The Orbital Browser

Composing Ubicomp Services Using Only Rotation and Selection

Nicolas Ducheneaut
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304 USA
nicolas@parc.com

Trevor F. Smith
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304 USA
tfsmith@parc.com

James "Bo" Begole
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304 USA
begole@parc.com

Mark W. Newman
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304 USA
mnewman@parc.com

Chris Beckmann
University of California, Berkeley
387 Soda Hall
Berkeley, CA 94720 USA
beckmann@desuma.net

CHI 2006, April 22–27, 2006, Montréal, Québec, Canada.
ACM 1-59593-298-4/06/0004.

Abstract
Most ubiquitous computing environments are designed as collections of highly distributed and heterogeneous services. In this paper we describe a user interface, the Orbital Browser, which reduces the complexity of ubicomp service composition to two simple end-user operations: rotation and selection. We discuss the design requirements imposed by service composition and how we addressed them with our system.

Keywords
Rotation-based UI, Ubiquitous Computing, Service Composition

ACM Classification Keywords
H5.2 [Information interfaces and presentation]: User Interfaces - Graphical user interfaces.

Introduction
As we get closer to realizing Mark Weiser's [5] vision, the range of possible interactions and interconnections among computational devices is exploding. This makes developing individual applications for each combination of devices and services more and more impractical.

One possible way to address this issue is to rely on a ubiquitous computing infrastructure to proactively interconnect services for users (using, for instance, sensors). Our project is exploring alternative

possibilities where users are more actively in control of service composition. We are interested in designing interfaces allowing end-users to discover networked services, select a subset of them, connect them, and finally control them in an appropriate manner.

Service composition can take many forms. To use a simple example, users might want to direct their home network to play music files, stored on a portable music player, on the living room's stereo receiver. As the evening progresses they might want to redirect the music to speakers in the bedroom. To do so users have to discover heterogeneous resources (music players and speakers attached to different devices), connect them, and control them.

End-user service composition poses a set of important interface design constraints. In particular, users may have to interact with a large number of services having widely diverging capabilities. Also, users often have to use devices with limited input capabilities (e.g. cell phones, TV remotes, public kiosks), especially in common use domains such as the home or public spaces. In this paper, we describe how we designed and implemented an interface addressing these constraints: the Orbital Browser. Our system allows end-users to compose services using only two simple operations (rotation and selection) and a scalable and extensible set of interface elements.

The Orbital Browser

To test our concept with real applications and data, we built the Orbital Browser on top of Obje [4]. While our purpose is not to discuss Obje in detail in this paper, it is important to understand its main features:

- *Components* are available on the network (for instance screens, speakers, webcams, files, etc.).
- Several components can be grouped into an *aggregate* (e.g. a collection of multiple mp3 files).
- Each component can act as a *source* of data, a *sink* for data, or both.
- One or more components can be attached to a single *host* (a PC in an office, an integrated set-top box in a living room, etc.). For example, a set-top box may have separate components for the TV tuner (a data source), the recorded programs (an aggregate), and the connected display (a data sink).

Note that these concepts are generic enough that they are not limited to Obje. Any distributed system where the user is required to establish, maintain, and monitor connection could be mapped the same way.

To compose services in Obje users need to create connections between components, thereby initiating the transfer of data. This entails *browsing* a list of available hosts (e.g. Bob's laptop) and, once the right one has been found, *selecting the host* to browse its list of attached components (e.g. Bob's DVD player, Bob's screen). If one of these components is an aggregate (e.g. Bob's "music collection" directory), users will need to *expand* it to see the list of components it contains.

Assuming the right target has been found (for instance, "love me tender.mp3" on Bob's machine), users then need to be able to *select this component* as an endpoint (note that users can start with any arbitrary component, be it source or sink). Once this is done they can repeat the above process to select a second endpoint (such as "Living Room Speaker"). At this

stage they need to be able to effectively *connect* the two components. Once a connection is established, users need to remain aware that a live link exists between two components. Eventually they need to be able to *disconnect* them, starting from any endpoint.

Requirements
To make service composition more tractable, our interface had to meet a set of important requirements:

- *Minimalism*: we wanted to limit the number of basic user operations required to interact with our infrastructure. This requirement is not simply an attempt at reducing the user's cognitive load: with a small set of operations, users could compose services from a variety of devices - even those with severely limited input capabilities.
- *Scalability*. Highly networked systems can be difficult to visualize: as the number of components and connections increases, so does the visual clutter. We had to design our interface such that it could accommodate a large number of elements and yet remain visually sparse and easy to read.
- *Extensibility*. The components available within ubiquitous computing systems such as Obje are potentially very diverse, ranging from webcams to mp3 files. To accommodate this diversity and allow for the inclusion of as-yet unknown resources in the future, our interaction techniques had to be independent from the nature of the components being manipulated.
- *Traceability*. Establishing a connection between components can take many steps, and users can easily lose track of where they are in the process. Our interface had to clearly indicate what the user had already done and what remained to be done.
- *Explorability*. Since the nature of some services might initially be unknown, users need to be able to test certain operations and backtrack if needed – in other words, users need an "undo" mechanism for service composition.

