world. We have used this model in other domain, but will not make any claims that this is applicable to any domain and maybe we will come back next year with the 40 dimensions version!

[Rick Kazman] How do you deal with multiple updates to a single document when people work asynchronously but they want to merge their work?

[Nick Graham] We do not support merging in general since it is a difficult problem, but we do support merging of the whiteboard freehand drawings. Merging MS Word documents alone is big problem in it self!

[Rick Kazman] Are you aware of any general solution to the multiple merge problems?

[Nick Graham] No, all the solutions I have seen are point solutions often commercial, such as for MS Word, but no good general solutions.

Supporting Group Awareness in Distributed Software Development

Carl Gutwin, Kevin Schneider, David Paquette, and Reagan Penner

Department of Computer Science, University of Saskatchewan
Computer Science Department, University of Saskatchewan
57 Campus Drive, Saskatoon, SK
Canada, S7N 5A9
gutwin,kas,dnp972,rpenner @usask.ca

Abstract. Collaborative software development presents a variety of coordination and communication problems, particularly when teams are geographically distributed. One reason for these problems is the difficulty of staying aware of others – keeping track of information about who is working on the project, who is active, and what tasks people have been working on. Current software development environments do not show much information about people, and developers often must use text-based tools to determine what is happening in the group. We have built a system that assists distributed developers in maintaining awareness of others. ProjectWatcher observes fine-grained user edits and presents that information visually on a representation of a project's artifacts. The system displays general awareness information and also provides a resource for more detailed questions about others' activities.

1. Introduction

Software projects are most often carried out in a collaborative fashion. The complexities of software and the interdependencies between modules mean that these projects present collaborators with several coordination and communication problems. When development teams are geographically distributed, these problems often become much more serious [2,10,11,14]. Even though projects are often organized to try and make modules independent of one another, dependencies cannot be totally removed [14]. As a result, situations can arise where team members duplicate work, overwrite changes, make incorrect assumptions about another person's intentions, or write code that adversely affects another part of the project [10].

These problems occur because of a lack of awareness about what is happening in other parts of the project. Most development tools and environments do not make it easy to maintain awareness of others' activities [10]. Current tools are focused around the artifacts of collaboration rather than people's activities (e.g., the files in a repository rather than the actions people have taken with them). An artifact-based approach is clearly necessary for certain types of work, but without better information about people, smooth collaboration becomes difficult. Awareness is a design concept that holds promise for significantly improving the usability of collaborative software development tools.

We have built a system called ProjectWatcher that provides people with awareness information about others on the development team. The system is designed around our observations of the awareness requirements in several distributed software projects. We found that developers first maintain a general awareness of who is who and who is doing what on a project; and second, they actively look for information about people when they are going to work more closely with them. However, developers often have to use text-based sources to get that information.

ProjectWatcher observes and records fine-grained information about user edits and provides visualizations of who is active on a project, what artifacts they have been working on, and where in the project they have been working. This information about others' activities can help to improve coordination between developers and reduce some of the problems seen in distributed development.

In this paper, we introduce ProjectWatcher and describe its design and implementation. We first give an overview of the issues affecting collaboration in software development, and then discuss group awareness in more detail and the awareness requirements of a distributed development project. We then describe the two main parts of ProjectWatcher: a fact mining component that gathers developer activity information, and a visualization component that overlays activity data onto a representation of project artifacts.

2. Background

Although collaboration is an important research area of software engineering – where teams are common and where good communication and coordination are essential for success – little work has been done on group awareness in software development. Similarly, although awareness has received attention in the Computer-Supported Cooperative Work (CSCW) community, this knowledge has not been considered extensively in development settings. We believe that awareness is a design concept that holds promise for significantly improving the usability of collaborative software development tools. In the next sections, we review issues of collaboration in distributed software development, the basics of group awareness, and the awareness requirements that we have determined from observations of open source projects.

2.1 Collaboration Issues in Software Development

Collaboration support has always been a part of distributed development – teams have long used version control, email, chat groups, code reviews, and internal documentation to coordinate activities and distribute information – but these solutions generally either represent the project at a very coarse granularity (e.g., CVS), require considerable time and effort (e.g., reading documentation), or depend on people's current availability (e.g., IRC).

Researchers in software engineering and CSCW have found a number of problems that still occur in group projects and distributed software development. They found that it is difficult to:

- determine when two people are making changes to the same artifacts [14];
- communicate with others across timezones and work schedules [11];
- find partners for closer collaboration or assistance on particular issues [20];
- determine who has expertise or knowledge about the different parts of the project [24];
- benefit from the opportunistic and unplanned contact that occurs when developers are co-located, since there is little visibility of others' activities [10].

