Automatic Discovery and Execution of Personal Applications from Shared IO Devices

Bradley J. Rhodes, Sergey Chemishkian, Edward L. Schwartz, Stephen Savitzky, Haixia Yu
Ricoh Innovations, Inc.
2882 Sand Hill Rd
Menlo Park, CA 94025 USA

***Abstract*-Shared office information appliances such as copiers, scanners and meeting-room displays are increasingly being integrated into complex electronic document workflows. To support this new role these appliances are being designed with advanced features such as optical character recognition, networked storage, content-based routing and integration with back-end databases, yet availability of these features is often hampered by various factors. Here we present an architecture, code named Odessa, that moves all task-specific processing off the office appliance and on to a user's own device: PC / laptop, or smartphone. Key features of the architecture include a simple RESTful HTTP application interface and automatic service discovery.**

I. Introduction

Struggling to bridge the paper / electronic document divide, modern office information appliances such as copiers, scanners, printers and meeting-room displays have introduced a wide range of valuable advanced functions, like scanning to / printing from networked document servers, converting a scan into a searchable PDF file, scanning to email, automatic document routing and workflow tracking based on a document's content. These features are typically initiated from the MFP via a user interface, usually a touch screen.

In line with this trend, many modern MFPs allow deployment of applications created by independent software vendors (ISVs) directly on MFP. Examples include Canon's MEAP, Ricoh's ESA [1], Lexmark's eSF, Samsung's JScribe. This approach achieves desirable tight coupling with the MFP's control panel and underlying scan and print functions, but application development for these embedded systems falls short in several key categories (see Table 1).

Newer MFP platforms address these deficiencies by encouraging ISVs to implement their applications as Web services: business logic and image processing applications move to an external server, the MFP hosts only the core functionality and a Web-based user interface, and finally, applications and MFPs communicate using standard Web-based protocols such as HTML and SOAP/XML. Prominent examples of this style architecture include Sharp's OSA, Xerox's EIP [2] and HP's OXP [3].

Yet, these newer MFP platforms fail to address the latest productivity trends, loosely defined as "Web 2.0," in several key areas (see Table 2). To address these, researchers at Ricoh Innovations have proposed an alternative concept for functions delivery to MFPs (code named *Calypso*), and later have realized the Calypso concept in an architecture code-named Odessa, the subject of this paper.

Table 1. Deficiencies of MFP-deployed platforms

N	*Requirements*	*Problem Descriptions*
1	Developer-friendly	Application developers are limited to using whatever language and libraries are installed on the MFP
2	Maintenance-friendly	Applications reside on the MFP, deploying or upgrading may take long time, especially for sites with a large fleet of MFPs
3	Performance	Resource contention: applications run within the confines of the MFP's limited processor and memory, and must share these resources with core functions, often on a secondary basis.
4	User-friendly	Application interfaces are restricted to the MFP control panel, typically not well-suited for configuring complex tasks.

Table 2. Deficiencies of Web services-based platforms

N	*Requirements*	*Problem Descriptions*
1	*Individual control:* employees want to manage their own computing environment, especially in small-to-medium sized businesses.	Shared office resources, including office MFPs and related server-based applications are managed by IT departments, reluctant to delegate controls to end users.
2	*Simple tasks are simple:* users like to create custom solutions using a glue of lightweight scripting languages and popular frameworks, e.g. Yahoo! Widgets.	The overhead of developing Web services and corresponding Web clients is high, unacceptable both to IT departments and to power users.
3	*Custom applications and personal settings follow the user:* office workers like to having their personalized work environment where ever they happen to be working.	That is not the experience when the user walks up to an MFP at a hotel business center or coffee shop: generic interfaces, customized for the locality.

II. The Calypso Concept

The Calypso concept is based on several key requirements. Advanced functions move off the MFP to a user's own personal device (PC, laptop, phone). These functions are

packaged as "widgets," i.e. small pieces of code that users can trivially download from the web and install on hardware they themselves control, much like one can install Yahoo! Widgets or Apple's Dashboard Widgets. Each widget installation is 'owned' by a single user, and personalization and configuration is done at the user's personal device. Widget features are discovered as services by all MFPs on the local subnet. Discovered widget service(s) are automatically added to a menu on the control panel of every MFP on the local subnet.

