Appliance Data Services: Making Steps Towards an Appliance Computing World

Andrew Huang[1], Benjamin Ling[1], John Barton, Armando Fox[1]
Internet and Mobile Systems Laboratory
HP Laboratories Palo Alto

HPL-2001-30
February 14th , 2001*

E-mail: ach@cs.stanford.edu, bling@cs.stanford.edu, John_Barton@hp.com, fox@cs.stanford.edu

digital,
appliance,
Internet,
infrastructure,
service

Although digital appliances are designed to be easy to use, their users often cannot even perform simple tasks because the devices lack infrastructural support. The Appliance Data Services project seeks to explore the attributes of an appliance computing world and develop the infrastructure required to support users with digital appliances.

Appliance Data Services: Making Steps Towards an Appliance Computing World

Andrew C. Huang
ach@cs.stanford.edu

Benjamin C. Ling
bling@cs.stanford.edu

John Barton
John_Barton@hp.com

Armando Fox
fox@cs.stanford.edu

Abstract

Although digital appliances are designed to be easy to use, their users often cannot even perform simple tasks because the devices lack infrastructural support. The Appliance Data Services project seeks to explore the attributes of an appliance computing world and develop the infrastructure required to support users with digital appliances.

1 Introduction

Digital appliances, such as digital cameras and digital photo frames, are designed to be easier-to-use, more powerful improvements of their non-digital counterparts. Thus, digital appliances should enable all consumers to accomplish traditional and advanced tasks more easily. For example, TiVo™ makes recording TV shows exceedingly simple by daily downloading TV broadcast information. Being connected to the network also enables extra functionality; TiVo™ recommends shows users are likely to enjoy by correlating the feedback of all its viewers. The simplicity and power of such devices is possible because they leverage the computational power, network bandwidth, content, and aggregate user-base of services in the infrastructure. We capture this effect by saying that these devices are "infrastructure enabled" [4].

Although many infrastructure enabled appliances are easy to use, the reality for many digital devices is that users are unable to accomplish even the simplest tasks with them. Dan Carp, CEO of Kodak™, identifies usability as the main hindrance to widespread acceptance of digital cameras [1]:

> The industry has made picture-taking more difficult and more complicated by cramming onto digital cameras more features, more buttons and more bells and whistles than most people want or need... The one lesson that 100 years of consumer marketing should have taught us: In the picture business, simple trumps megapixels, every time.

Therefore, despite a digital device's powerful features and low price, the user-experience turns many consumers away.

The usability problem does not end with the device. Extracting data from these devices requires users to install, configure, and learn how to use new software on their PCs or handheld devices. For example, mobile users that share information in a collaborative setting often reconfigure the network settings on their devices to gain network connectivity. Even printing photos from a digital camera requires the user to install new PC software. For everyday users, having to deal with these software issues can turn them away from adopting a digital device.

Hence, although digital appliances are designed to be easier to use and more powerful, the devices and the supporting services have the exact opposite attributes; they are **more difficult to use** and have **too many features**. Such devices are unable to achieve the status of "consumer devices" and the appliance computing world is still unrealized.

The goal of ADS is to address these usability problems by integrating digital appliances and Internet services, identifying principles that make appliance services successful for users, and to building a testbed to validate those principles.

2 An Appliance Computing World

People express high-level tasks in terms of concrete artifacts and the data residing on them: "Display the notes I've taken on my PDA on a wall monitor to be viewed by everyone at the meeting" or "Put this picture taken with my digital camera onto my Web page." Our vision is based on this observation:

> *An appliance computing world is one in which people move data effortlessly among artifacts to accomplish a variety of tasks.*

In ADS, we attempt to systematically explore this appliance computing vision by:

1. making observations on attributes inherent in this vision;
2. identifying the principles that underlie these attributes; and
3. building a framework based on these principles to test their effectiveness.

2.1 Bring devices to the forefront

In an appliance computing world, the number of steps required to perform high-level tasks should be few and the individual steps kept simple. A key insight into making the steps simple is this: people find it easier to use concrete artifacts to move data. Thus, our appliance computing vision has the following attribute:

Attribute 1: People move data using concrete artifacts.