As Herbsleb and Grinter [10] state, lack of awareness – "the inability to share the same environment and to see what is happening at the other site" (p. 67) is one of the major factors in these problems.

2.2 Group Awareness

In many group work situations, awareness of others provides information that is critical for smooth and effective collaboration. Group awareness is the understanding of who is working with you, what they are doing, and how your own actions interact with theirs [5]. Group awareness is useful for coordinating actions, managing coupling, discussing tasks, anticipating others' actions, and finding help [8]. The complexity and interdependency of software systems suggests that group awareness should be necessary for collaborative software development. Knowledge of developer activities, both past and present, has obvious value for project management, but developers also use this information for many other purposes – purposes that assist the overall cohesion and effectiveness of the team. For example, knowing the specific files and objects that another person has been working on can give a good indication of their higher-level tasks and intentions; knowing who has worked most often or most recently on a particular piece of code indicates who to talk to before starting further changes; and knowing who is currently active can provide opportunities for real-time assistance and collaboration.

In co-located situations, three mechanisms help people to maintain awareness: *explicit communication*, where people tell each other about their activities; *consequential communication* [22], in which watching another person work provides information as to their activities and plans; and *feedthrough* [4], where observation of changes to project artifacts indicates who has been doing what. Of these mechanisms, explicit communication is the most flexible, and previous research has looked at the ways that groups communicate over distance, through email, text chat, and instant messaging (e.g., [18,23]). However, since intentional communication of awareness information also requires the most additional effort, many awareness systems attempt to support implicit mechanisms as well as communication. General approaches include providing visible embodiments of participants and visual representations of actions that allow people to watch each other work, and overview visualizations of artifacts that show feedthrough information.

Although group awareness is often taken for granted in face-to-face work, it is difficult to maintain in distributed settings. This is particularly true in software development: other than access to the shared code repository, development environments and tools provide almost no information about people on the project. Although communication tools such as email lists and chat systems help to keep people informed on some projects, these text-based awareness mechanisms require considerable effort, and are not well integrated with information about the artifacts of the project. As a result, coordination problems are common in distributed settings, and collaboration suffers. A few research systems do show awareness information (e.g., TUKAN [21] or Augur [7]), but it is not clear that these tools really provide the awareness information that is needed by developers. As discussed in the next section, we based our tools and techniques on findings from a study of three distributed open-source projects.

3. Awareness Requirements in Distributed Development

Open-source software development projects are a good source of information about distributed development, since they are almost always collaborative and widely dispersed (in many cases, developers never meet face-to-face). To find out what the awareness requirements are for these long-running real-world projects, we interviewed several developers, read project communication, and looked at project artifacts from three open source projects [9]. We found that distributed developers do need to maintain awareness of one another, and that they maintain both a general awareness of the entire team and more detailed knowledge of people that they plan to work with. However, developers maintain their awareness primarily through text-based communication – particularly mailing lists and chat systems.

The three open source projects we looked at are NetBSD (www.netbsd.org), Apache httpd (www.apache.org), and Subversion (www.tigris.org/subversion). We chose these projects because they are distributed, they are at least medium-sized in terms of both the code and the development team, and they all produce a product that is widely used, indicating that they have successfully managed to coordinate development.

An initial issue that we looked at was whether distributed projects can successfully isolate different software modules from one another such that awareness and coordination requirements become insignificant. There are two ways that dependencies can be reduced – by reducing the number of developers, or by partitioning the code. However, in the three projects we looked at, neither of these factors removed awareness requirements. There were at least fourteen core developers who contributed regularly to each project, and although there was general understanding that people work in ‘home’ areas, there were no official sanctions that prevented any developer from contributing to any part of the code. On Apache and Subversion in particular, development of a particular module was almost always spread across several developers.

The next issue studied was what types of awareness the developers maintained. We found two types: general awareness and more specific knowledge. First, developers maintain a broad awareness of who are the main people working on their project, and what their areas of expertise are. This information came from three sources: the project mailing list, where people can see who posts and what the topics of discussion are; the chat server, which provides similar information but in real time; and the CVS commits (sent out by email), which allowed developers to stay up-to-date both on changes to the project and the activities of different people. Second, when a developer wishes to do work in a particular area, they must gain more detailed knowledge about who are the people with experience in that part of the code. We found that people use a variety of sources to gather this information, including project documentation, the records in the source code repository, bug tracking systems, and other people. Further details on this study can be found in [9].

