

# Tool Support for Divisible Interfaces

Jeffrey S. Pierce, Heather E. Mahaney  
College of Computing  
Georgia Institute of Technology  
{jpierce, mahaneyh}@cc.gatech.edu

## Introduction

The decreasing cost and increasing capability of computational devices has led to a shift away from users interacting with a single, personal computer. Users instead spread their activities across a variety of devices: desktop PCs, laptops, tablets, PDAs, cell phones, etc. While users currently interact with their devices largely independently of each other, the increasing communication capabilities of those devices will eventually lead users to coordinate their activities across devices, combining them to yield a personal information environment.

Current user interface design tools assume that users are working with a single device at a time. Even when an application supports multiple devices, the developers typically build each interface independently. That approach tends to lead to overly rigid, fragile interfaces: end users have little to no choice what parts of an interface to show on each device, and changes to an interface on one device can impact the interfaces on other devices.

The focus on developing interfaces for single devices has also led us to tightly bind the components of a user interface. We may build an application that allows an interface window to move across devices, but our tools typically do not allow us to *divide* a window so that some components appear on one device while others appear elsewhere. When users begin to coordinate their activities across multiple devices, we believe that developers will need to create *divisible interfaces* that allow end users to apportion interface elements across available devices to take maximum advantage of the available input and output resources.

## Supporting Divisible Interfaces

The basic idea of a divisible interface is straightforward: a user takes an interface that currently resides on a single device and selects interface components to either transfer completely to or mirror on other devices. We expect that the primary motivation for dividing an interface will be to take advantage of available I/O resources such as faster input devices and larger displays. As a simple example, a user might move the play, stop, and next and previous track buttons for a music application from his desktop to a nearby desktop PC to allow faster access to those controls while still using the desktop's higher-quality speakers.

The devices that a user divides an interface onto might be completely his own, or they might include devices owned by other people. As an example of the latter, a cell phone user might divide the interface for an email application on his cell phone onto a desktop computer in an Internet coffee shop in order to more easily read and respond to his messages. If we wish to support this ability to divide interfaces across untrusted devices, it is imperative that we address security and privacy concerns in our design tools from the start, rather than attempting to add them in an ad hoc fashion later on. In particular, we must allow the composition, content, and behavior of interfaces to change based on how much the user trusts employed devices. Divisible interfaces must be robust against “device-in-the-middle” attacks, where a device containing part of an interface can access functionality without the user's permission and modify information before displaying it to the user.

Based on initial experimentation, we believe that key elements for supporting divisible interfaces are:

- **Discovery and connection.** Users will need a light-weight, easy-to-use mechanism for discovering what devices are available, specifying which ones to divide an interface onto, and connecting to them.
- **Authentication.** Whether the user is pushing interface elements out to other devices or pulling those elements from his device, the device owners of those devices will want assurance that only authorized users can access them. Design tools will need to provide appropriate authentication mechanisms.

- **Interface description language.** Describing an interface using code is problematic when the interface might appear on heterogeneous platforms. A better approach is to describe the interface at a higher level and rely on description and interface generators to translate the description to an actual interface. In order to protect the user's privacy, the interface description language must be able to describe the conditional presence, content, and behavior of interface elements. For example, in the cell phone example above the user might want to hide the email address fields, remove any email addresses from the body of message he's responding to, and cause the Send button to return the message for verification rather than send it immediately when displaying a message composition window on an untrusted device.
- **Interface generators.** Rather than port an application and its interface to every possible device a user might employ, a more practical approach is to port a single interface generator to every platform. A particular device's interface generator will be responsible for taking an interface description and mapping the description to the interface elements provided by that device. We note that the mapping may be direct (for a highly capable device like a desktop PC) or it may require modification of the interface (for more limited devices such as cell phones).
- **Template library.** While it would be theoretically possible to create interfaces where users can arbitrarily divide interface elements across arbitrary devices, a more practical approach is to allow interface designers to create a small number of templates for the different divisions and devices that they expect. At least in the short-term, for example, providing templates for PCs, PDAs, and cell phones should cover the most commonly employed devices. Each application will then have its own template library for the supported interface divisions. Each device might also have its own template library for components (e.g. dialog boxes) that span applications.
- **Description generator.** An interface template describes that composition and layout of an interface, but does not contain any content. Applications will need access to a description generator that takes a template and fills it with content. The description generator will also bear the responsibility of determining the conditional properties of the interface template and modifying it accordingly. The application can then send the fully described interface to the appropriate device.
- **Application model.** While an application's interfaces might move between devices, we expect that the application's model will reside on a single device in order to protect potentially sensitive information. We believe that a possible consequence of divisible interfaces will be a stronger separation between the model, view, and controller components of an application.
- **Controller.** We expect that applications will incorporate multiple controllers. Each interface generator will provide a controller to quickly respond to local input: mouse clicks, key presses, etc. In addition, each application will also provide a controller to handle (possibly transformed) input remote devices.
- **Data bus.** Applications will need a logical data bus to move interface elements and commands between the devices that contain interface elements.

These elements are a necessary but insufficient condition for the development of divisible interfaces. We will also need to develop a set of design principles and best practices, based on a concrete understanding of the potential benefits and risks to end-users, that are well-suited to this new type of interface design.

## Conclusion

Users are moving away from interacting with a single, personal computer and toward interacting with multiple, heterogeneous computing devices. As users increasingly coordinate their activities across those devices, rather than working with them independently, we will need to create interfaces that allow users to apply the best combination of available I/O resources regardless of what devices those resources are attached to. We propose providing that functionality by allowing users to divide an interface across multiple devices. We describe some key elements that the next generation of user interface design tools will need to support in order to make divisible interfaces practical.