In a digital world, tangible artifacts include digital cameras, flash memory cards, wall-mounted displays, and Web pages. In these examples, users perceive the data as being produced by or residing on artifacts that are concrete and permanent, which makes reasoning about tasks involving these artifacts easier. The problem is that today's digital devices force users to deal with objects they do not understand – computers and files. Extracting data from digital devices and moving the data to the desired destination often involves interacting with PCs. For example, posting pictures on a Web page requires knowing what server hosts the page. File format conversions are even harder to reason about for some users who are unaware that files have different formats. For example, when posting TIFF images produced by a handheld scanner onto the Web, a user focused on Web publishing should not be required to fix the broken icon that results from trying to display a TIFF file on a Web page.

To allow people to reason about the source and destination of data, rather than the path it must take so that data can be moved seamlessly across artifacts, the appliances they use should be sufficient to complete the task; using a secondary computer, like a PC, should not be required. Thus, the first attribute gives rise to the following principle:

Principle 1: Bring devices to the forefront.

This idea of bringing devices to the forefront and pushing the experience of using computers into the background is related to Mark Weiser's vision of ubiquitous computing where devices and computers are "invisible" in that they are embedded into the physical infrastructure [5]. Both allow users to focus on the task at hand rather than the mechanisms for accomplishing the task.

2.2 Keep devices simple

One approach to eliminating the PC experience is to push more functionality onto the devices. For example, some high-end cameras have networking and image-editing capabilities. However, moving computers into the devices does not solve the problem. As Alan Cooper explains, such devices do not make tasks easier because they are basically hard-to-use, complex mini-PCs [2]:

> My newest camera... has a full-blown computer that displays a Windows-like hourglass while it "boots up"... its On/Off switch has now grown to have *four* settings... and none of my friends can figure out how to turn it on without a lengthy explanation... The camera may still take pictures, but it *behaves* like a computer instead of a camera.

Cooper's observations suggest that devices must be be simple so that the steps required to accomplish a task can be made simple.

Attribute 2: Devices are simple, single-purpose appliances.

Given that devices should be kept simple, the software and hardware placed on the device should be kept to a minimum. For digital devices that have analog counterparts, this means user-controllable features outside of the feature set to which users are accustomed should be kept to a minimum. In fact, the only extra mechanism required is the ability to transfer data to and from the device.

Principle 2: Keep the number of user-controllable features on devices to a minimum.

2.3 Place software in the infrastructure

Up to this point, we have focused on simplifying the steps required to perform high-level tasks. We accomplish this by removing features from devices and eliminating the need to use PCs. Now the question is, what will people be able to do with these simple devices? At the very least, users should be able to use these devices to perform the same tasks as their traditional, non-digital counterparts, if not more.

Attribute 3: People perform a variety of traditional tasks, as well as a new set of advanced tasks with their devices.

This third attribute, when put next to attributes one and two, reveals the tension between providing functionality and providing a usable experience. We want to minimize the set of features placed on the device, but we want to allow users to control data movement using these devices without having to interact with a PC. At the same time, we want to provide users with a meaningful set of tasks that they can perform with their devices. The tension introduces tradeoffs that can be addressed using human-computer interaction and systems ideas.

So where does the functionality lie to perform the high-level tasks that users demand? One possible location is the user's PC. This software can be designed in such a way as to eliminate the look and feel of a PC, thus shielding the user from the PC experience. However, this does not relieve the user of other PC experiences such as installing, configuring, and upgrading software. Another possible location for placing functionality is the network infrastructure. Placing software in the infrastructure has the advantage of fully relieving the user of the PC experience:

Principle 3: Place the software required to accomplish tasks in the network infrastructure.

Taking an infrastructure-centric approach, one where we move functionality from PCs and devices into the supporting infrastructure, has other benefits. Having logically centralized software makes upgrades and administration much simpler. Furthermore, by selecting the Internet infrastructure, we are able to take advantage of the wealth of existing Internet services.

2.4 Appliance Data Services

The ADS framework is a general application framework on top of which appliance computing applications are built. The framework implements the previously mentioned principles of appliance computing: bring devices to the forefront, minimize the number of features on devices, and place software required to accomplish tasks in the network infrastructure.