Even though these open-source projects do successfully manage their coordination, our interviews also identified some problems with the way awareness is maintained. Two problems that we consider further in this paper involve watching CVS commits, and maintaining overall awareness about project members and their activities. Although the 'CVS-commit' mailing list provides the only information that is actually based on the project artifacts, several developers said that they do not follow them because they are too time-consuming to read. Developers also suggested that some of the information sources they use often go out of date, and that understanding the relationships between people and activities was often difficult. One developer stated that new members of the project in particular could benefit from tools that provided more information than what was currently available.

4. Project Watcher

We have developed an awareness system called ProjectWatcher to address some of the awareness issues that we have seen in distributed development projects. ProjectWatcher gathers information about project artifacts and developer's actions with those artifacts, and visualizes this awareness information either as a stand-alone tool or as a plugin inside the Eclipse IDE. ProjectWatcher consists of two main parts – the mining component, and the awareness visualizations.

4.1 Mining Component

The mining component analyzes a project's source code to produce facts for use by the ProjectWatcher visualization displays. To gather developer activity information at a finer grain size than repository commits, a shadow CVS repository is maintained (see Figure 1). User edits are auto-committed to the shadow repository as developers edit source code files (e.g., on every save of the file). With each auto-commit a new version of the file is stored in the shadow repository. The mining component analyzes the auto-committed versions against each other and the versions in the shared CVS repository to obtain user edit information that can be understood in terms of the project's software architecture.

The mining component is composed of two fact extractors: the software architecture fact extractor and the user edit fact extractor. The software architecture fact extractor is run against the software repository to obtain entity/relationship facts.

Entity facts extracted include: *package*, *class* and *method* facts. Relationship facts extracted include: *calls*, *contains*, *imports*, *implements* and *extends* relationships. The software architecture facts are used by the visualization system to present the software structure. The user edit fact extractor is run against the shadow repository to obtain information about the methods a developer is changing. The user edit facts are used by the visualization to present developer activity information.

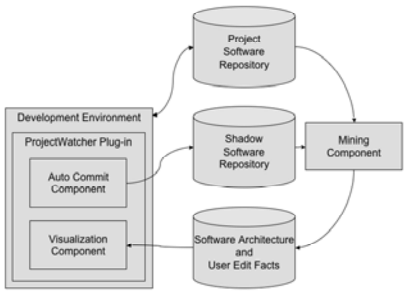


Fig. 1: User edit fact extraction.

The software architecture fact extractor is implemented in two stages and may either be run on the shadow repository or on the shared software repository (see Figure 2). The first stage, the *base fact extractor* uniquely names the entities in the source code and extracts the facts of interest. This process is accomplished with a TXL [15] program using syntactic pattern matching [3]. The second stage, the *reference analyzer*, resolves references between software architecture entities.

The reference analyzer extracts scope facts from the project source code and integrates them with the facts extracted in stage one. Next, the method call facts are analyzed to determine which package and class the method that was called belongs to. This process involves resolving the types of variables and return types of methods that are passed as arguments to method calls. The types of all the arguments are identified. Then scope, package, class, and method facts are analyzed to determine which package and class the method belongs to. To resolve calls to the Java library, the full Java API is first processed by the ProjectWatcher mining component (this is only done once for all projects).

