White Paper: The UIML Vision

1. Introduction

Imagine what could happen if the next time you sit down to use your computer, and punch in your name, the computer recognizes you and adapts itself to fit you. Everything makes sense to you. It fits your style and personality. The computer begins to anticipate your needs like a good nurse who readies a scalpel before a surgeon ask for it.

For example, you're a doctor on the way to the hospital. If your hospital's software was built with Harmonia's product, then with one call, your cell phone could display your schedule for the day. When you start your rounds carrying a handheld device, Harmonia's product displays screens that give you instant access to patient information, drug interactions, or whatever else you need. You no longer interrupt your rounds to find a computer terminal and wade through endless function keys. Later that day, on your office PC, Harmonia delivers customized screens to access your preferred medical database for diagnostic information.

Imagine what could happen if you could tailor the user interface[†] to the user: The format and content of the user interface would change to fit the resolution of your screen (e.g., a handheld device with a small screen versus a large desktop monitor). The portions of the application exposed by the interface would fit your user profile. A software application could "follow you" around: you could use it at the PC on your desk, then from a cell phone with a display while on the road, each time resuming from where you left off last time you used it.

It's been said that 80% of the people using complex applications only use 20% of the features, but everyone uses a different 20%. Harmonia's software can give each user a customized application, with a user interface that's customized to each person's department, job title, training level, spoken language, and so on. Everyone will see fewer mistakes, improve performance, require less training, and experience lower stress. And people won't feel as resistant to trying new technologies – an interface on your new handheld device will resemble the one you've used before on your desktop computer.

2. Dynamic User Interfaces

User interfaces today are *static*: everyone that uses a software application sees the same interface. Each person using the application must conform themselves to think and work the way the programmers who wrote the interface expected them to act. Today, interface customization is limited to things like setting colors and fonts, or hiding toolbars.

Harmonia is introducing a new technology: *dynamic* user interfaces. When you use a computer with a dynamic user interface, the computer transmits a profile of the user to an *Interface Server*. The profile specifies the *network appliance* (e.g., PC, cell phone) being used and information like the user's spoken language, the user's role (e.g., customer, sales person, technical support), the user's experience level (e.g., novice, expert), even the user's fatigue level. The Interface Server delivers a customized interface suited for that user using that network appliance under those circumstances.

[†] "User interface" means the menus, buttons, windows, and other images painted on your computer monitor when you use software. Or it could be a voice that you hear when using a computer application over a telephone. The interface defines what happens when you press keys, move a mouse, or press buttons on a telephone.

But dynamic user interfaces don't just make life easier for users. They also help the people that design and implement the interface. That's important because today you'd need an army of programmers to design a software application with different user interfaces customized to multiple network appliances, multiple spoken languages, and so on. It's too time consuming to write low-level programming language code. It's hard to even find the programmers with today's shortage of programmers. This is where UIML enters into the picture.

3. User Interface Markup Language (UIML)

Rather than write the user interface in a standard programming language, you use UIML. UIML is faster to use than traditional procedural languages, such as C++ or Java for composing interface components into a graphical interface or into an audio stream played on a speaker.

UIML is patterned after the World Wide Web's HTML in several ways:

1. UIML is easily learned:
    - UIML looks like HTML. UIML uses the emerging XML specification, an outgrowth of HTML.
    - Like HTML, a non-programmer can learn a little UIML and start designing interfaces, thus opening user interface design to those trained in graphic arts, psychology, and related fields.
    - UIML can be learned by example: when you encounter a dynamic user interface built by someone else with a feature you'd like to use in your own work, you can view the UIML source to quickly learn how to implement that feature.
2. Just as HTML can specify separately document content and presentation style (i.e., cascading style sheets [CSS-1]), so does UIML split content from appearance. In UIML, an appliance independent *user interface definition* specifies user interface content, and an appliance dependent *style sheet* guides the placement and appearance of user interface elements. Multiple style sheets can render UIML differently for different network appliances. Style sheets can also be cascaded, letting a user with poor vision override default font sizes with larger sizes.

    The advantage of style sheets is shown in Figure A1. Without UIML, programmers would have to code *12* separate user interfaces to run three applications on four devices. With UIML, only three UI definitions and four style sheets are written. Adding a fourth application that can be used on all of the four devices is done by simply writing one more UI definition!