Clearly, other principles and challenges exist in the creation of an appliance computing world. One example is the challenge of actually designing easy-to-use device and system user interfaces. Although this area of research is beyond the current scope of this project, we design the ADS framework such that it is amenable to new user interfaces and usage models.

In the following sections, we describe the ADS architecture, observations on building ADS applications, and next steps for future research.

3 The Architecture

Before discussing the ADS architecture, we describe its basic data unit. The basic data unit is composed of a user identifier, the command to be executed, and the data to be operated on: (userid, command-tag, data). The reason the userid and command-tag are part of the basic data unit is that they are used for:

1. Application selection: The command-tag names the high-level application the user wants to perform. However, since different users may have different semantic meanings for the same tag (e.g. the mapping for "my Web site" is user-specific), a userid is required to fully specify an application.
2. Access control: The userid is required to determine what credentials are to be attached to the application request since some services may limit access to a set of authorized users.
3. Other service features: Services such as billing, security, and personalization are not implemented in ADS, but we have left the userid as a "hook" for adding such capabilities later.

The ADS framework architecture, shown in Figure 1, is described in the rest of this section. The framework is split into three main components – Data Receive stage, Application Control stage, and Services Execution stage – each of which corresponds to a high-level function that is performed on the data.

3.1 Data Receive

The components in the Data Receive stage, Access Point and Aggregator, interface with devices to receive data and output completed (userid, command-tag, data) triples.

<u>Access Point</u>: The Access Point consists of the hardware and software needed to receive data from devices. Examples of hardware are IR transceivers and cradles for "docking" a device. The software is organized as a set of device adaptors, each enabling the Access Point to "speak" a different device communication protocol.

The architectural role of the Access Point is to address the issue of dealing with the various devices communication protocols.

Role: Deals with device heterogeneity.

Isolating device heterogeneity to one component relieves the rest of the system from having to deal with device-specific communication protocols, which makes building the rest of the system simpler.

The key challenge in designing the Access Point is extensibility. This challenge exists because of the lack of device standardization and the increase in the variety of devices being introduced. Standardization among competing vendors is not likely anytime soon, and since general-purpose devices are often difficult to use for everyday consumers, the trend towards single-purpose devices is likely to continue. Thus, the rapid development of heterogeneous devices makes it crucial to make adding support for new devices simple.

<u>Aggregator</u>: A stateless Access Point improves its extensibility because adaptor writers do not have to deal with state management. To allow statelessness and extensibility in the Access Point, the Aggregator manages state by gathering the devices' data and sending completed triples to the next stage.

Role: Simplifies adding support for new devices and protocols.

The net effect of separating the Data Receive stage into Access Point and Aggregator components is to separate the two concerns of device heterogeneity and state management.

3.2 Application Control

In the Application Control stage, the userid and command-tag are used to determine the chosen application. The data is then added to a list of parameters required for the selected application. Once all the parameters for a given application have arrived, they are sent to the Services Execution stage.

<u>Command Canonicalizer</u>: The Command Canonicalizer facilitates the design of "no-futz," easy-to-use user-interfaces. Canonicalization involves converting the command-tag from its original data type to plaintext. Giving the system the ability to handle command-tags

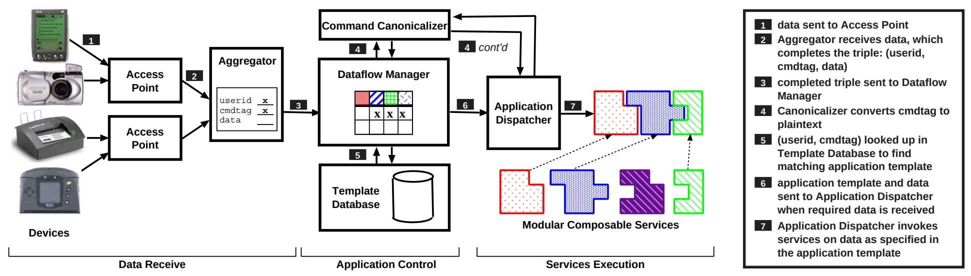
1
Access Point
Access Point
2
Aggregator
userid x
cmdtag x
data
3
Command Canonicalizer
4
4 cont'd
Dataflow Manager
x x x
5
Template Database
6
Application Dispatcher
7
Modular Composable Services
Devices
Data Receive
Application Control
Services Execution
1 data sent to Access Point
2 Aggregator receives data, which completes the triple: (userid, cmdtag, data)
3 completed triple sent to Dataflow Manager
4 Canonicalizer converts cmdtag to plaintext
5 (userid, cmdtag) looked up in Template Database to find matching application template
6 application template and data sent to Application Dispatcher when required data is received
7 Application Dispatcher invokes services on data as specified in the application template