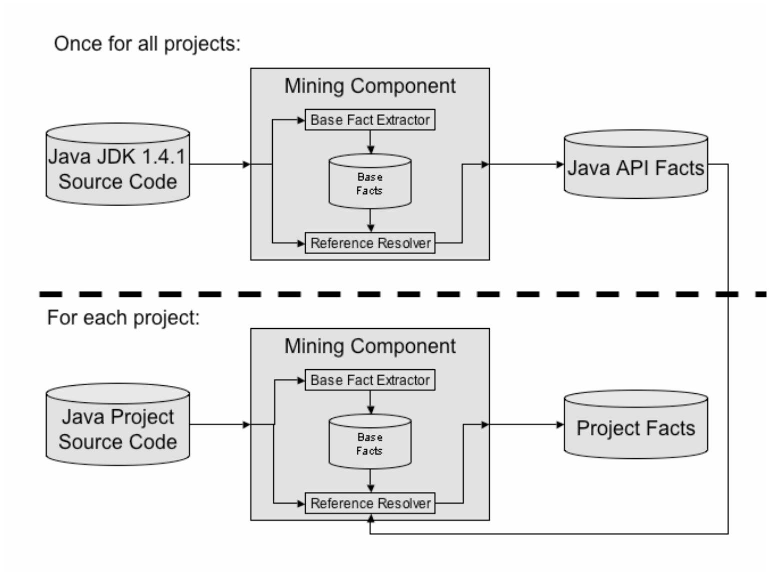


Fig. 2: Software architecture fact extraction from Java projects

The user edit fact extractor (Figure 3) is implemented in three stages and is run against two versions of the project source code. The first stage splits the files into separate class and method snippets. The second stage compares and matches revisions of the code snippets. Initially, methods are matched based on their names. If a method match is not found at the method name level, methods are compared based on the percentage of lines of code that match between all methods. If a method's name is changed, a match based on percentage of similarity is still found between the two versions. When no match is found for a method from an earlier revision, the method is identified as having been added. When no match is found for a method from a later revision, the method is identified as having been removed. Facts about method additions and method removals are stored in the user edit factbase. Once the methods from each revision have been matched, a line diff is performed on each pair of methods. The diff algorithm gives us information about what lines have been added and removed from a method, and this information is stored in the user edit factbase.

The complete factbase contains uniquely identified facts indicating all packages, classes, methods, variables, and relationships for a Java project and all user edits. These facts are used by the visualization component to show activity and proximity information. The time and space needed for fact extraction and factbase storage depends on the size of the code; for example, the Java Development Kit 1.4.1 contains 202 package facts, 5,530 class facts, 47,962 method facts, and 106,926 method call facts.

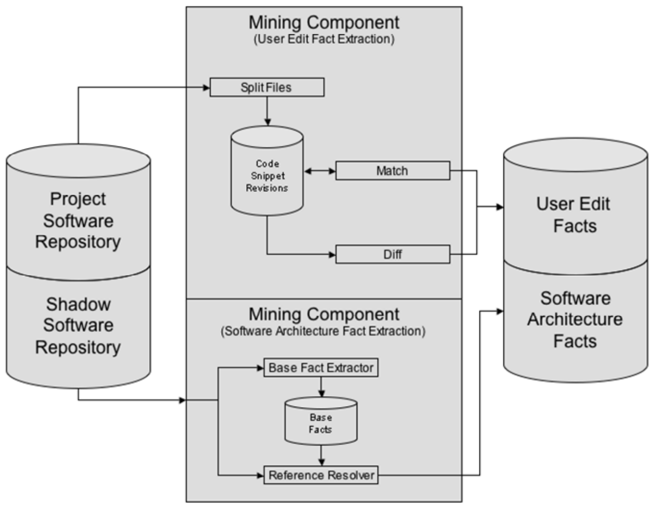

Mining Component
(User Edit Fact Extraction)
Split Files
Code Snippet Revisions
Match
Diff
Project Software Repository
Shadow Software Repository
User Edit Facts
Software Architecture Facts
Mining Component
(Software Architecture Fact Extraction)
Base Fact Extractor
Base Facts
Reference Resolver

Fig. 3: User edit fact extraction.

4.2 Visualization of activity and commits

ProjectWatcher's activity awareness display visualizes team members' past and current activities on project artifacts (see Figures 4 and 5). The goals of this display are:

- to give collaborators an overview of who works on the project
- to provide a general sense of who works in what areas
- to allow changes (i.e., commits) to be tracked without much effort
- to provide more detail when the user wants to look more closely.

The display uses the ideas of edit wear, interaction histories, and overviews. *Edit wear* is a concept introduced by Hill and colleagues [13]. Their overall motivation is the question of how computation can be used to improve "the reflective conversation with work materials" (p. 3), and the observation that most computational artifacts do not show any traces of the ways that they have been used, unlike objects in the real world. Starting with this idea of 'object wear,' their research proposes an 'informational physics' in which the visual appearance of an object arises not from everyday physical laws, but from informational rules that are semantically useful. Their notion of physics has objects explicitly show different aspects of their use over time – that is, their interaction history:

> The basic idea is to maintain and exploit object-centered interaction histories: record on computational objects…the events that comprise their use…and display useful graphical abstractions of the accrued histories as part of the objects themselves." ([13], p. 3)

Hill and colleagues were primarily interested in an individual's reflection on their use of work artifacts, but there is obvious value for group awareness as well. In ProjectWatcher, the artifacts are the files in a CVS repository (shadow or regular), and the interaction history is a record of all of the actions that a person undertakes with them (gathered unobtrusively by the fact extractor as people carry out their normal tasks).