Implementation
In the course of our design, we came to realize that we could both support the necessary set of end-user operations and satisfy our requirements' constraints by reducing the interface to two basic interaction techniques: rotation and selection. These two primitives are easy to implement with limited inputs (for instance a knob/button combination such as Griffin's Powermate or, on remote controls and cell phones, arrow keys and a "select" button, or a jog dial). They also map to graphical interface elements that are minimalist, scalable, and extensible: namely, nodes and edges.

Consequently, our Orbital Browser is (loosely) based on the "ball and stick" metaphor from the world of chemistry. In our system each "molecule" is a unique host on the network, represented by a small circle. Components are represented by larger "balls" connected to their host by a "stick." These components are all placed at an equal distance from their host, as if they were "orbiting" it (see Figure 1). In turn, all "ball and stick" compounds are arranged at an equal distance from each other.

In our current implementation, users interact with the Orbital Browser using a Powermate knob that can rotate in two directions and can also be pressed for a "click" action (Figure 1). A demonstration video is available at:
http://www.parc.com/nicolas/documents/orbital.mov

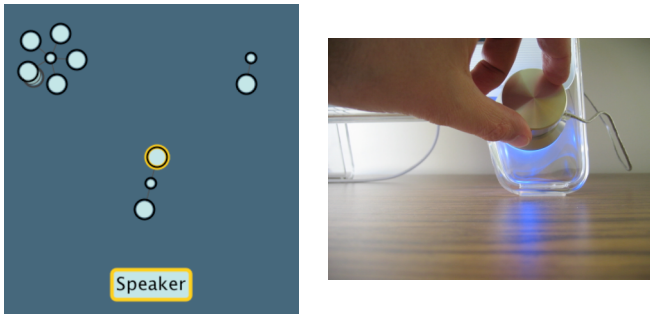


figure 1: the Browser's interface and its input device

Discovering and selecting components

In Figure 1, a yellow circle surrounding one of the "balls" represents the user's current focus. The "information bar" at the bottom of the window describes the current target (in this example, a Speaker component attached to one of three hosts). In the starting condition, focus is placed on the host closest to the top of the window (this is an arbitrary convention). Rotating the knob moves the focus from one host to another; pressing the knob selects the current host. At this point focus is transferred to the uppermost "satellite" of this host (as in Figure 1). This time, rotating the knob moves the focus from one component to another, on the same host. When the knob comes full circle, the focus steps through the host before going back to the level of components – by clicking the knob at this point, the user can go back to moving between hosts. This "undo" feature allows users to explore which components are available on a given host, and to backtrack easily if needed.

Note that our interface's graphical elements are exclusively "balls" and "sticks." The semantics of a "ball" are conveyed solely by its label. This way our interface can be used with components and hosts of any nature: it is extensible. It is also scalable: whenever the display becomes saturated with components (in our case, more than six), two arrows indicate that more material is available (upper left in Figure 2). Rotating the knob in the direction of the arrow scrolls the components' list accordingly. This allows our UI to scale to arbitrarily large number of components. Even with large lists, it is easy to quickly spin the knob and reach the desired focus.

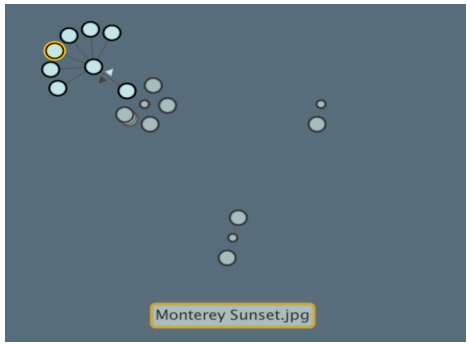


figure 2: navigating large sets of components

Once the user has selected a component, the Orbital Browser automatically grays out any component that cannot be connected to (Figure 3). The focus switches back to the host level and the user can start searching for another component to complete the connection. The grayed-out components are excluded from the rotation's path, speeding up this second selection.

Managing connections

Rotation and selection in the Orbital Browser allow users to navigate through an arbitrarily large number of hosts and discover the services/devices attached to them (no matter what these services are and how many there are). Once a component is selected for

connection, the Orbital Browser automatically narrows down the navigation path to the set of possible endpoints. Each step in the connection process is connected to the previous one, allowing users to backtrack at any point and clearly see what they have accomplished, as well as what remains to be done.