Figure 1: The ADS Architecture

of arbitrary types makes it possible to support arbitrary devices, even those with limited user interfaces.

Role: Allows devices to have simple user interfaces.

For example, the most natural method for digital camera users to specify the command-tag might be to record a short WAV file annotating each picture. In this scenario, the user takes a picture and speaks the desired command-tag into the camera; later, when the pictures are transfered into an Access Point, the command-tag has already been specified so the system knows what to do with the picture. Canonicalization frees the device designers from being constrained to relying solely on menu or other text-based user-interface elements, thus facilitating the most natural user interface for a no-futz, easy-to-use experience.

Template Database: The canonicalized command-tag, userid pair is looked up in the Template Database to find a matching application template. Templates define an application's behavior by describing the data required for a given application and specifying the services to invoke on the data. Binding command-tags to application templates in the Template Database has the benefit of minimizing device configuration and supporting devices with non-extensible user interfaces, thus achieving out-of-the-box operation for devices.

Role: Minimizes device configuration.

Application templates and the command-tag mappings are configured for a particular user independently of the user's devices. Further, in situations where a command-tag cannot be specified, such as may be the case for devices with non-extensible user interfaces, the command-tag "default" can be mapped to the appropriate application template.

This binding mechanism provides a level of indirection between application selection and application specification, which separates the concerns of applications users and application creators. This separation provides an easy way for third-party developers to make their templates available to ADS users. Furthermore, with authentication mechanisms in place, third-party template providers can effectively restrict access to the templates it has developed.

For example, say Kodak™ develops a set of ADS applications, which only Kodak™ camera customers can use. If the templates are shipped with each camera, not only is upgrading or adding new applications difficult, it may be possible for the template to be "pirated" and given to users with non-Kodak™ cameras. With the Template Database, the user's act of registering gives that user the credentials to view and select the Kodak™ templates. Furthermore, upgrading applications or adding new ones only requires dealing only with the Template Database.

Dataflow Manager: The role of the Dataflow Manager is to coordinate data received from the user and to make certain an application has all the data it requires. When data is received from the Data Receive stage, the Dataflow Manager uses the application template to place the data into the proper parameter slot for the chosen application. Once all necessary data is received, the Dataflow Manager sends the template and all the data to the Services Execution stage.

Role: Coordinates data input by the user.

Coordinating data in this way allows users to input the data from different devices and at different periods of time. For example, a user who uses a PDA to store captions for the photos taken on a digital camera can create a Web-based photo album by inputing the data from these two devices. As an alternative, a user can input photos into ADS while still on vacation to conserve the camera's memory. At the end of the vacation, the user can use a Web browser to fill in the captions for all the pictures.

3.3 Services Execution

In the Services Execution stage, the Application Dispatcher invokes the services specified in the application template on the data it receives. The reason modular composeable services are used is that it results in applications that are flexible and whose components are reusable. However, such a service framework does not preclude the use of stand-alone, monolithic applications. Such an application would simply be a single service in the framework and would not be invoked in conjunction with other existing services.

3.4 Architecture Summary

The ADS framework was designed to provide applications built on top of it with the three appliance computing principles discussed in the previous section:

Principle 1: We bring devices to the forefront so that people can focus on using concrete artifacts for moving data. The Template Database separates application selection and creation so that users can perform high-level tasks simply by selecting a command-tag. Meanwhile, the extensible Access Point allows ADS to support a variety of devices.

Principle 2: We facilitate keeping the number of features on devices to a minimum so that devices can be made simple and easy-to-use. The Command Canonicalizer allows devices to be extended for command-tag selection in the most natural way without loading-up the device with features. Meanwhile, the Template Database minimizes device configuration by allowing application creation and customization to be done independently of the user's devices.