We take these interaction histories and visualize them on an overview representation of the entire project. Overviews provide a compact display of all the project artifacts, and allow information to be gathered at a glance. In addition, the overview representation can be overlaid with visual information about the interaction history or about changes to the artifacts. Although some tools such as CVS front-ends do limited visualization of the source tree (e.g., by colour), our goal here is to collect much more information about interaction, and provide richer visualizations that will allow team members to quickly gather awareness information.

ProjectWatcher uses the extracted fact base to create a visual model of what each developer is doing in the project space. Project artifacts are shown in a simple stacked fashion that displays packages, files, classes, and methods. We chose this method of organization because it is much more compact than other approaches, such as class diagrams or dependency graphs. With the stacked representation, even a small overview can completely display projects with up to several hundred files (e.g., Figure 4 shows 322 files); in larger projects, developers can collapse particular packages to save space. The drawback with the stack is that there is little contextual information available to help users determine which artifact is which. To try and reduce this problem, artifacts are always stacked by creation date, so that their location in the overview is fixed, and can over time be learned by the user. We are also experimenting with allowing users to reorganize the display, so that they can arrange and group the artifacts in ways that are more meaningful to them.

On this basic overview representation, we overlay awareness and change information. First, each developer is assigned a unique colour, and this colour can be added to the blocks in the overview based on a set of filters. Common filters that involve developer information include who has modified artifacts most recently, and who has modified them most often. Other filters exist as well, such as one that shows time since last change (see Figure 5). Second, we show a summary of the activity history for each artifact with a small bar graph drawn inside the object's rectangle; bars represent amount of change to the class since its creation. More information about an artifact can be obtained by holding the cursor over a rectangle: for example, the name of the class and a more detailed bar graph.

Change information can be shown in addition to information about developers. The system highlight artifacts (using coloured borders) if they have changed recently – this provides users with dynamic information about commits to the project. When a change occurs to the CVS repository, the changed files are highlighted in the overview representation. More details about the change can be seen using the popup detail window, and further information (such as the difference between the two versions) can be seen through a context menu.

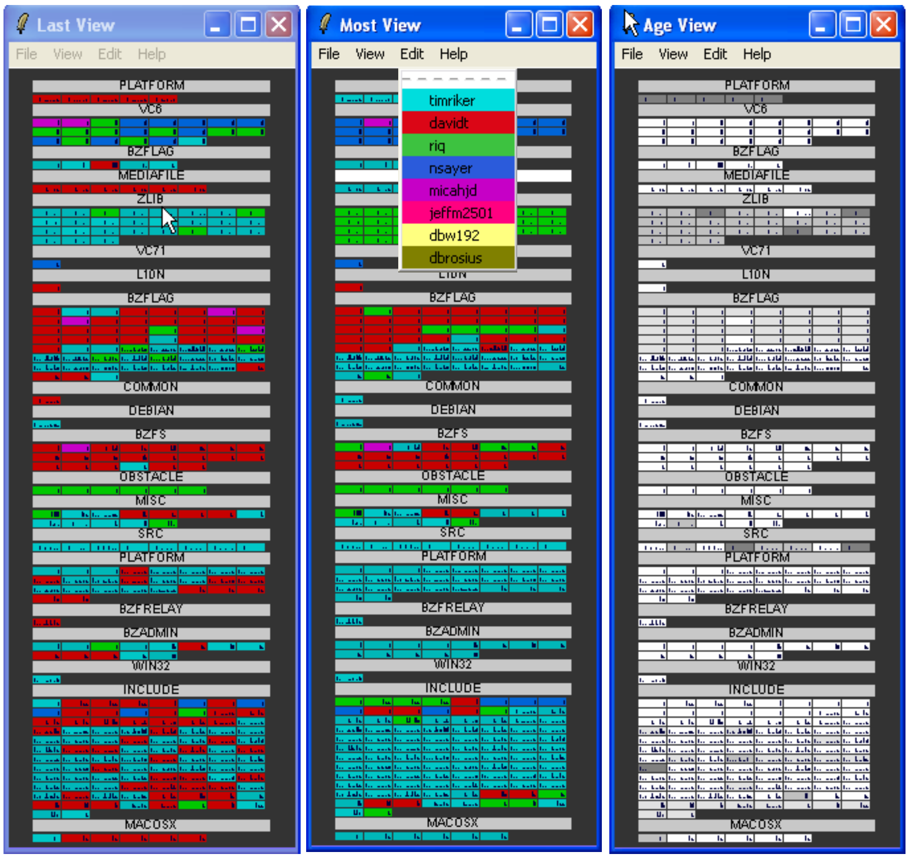


Fig. 4. Project overviews showing directories (grey bars) and files (coloured blocks) for a medium-sized game project with 322 files. Three types of filters are shown: at left, block colour indicates who changed the file most recently; at middle, colour shows who has changed the file most often; at right, grey level indicates the amount of time since last change. In each block, the bar graph shows the edit history since the start of the project. Developer colours are shown in a menu. Note that normally only one window would be used, with the filter changed through a menu selection.