UIML does things that have no analog in HTML:

- A user interface rendered on a monitor sometimes consists of hundreds of screens with similar appearance but different content (e.g., an interface for a hospital that serves many different specialists in the hospital). One can design a *user interface definition* in UIML, and content from a database will be added when the interface is actually rendered for a user. This saves labor in building interface software, and allows interface creation to be partitioned into work done by graphic or interface designers and work done by content providers.
- While HTML was originally intended to display text and images, UIML expresses interactions between a person and a computer. UIML does not embody any interface metaphor (e.g., graphical user interfaces for windowing systems), or interface hardware (e.g., a mouse or keyboard for input). A UI definition in UIML can, in theory, be rendered on any interface medium (e.g., speech and handwriting recognition, speech synthesis, eye tracking, virtual reality), even those yet to be invented. A style sheet maps a UI definition to the artifacts and interface metaphors of a particular interface medium.

- HTML can be embedded in UIML, so that Web pages can appear as part of an application interface.
- The *content* of an interface – text, images, sounds – are not embedded in UIML, but rather are retrieved from a database. This simplifies creation of international interfaces, and language appearing in the interface can be tailored to the audience (e.g., novice versus expert, adult versus child, engineer versus salesperson).
- UIML is user extensible. Arbitrary interface components (e.g., Active-X controls, Java Beans) can be used with UIML.

UIML interfaces with the underlying application by generating events for and receiving events from the application. In the initial version of UIML, the interface and the underlying application appear to each other either as Active-X controls or as Java beans.

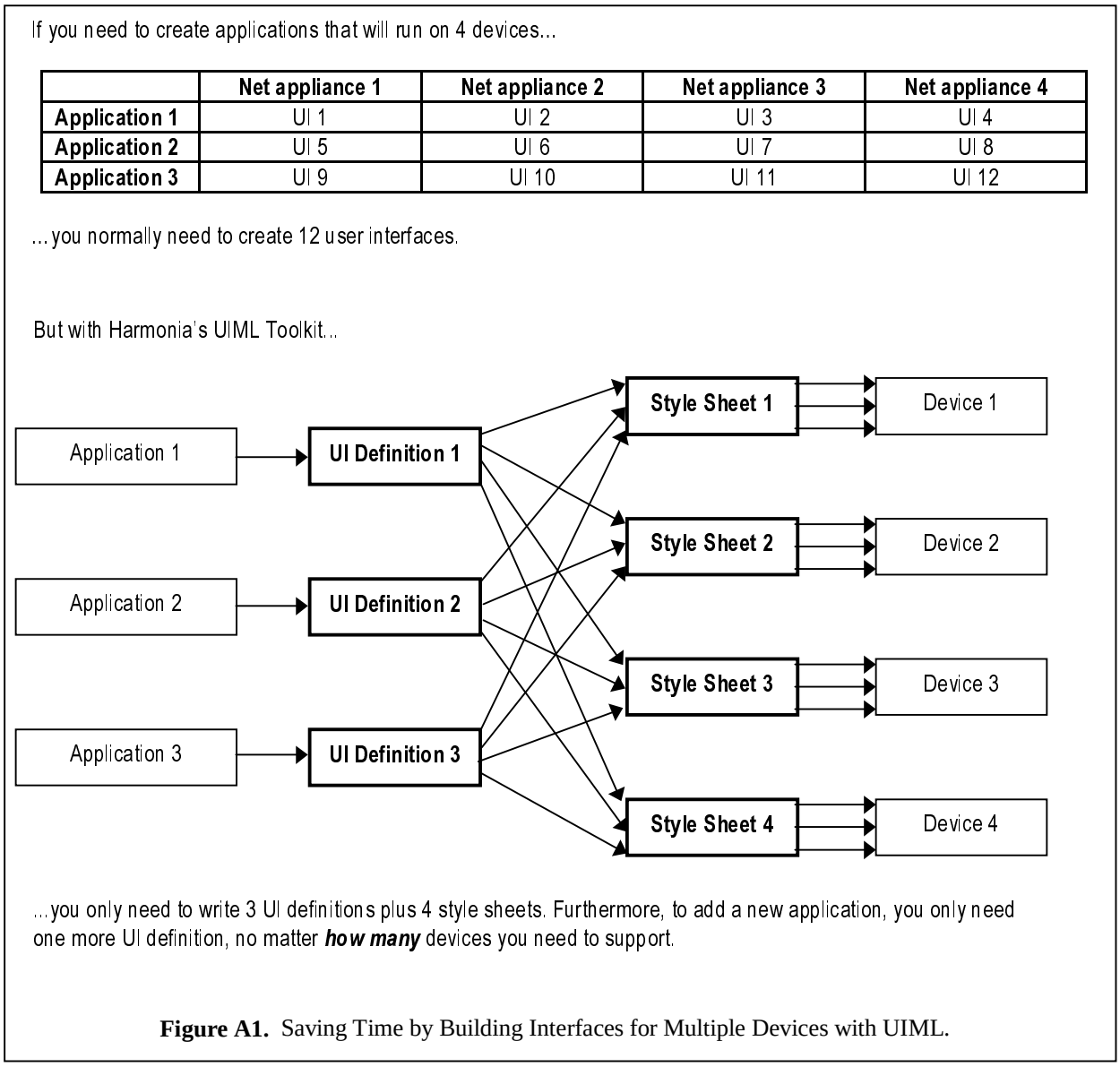

Figure A1. Saving Time by Building Interfaces for Multiple Devices with UIML.