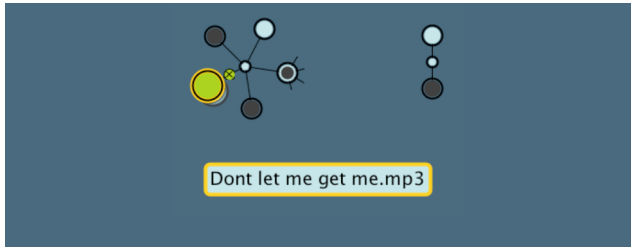
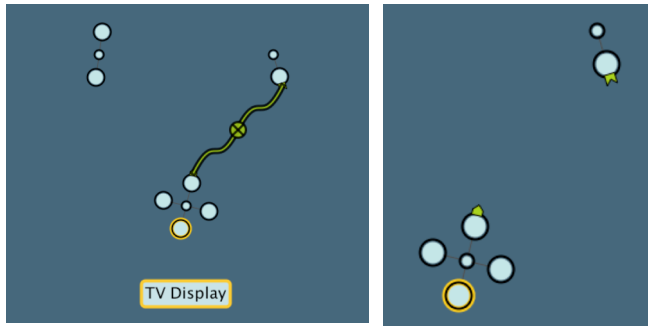


figure 3: selecting one component

Once a connection has been established by selecting two components, a line connecting "receptors" at the two endpoints briefly appears and then fades away (Figure 4a). The Orbital Browser does not constantly display lines between connected components to prevent visual clutter: instead, once a connection is established, only the receptors remain visible and are colored green (Figure 4b). To disconnect two components, the user can simply rotate and select until one of the active receptors is highlighted. This temporarily brings back the connection line on the screen, with the central "disconnection point" in focus. Pressing the knob at this point disconnects the two components; waiting a few seconds allows the line to disappear and lets the user backtrack, keeping the connection alive.

Dealing with aggregates

So far we have described how the Orbital Browser handles all the user operations we listed earlier except one: expanding aggregates. Aggregates potentially make navigation and selection more complex. First, they add additional "layers" of components around a host, beyond the simple ring of orbiting "satellites." Indeed aggregates can contain many components, and some of these components may be aggregates that, in turn, contain even more components, etc. Second, aggregates can either *contain* components the user is looking for (e.g. a file deeply nested in several directories) or *be themselves* the component the user is looking for (e.g. a directory can be a target for a file). Third, selecting a component within an aggregate is analogous to saving a search query: for instance, when the user has found the particular file he is looking for, this file becomes the component he needs to interact with – the aggregate containing it becomes irrelevant at this point.



(a) **(b)**

figure 4: (a) the connection line; (b) the receptors at the two ends of an active connection

To address these three issues and preserve the ability to interact with the Orbital Browser using only rotation and selection, we handle aggregates as follows. First, to avoid visual clutter, we never show more than two levels of depth when the user is navigating an

aggregate (see Figure 2). The previous levels, if they exist, are simply "stacked" below the aggregate's "ball"; they are "unstacked" if the user backtracks (using the same mechanism we described earlier with hosts and components). Second (and only in the case of aggregates), we disambiguate selection and navigation based on the duration of the user's click. A single click selects the aggregate itself as the component of choice, while a "click and hold" expands the aggregate into its sub-components. Third, if the user has selected one of these sub-components, the result is saved and appears as a new "satellite" directly connected to the host, not the aggregate. These "saved queries" are displayed as small stacks (see the upper left host in Figure 1), easily accessible from the main rotation path.

RELATED WORK

The concept of radial layouts is discussed in [1]. Recent applications have been, for instance, the visualization of large networks such as Gnutella [6], or the selection of songs in a list using the Apple iPod's click-wheel. However, radial layouts have not been applied to creating connections between devices and managing the resulting network.

Alternative ubicomp service composition interfaces include direct combination [2] and "jigsaw puzzles" [3] but these are not based on rotation and selection.

Conclusion

Our experience developing the Orbital Browser shows that it is possible to design user interfaces for service composition using a surprisingly small number of primitives. Of course, our system represents a clear departure from familiar HCI metaphors and users may initially be surprised by it. To address this potential limitation we plan to test the learnability and usability of our interface in upcoming user studies.

While the Orbital Browser satisfies our design constraints, some important issues remain to be addressed. In particular, the next version of the Orbital Browser will need to deal with errors more gracefully. For instance, it is currently impossible to tell whether a connection cannot be completed because the network is unresponsive, one of the components has crashed, the host has been turned off, or any other reason. This kind of system accountability is our next goal – end-user error diagnosis is another notoriously difficult problem with ubiquitous computing systems, since the potential points of failure multiply as the number of devices and connections grows.

References

[1] Di Battista, G., Eades, P., Tamassia, R.,Tollis, I.G.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice Hall, Upper Saddle River, N.J. (1999)

[2] Holland, S., Morse, D.R.,Gedenryd, H.: Direct Combination. Mobile HCI 2002, Springer Verlag, (2002)

[3] Humble, J., Hemmings, T., Crabtree, A., Koleva, B.,Rodden, T.: 'Playing with your bits': user-composition of ubiquitous domestic environments. Ubicomp 2003, Springer Verlag, (2003), 256-263

[4] Newman, M.W., Izadi, S., Edwards, W.K., Sedivy, J.Z.,Smith, T.F.: User interfaces when and where they are needed. UIST 2002, ACM, New York, (2002)

[5] Weiser, M.: The Computer for the 21st Century. Scientific American, 265 (3). (1991) 94-104

[6] Yee, K.-P., Fisher, D., Dhamija, R.,Heart, M.: Animated Exploration of Graphs with Radial Layout. Infovis 2001, (2001)