The overview displays help developers to answer a variety of questions about the project and about the activities of their collaborators. For example, it can be seen that the developers timriker (light blue) and davidt (red) are currently active (since they have each been the last to touch several files), and are core developers on the project (since they are both the most frequent committer for many files). We can also see that developers riq (green) and nsayer (dark blue) are each likely responsible for one main module in the project, since they are the most frequent for all the files in a particular directory. Two other people, dbw192 (yellow) and dbrosius (brown) are neither recent or frequent committers, since neither filter shows any files in their colour. Finally, we can see from the ‘age’ filter (Figure 4, right) that most of the project has recently been changed, since most of the blocks are white or light grey.

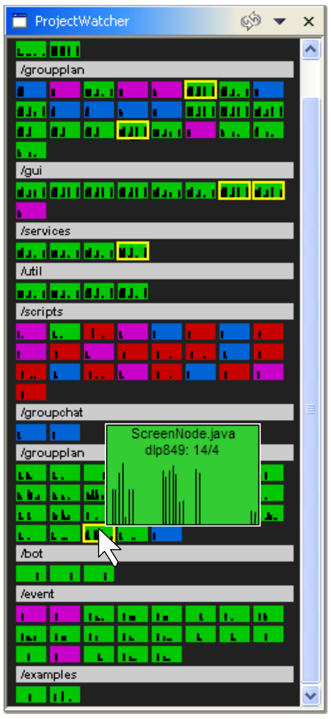


Fig. 5. ProjectWatcher as an Eclipse IDE plugin (www.eclipse.org), showing highlights (yellow borders on blocks) to indicate others’ recent changes, and popup window to show more detail about a particular file.

The highlights (see Figure 5) provide an analogue to the CVS-commits mailing list, but with considerably less effort. As can be seen in the figure, there are six files that have been changed since the local user last updated files from the repository. It is easy to determine how much change is occurring, and in general where it is happening. By holding the mouse cursor over any of these blocks, the developer, can get more information about what file has been changed, who committed the most recent change, and the number of lines added and deleted in the change (the ‘14/4’ in the popup indicates that 14 lines were added, and 4 deleted).

5. Comparison to Related Work

A number of software engineering tools provide some degree of information about other members of the team (such as their identities or their assigned tasks), or provide facilities for team communication (e.g., [2,6,19]). However, only a few systems combine information about people's activities with representations of the project artifacts. Two that do this are Augur [7] and TUKAN [20,21].

TUKAN is one of the first systems to explicitly address the question of awareness in software development. The basic representation used in TUKAN is a Smalltalk class browser, onto which awareness information is overlaid. In particular, the system shows the distance of other developers in 'software space,' using a software structure graph as the basis for calculating proximity. The main difference in our approach with ProjectWatcher is in the use of an overview; where TUKAN presents relevant information about others who may be encroaching on a developer's current location, ProjectWatcher provides a general overview of the entire project.

Augur is a system similar to Ball and Eick's SeeSoft [1], that presents line-based visualizations of source code along with other visual representations of the project. The goal of Augur is to unify information about project activities with information about project artifacts; the system is designed to support both ongoing awareness and investigation into the details of project activity. ProjectWatcher also uses the ideas of edit/read wear and combining activity and artifact information; the main difference between the two systems is that Augur is a large-scale system with many views and a highly detailed representation of the project, whereas ProjectWatcher's visualization is designed only to support the two awareness questions seen in our work with existing projects ("who is who in general" and "who works in this area of the code"). In addition, ProjectWatcher is based on a much finer temporal granularity of activity than is Augur, which uses repository commits as its source of activity information. We see ProjectWatcher as more suited to day-to-day activities on a collaborative project, and Augur to specific investigations where developers wish to explore the history of the project in more detail.