4. Creating New User Interfaces

UIML is used through a *UIML Development Environment*. The Development Environment integrates the following into one convenient system:

- An *editor* through which one can directly write UI definitions and style sheets in UIML, to get total control over every aspect of the interface and its appearance.
- An *object browser*, used to find and reuse previously created UI definitions, style sheets, and interface components, and to view their properties.
- A *previewer* that interprets UIML to show how a UI definition and style sheet will be rendered on a particular network appliance: for example, the interface for a television is rendered with the television's low resolution, and the interface for a handheld device is rendered with the device's physical dimensions.
- A drag-and-drop *interface builder* for users who wish to draw interfaces graphically. The user can compose the interface by dragging user interface components from a palette onto the previewer's window. The builder automatically writes UI definitions and style sheets in UIML.
- A *database* into which one can store content that is needed by a UI definition.

Creating an interface with the UIML Development Environment differs from using conventional interface builders in several ways.

- One is not limited by the drag and drop capabilities of the interface builder, because the editor can be used to directly write and modify UIML by hand.
- Changes to appearance made in one interface screen can automatically update the appearance of other screens, through updates to a style sheet.
- A style sheet can be reused in other interfaces to gain a consistent look and feel across different software applications.
- One can automatically generate interfaces for several network appliances at one time (e.g., by writing one UI definition in UIML and then reusing existing style sheets for different network appliances) (see Figure A1).

5. Deploying the Interface to Users

After you've created UI definitions, style sheets, and content to create a user interface, it's time make the interface available to end users. Two more Harmonia components come into play (see Figure A2):

- The *Interface Server* combines the UI definition, style sheet, and content from a database into a *UIML instance*.
- The *UIML Renderer* maps a UIML instance into an arbitrary language and API, such as the Handheld Device Markup Language (HDML) for hand-held devices, the Windows MFC API for devices running Microsoft operating systems, and Java.

The Renderer runs either on the on the network appliance *(e.g., Device 1 in Figure A2)* or on the Interface Server *(e.g., Device 2 in Figure A2*. In the former case, the Renderer paints on the screen or plays on the speaker the buttons, menus, sounds, and so on that form the user interface, according to the UIML instance. In the later case, the network appliance requires an interface in a certain language, such as HDML, spoken voice for a conventional telephone, or even HTML. Thus the Renderer runs on the Interface Server to convert UIML into the needed language (e.g., HDML), and sends the language over

the network to the network appliance. In either case, a new Renderer and style sheet can be written when a new interface technology is invented (e.g., speech or holograms) to present existing UI definitions. With a new Renderer, organizations using UIML can immediately use the new technology on existing applications.

The user can then interact with the user interface on the network appliance. The appliance sends necessary information to the Interface server which, in turn, communicates with a back-end application (e.g., database, legacy system), shown on the right side of Figure A2.

6. Procedural Languages Versus UIML for Creating Interfaces

Traditional programming languages (e.g., C++ and MFC) are *procedural*: they specify *how* an interface is constructed. In contrast, UIML is *declarative*: it specifies *what* the interface contains without detailing how it is constructed. Switching to UIML offers many advantages:

- A few lines of UIML take the place of may lines of a procedural language.
- Novice programmers can eat up a lot of time discovering the details and idiosyncrasies of toolkits used with procedural languages (e.g., Java's AWT). UIML shields programmers from these problems.
- It's easy to enforce a consistent style (e.g., layouts, fonts, colors, keystroke accelerators) across many interface screens in an application with UIML's style sheets. There's no comparable facility with procedural languages.
- UIML offer everything in a traditional toolkit plus new, time-saving high level abstractions. UIML exposes relationships between pieces of the interface that low level procedural code obscures.
- The UIML Renderer looks like an Active-X control or a Java bean to the rest of an application, so UIML can be combined naturally with procedural languages.
- It's easy to move a user interface from a PC to a new network appliance (e.g., a handheld device) with UIML. Just write a new style sheet and do some simple editing of UIML But without UIML, heavy modification of procedural code is often needed.

7. Summary

Harmonia's Dynamic Interfaces and UIML can help you in situations such as these:

- You must deploy interfaces on a variety of network appliances, not just on PCs.
- You write client-server applications that run over networks.
- The content in your interface changes frequently.
- You want interfaces tailored to user demographics.
- Your application requires many interface screens.
- You want to create interfaces for disabled persons.
- Your application is used internationally.