Principle 3: We place the software required to accomplish tasks in the network infrastructure so that people can perform a variety of tasks without dealing with complex devices or the PC experience. The Dataflow Manager coordinates data so that a variety of tasks using one or more devices can be performed. Furthermore, all components except for the Access Point, which can be deployed as public Web-kiosks or appliances in people's homes, are placed in the Internet infrastructure to free the user from software issues.

4 Development Experience

To test the effectiveness of our appliance computing principles, we built a prototype of the framework as described in the previous section. We evaluate the implementation based on how easy it is for developers to build applications on the framework. To do this, we built a set of services and device adaptors that supported two target applications: Web Photo Album and Guest Book. The Web Photo Album shows how the usage model of digital devices can be simplified by infrastructure services; in this application, users create and publish Web-based photo albums on Geocities™ in a few simple steps using their digital cameras. The Guest Book exhibits the potential of ADS to coordinate input from multiple devices; this application takes input from a Web cam, business card scanner, and PDA to create a Web-based guest book containing people's pictures and business card information.

Given the necessary services and device adaptors, building these applications simply involved creating an XML application template to describe the input data and services to invoke; no coding was necessary. Our experience building these applications showed us that: 1. creating and extending applications using XML templates is simpler than building standalone applications from scratch, and 2. adding support for new devices is simple because of the statelessness of device adaptors in the Access Point. Thus, ADS has been successful in providing a framework on which appliance computing application can be quickly built, customized for each user, and evolved. Furthermore, the ADS architecture provides these applications with the desirable attributes of our appliance computing vision in that people use simple artifacts to move data around in a variety of ways.

5 Next Steps

Two areas of research we intend to explore in the immediate future involve deploying ADS in "Smart Space" environments and adding mechanisms to convey status information back to the user.

In Stanford's Interactive Workspaces Room (IW-Room) [3], researchers explore new possibilities for people to work collaboratively in "meeting rooms of the future" using a variety of computing and interaction devices. We intend to explore ways ADS can be deployed in "Smart Space" environments such as these by installing a production version of the ADS framework in the IW-Room. Initial applications will include the Guest Book deployed at the entryway of the room and an application that allows users to share information from their personal devices by "beaming" data onto wall-mounted displays via IR-dongles. Since IW-Room is in production use for regularly scheduled meetings, we expect to gain insight on new applications and deficiencies in the architecture that need to be addressed.

As ADS is used in production form, we expect users will want more feedback about the status of the data they send into the system. The solution is not as simple as adding error dialog boxes with error messages. The lack of a traditional computer interface and the target audience of everyday consumers means that novel approaches to convey status and error information to the user need to be explored.

6 Conclusion

Our vision for appliance computing is a world in which everyday users move data seamlessly and effortlessly among various artifacts. While the devices required for such a world exist, users often cannot even perform the simplest tasks with current devices. These devices are simply too difficult to use because they lack infrastructural support. The goal of the Appliance Data Services project is to explore the attributes of this appliance computing world and develop the infrastructure required to support users with digital appliances. To accomplish this, we identified three principles for realizing this vision and implemented a testbed to see how successful the principles are in making appliances more useful and easier to use. Next steps involve putting the ADS framework into production use and gaining valuable user and developer feedback on how the framework can be improved and what applications are needed.

References

[1] Dan Carp. Keynote address. In *Advanced Digital Photography Forum*, Boston, MA, USA, April 2000.

[2] Alan Cooper. *The Inmates Are Running the Asylum: Why High Tech Products Drive Us Crazy and How To Restore The Sanity.* Sams, 1999.

[3] Armando Fox, Brad Johanson, Pat Hanrahan, and Terry Winograd. Integrating information appliances into an interactive workspace. *IEEE Computer Graphics and Applications*, 20(3):54–65, May/June 2000.

[4] Andrew C. Huang, Benjamin C. Ling, John Barton, and Armando Fox. Running the web backwards: Appliance data services. In *Ninth International World Wide Web Conference (WWW9)*, Amsterdam, Netherlands, May 2000.

[5] Mark Weiser. The computer for the twenty-first century. *Scientific American*, pages 94–100, September 1991.