6. Future Research

Our future plans for ProjectWatcher involve improvements and new directions in both the mining and the visualization components. The current version of the system primarily addresses those awareness issues that we saw in distributed projects, but the basic tools and approaches can be used for a variety of additional purposes.

First, we currently visualize source code that is in the process of being edited, and therefore the source code may be inconsistent, incomplete and frequently updated. We are investigating techniques for improving the robustness and performance of the fact extraction process, and techniques for visualizing partial information given these circumstances. Our system also only records user edits to the method level. We plan to move towards even finer grained awareness so that we can handle concurrent edits in some situations.

Second, the capturing and recording of developers' activities supports new software repository mining research in addition to supporting awareness. Developers normally change a local copy of the software under development, and periodically synchronize their changes with the shared software repository. Unfortunately, the developer's local interactions with the source code are not recorded in the shared software repository. With our finer-grained approach, the local interaction history of the developer is recorded and is available to be mined. Example software mining research directions include:

- *Discovery of refactoring patterns*. Analysing local interaction histories may be useful for identifying novel refactoring patterns and coordinating refactorings that affect other team members.
- *Discovery of browsing patterns*. Local interaction history includes the developer's searching, browsing and file access activities. Analysing this browsing interaction may be useful in supporting a developer in locating people or code exemplars.
- *Discovery of expertise*. Since the factbase contains facts from the Java API, we can determine what parts of that API each developer has used, and how often. It can now be possible to determine who has used a particular Java widget or structure frequently, and to build that knowledge into the development environment.

We also plan to refine and expand the visualization component. Short-term work will involve testing the representations and filters to determine how the information can be best presented to real developers. Longer range plans involve extensions to the basic idea of integrating information about activities with information about project artifacts. For example, we plan to extend our artifact collection to include entities other than those in source code. Many other project artifacts exist, including communication logs, bug reports and task lists. We hope to establish additional facts to model these artifacts and to use the new artifacts and their relationships in the awareness visualizations. We can also extend our use of the interaction histories to other areas. As discussed above, recording developers' interaction history and extracting method call facts from the source code provides us with basic API usage information. We can present this information in the IDE to provide awareness of technology expertise.

Finally, we plan to extend the range of awareness information that can be seen in the visualizations. As mentioned above, displaying information about refactoring, browsing, and expertise may be useful to developers in a distributed project. Other possibilities include questions of proximity – “who is working near to me?” in terms of the structures and dependencies of the software system under development, and questions of scope and effect – “how many people will I affect if I change this module?” Proximity is an important concept in software development because developers who near to one another (in code terms) form an implicit sub-team whose concerns are similar and whose interactions are more closely coupled [20]. Proximity groups are not defined in advance and change membership as developers move from task to task; therefore, it is often very difficult to determine who is currently in the group. We will address this problem by extending the ProjectWatcher visualizations to make it easier to see proximity-based groups.

6. Conclusions

We have presented a system to address some of the awareness problems experienced in distributed software development projects. ProjectWatcher contains two main parts: a mining component and a visualization system. The system keeps track of fine-grained user activities through the use of a shadow repository, and records those actions in relation to the artifact-based dependencies extracted from source code. Second, visualizations represent this information for developers to see and interact with. The visualizations present a project overview, overlaid with visual information about people's activities. Although our prototypes have limitations in terms of project size, they can provide developers with much-needed information about who is working on the project, what they are doing and how the project is changing over time.

Acknowledgements

The authors would like to thank IBM Corporation for supporting this research.

References

1. Ball, T., and Eick, S. Software visualization in the large. *IEEE Computer*, Vol 29, No 4, 1996.
2. Chu-Caroll, M., and Sprenkle, S. Coven: Brewing better collaboration through software configuration management. *Proc FSE-8*, 2000.
3. Cordy, J., Dean, T., Malton, A., and Schneider, K., Software Engineering by Source Transformation - Experience with TXL, *Proc. SCAM'01 - IEEE 1st International Workshop on Source Code Analysis and Manipulation*, 168-178, 2001.
4. Dix, A., Finlay, J., Abowd, G., and Beale, R., *Human-Computer Interaction*, Prentice Hall, 1993.
5. Dourish, P., and Bellotti, V., Awareness and Coordination in Shared Workspaces, *Proc. ACM CSCW 1992*, 107-114.
6. Elliott, M., and Scacchi, W., Free software developers as an occupational community: resolving conflicts and fostering collaboration, *Proc. ACM GROUP 2003*, 21-30.
7. Froehlich, J. and Dourish, P., Unifying Artifacts and Activities in a Visual Tool for Distributed Software Development Teams. To appear, *Proc. ICSE 2004*.
8. Gutwin, C. and Greenberg, S. A Descriptive Framework of Workspace Awareness for Real-Time Groupware. *Journal of Computer-Supported Cooperative Work*, Issue 3-4, 2002, 411-446.
9. Gutwin, C., Penner, R., and Schneider, K., Group Awareness in Distributed Software Development, to appear, *Proceedings of ACM CSCW 2004*, Chicago, 2004.
10. Herbsleb, J., and Grinter, R., Architectures, coordination, and distance: Conway's law and beyond. *IEEE Software*, 1999.
11. Herbsleb, J., Grinter, R., and Perry, D., The geography of coordination: dealing with distance in R&D work. *Proc. ACM SIGGROUP conference on supporting group work*, 1999.