The user standing at the MFP uses the MFP UI to select his or her own name from a list of known widget owners, select the desired widget, and press Start. Depending on the widget, the user might be prompted to scan a document (which would be transmitted to the widget for further processing), the widget might cause a document to print, or both. The user might also be prompted to enter additional information at the control panel, for example by checking off checkboxes.

III. The Odessa Architecture

In designing an architecture to realize the Calypso concept, our team identified several key requirements. MFPs have to automatically discover new widgets without the need for *any* additional configuration on the part of the user. Widgets and applications must be implementable using a wide range of frameworks, including lightweight scripting languages such as JavaScript which typically cannot supporting incoming network connections or run network discovery protocols. Finally, the architecture has to be lightweight and based on widely accepted standards. The resulting architecture (code named *Odessa*) consists of three types of components: *widgets*, *providers,* and *widget proxies*. Widgets run on the user's device and handle task-specific logic, providers (MFPs) allow users to select a widget from a menu and provide input and output services, and widget proxies run on user's device and handle the server and advertisement functions required to make the system work (fig. 1).

Communications between widgets and widget proxies are defined by a Widget Protocol, and communications between widget proxies and providers are defined by Provider Protocol. Communication is encoded using a RESTful architecture [4], with each service represented by an unguessable unique base URL. Widget proxies advertise widget services, and providers discover widget services using the Bonjour protocol [5].

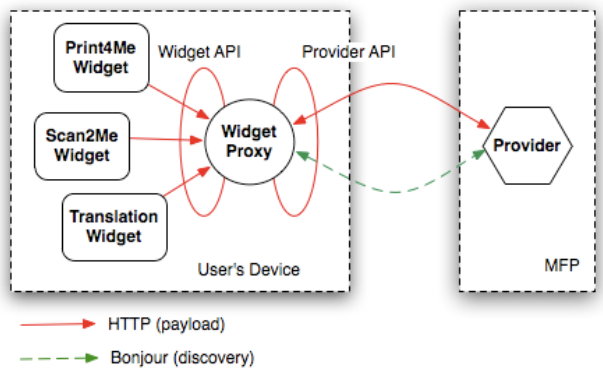

Figure 1. Odessa architecture components

IV. Evaluation

We have implemented cross-platform Widget Proxies (Java and Python), and extended Ricoh MFPs to host the provider protocol using Ricoh's Java-based Embedded Software Architecture™ [1]. These extensions have been available internally to Ricoh Innovations researchers, and to Ricoh Company, Ltd. application developers for about a year., and RII has hosted several workshops for interested developers. Developers were given sample code in a variety of languages and frameworks for two baseline widgets: *Print4Me,* a drag-and-drop print-on-demand widget, and *Scan2Me*, which received a scanned page and displayed it on the screen. Using these examples, researchers and developers created a variety of widgets, using a wide variety of languages and programming frameworks (including Java, C#, Python and Objective C / Cocoa, Bash scripts, JavaScript, Yahoo! Widgets, and basic unscripted HTML forms), on a variety of mobile devices, and in many cases within a few days of being introduced to the Odessa architecture. Examples of the different kinds of widgets produced include a *translation widget* that scanned a document, performed OCR, passed the text through Google Translate and printed the results, and a *forms widget* that accepted a filled-out form and then printed a next page to be filled out based on the previous form's content.

V. Open Issues and Future Work

Odessa protocol has only minimum security features, adequate for a typical home or small-to-medium sized business environment. The protocol has to be hardened for a enterprise or a higher-security environment.

In several use scenarios Odessa services advertisement have to go beyond the local subnet, e.g. when MFPs and user devices are on separate subnets. This can be addressed via advanced router configuration, or with Bonjour extensions collectively known as Wide Area Bonjour [5].

References

[1] "Embedded Software Architecture™," white paper, Ricoh Company, Ltd., March 2008.

[2] Brian Bissett, "Expanding the MFP Ecosystem with Xerox's Extensible Interface Platform (EIP)," white paper, Bissett Communications Corp. on behalf of Xerox Corp., June 2008.

[3] Keith Moore, "HP Open Extensibility Platform: streamlining paper-intensive business workflows," white paper, Hewlett-Packard, June 2008

[4] R. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," doctoral dissertation, Dept. Information and Computer Science, U.C. Irvine, Irvine, California, USA, 2000.

[5] D.H.Steinberg and S.Cheshire, *Zero Configuration Networking: the Definitive Guide*. O'Reilly, 2005.