12. Herbsleb, J., Mockus, A., Finholt, T., and Grinter, R., Distance, Dependencies, and Delay in a Global Collaboration, *Proc. ACM CSCW 2000*, 319-328.
13. Hill, W.C., Hollan, J.D., McCandless, J., and Wroblewski, D. Edit wear and read wear. *Proc. ACM CHI 1992*, 3-9.
14. Kraut, R., and Streeter, L., Coordination in software development. *CACM*, 1995.
15. Malton, A., Schneider, K., Cordy, J., Dean, T., Cousineau, D., and Reynolds, J., Processing Software Source Text in Automated Design Recovery and Transformation. *Proc. 9th International Workshop on Program Comprehension*, 127-134, 2001.
16. McDonald, D., and Ackerman, M., Just Talk to Me: A Field Study of Expertise Location Finding and Sustaining Relationships, *Proc. ACM CSCW 1998*, 315-324.
17. Mockus, A., Fielding, R., and Herbsleb, J. Two Case Studies of Open Source Software Development: Apache and Mozilla, *ACM ToSEM*, 11, 3, 2002, 309-346.
18. Monk, A., and Watts, L., Peripheral Participants in Mediated Communication, Proc. ACM CHI 1998, v.2, 285-286.
19. Raymond, E., The Cathedral and the Bazaar, O'Reilly, 2001.
20. Schummer, T., Lost and found in software space. *Proc 34th HICSS*, 2001.
21. Schummer, T., and Schummer, J., TUKAN: A team environment for software implementation. *Proc. OOPSLA 1999.*
22. Segal, L., Designing Team Workstations: The Choreography of Teamwork, in *Local Applications of the Ecological Approach to Human-Machine Systems*, P. Hancock, J. Flach, J. Caird and K. Vicente ed., Erlbaum, 1995, 392-415.
23. Whittaker, S., Frohlich, D., and Daly-Jones, O., Informal Workplace Communication: What is It Like and How Might We Support It?, *Proc. ACM CHI 1994*, 131-137.
24. B. Zimmermann and A. M. Selvin. A framework for assessing group memory approaches for software design projects. *Proc. Conference on Designing interactive systems*. 1997.

Discussion

[Bonnie E. John] You chose no to look at video or IM Buddy lists, is that because prior research suggests that that is not where the action is, or was it easier not to do that, or what?

> [Kevin Schneider] We were interested in the software artefacts and what we could get from that! Other people in the CSCW field are working on other aspects such as the ones you mention. The field does not really know where the bang for the buck is.

[Bonnie John] You mentioned scalability! How big does it scale and do you have ideas of how you could chunk or aggregate to allow you to scale further? Are we talking about 10 person projects with 10,000 lines of code or a 100 person project with 1,000,000 lines of code?

> [Kevin Schneider] It is a big issue! I think the visualisation might not scale and that is why we are trying to think of other metaphors! Currently 10,000 to 100,000 would probably be the limit! Currently we use relatively little screen space and the projects we have looked at does not seem to need more than that! Other studies have shown that even large projects such as Linux tends to be organised around specific parts of the code and that might help solve the scalability problem you mention! Maybe it is software architecture that will have to solve that problem!

[Peter Forbrig] I like your tool very much. What about the software developers? Did they like to be tracked in this way?

> [Kevin Schneider] Because we were looking at open source projects there was no problem with privacy. Their community is willing to publish all activities. We can combine our approach with techniques to achieve privacy, but we did not look at it up to now.

[Bonnie John] Are real people using it and would they hate you if you took it away from them?

> [Kevin Schneider] Only internal people are using it, and we do not know if they would hate us if we took it away!

Author Index

Keywords Index