Making Computers Disappear: Appliance Data Services

John Barton, Andrew C. Huang[1], Benjamin C. Ling[1], Armando Fox[1]
Internet and Mobile Systems Laboratory
HP Laboratories Palo Alto
HPL-2001-21
January 29th, 2001*

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

appliance, ubiquitous computing, Internet, infrastructure, service

Digital appliances designed to simplify everyday tasks are readily available to end consumers. For example, mobile users can retrieve Web content using handheld devices since content retrieval is well-supported by infrastructure services such as transformational proxies. However, the same type of support is lacking for input-centric devices, those that create content and allow users to share content. This lack of infrastructural support makes input-centric devices hard to use and less useful.

The Appliance Data Services project seeks to explore a vision of an appliance computing world where users move data seamlessly among various devices. Based on this vision, we formulate three principles that guide the design of an architecture that helps realize this vision: bring devices to the forefront, minimize the number of device features, and place functionality in the network infrastructure. We evaluate our implementation of the ADS architecture based on these principles, and build applications using the ADS framework to evaluate the ease with which appliance computing applications can be built using the framework. We find that it is relatively simple to build and extend applications on ADS that make using digital devices easier, and results in the devices themselves becoming more useful.

* Internal Accession Date Only
Approved for External Publication
[1] Stanford University, Stanford, CA 94305

Making Computers Disappear: Appliance Data Services

Andrew C. Huang
ach@cs.stanford.edu

Benjamin C. Ling
bling@cs.stanford.edu

John Barton
John_Barton@hpl.hp.com

Armando Fox
fox@cs.stanford.edu

January 25, 2001

Abstract

Digital appliances designed to simplify everyday tasks are readily available to end consumers. For example, mobile users can retrieve Web content using handheld devices since content retrieval is well-supported by infrastructure services such as transformational proxies. However, the same type of support is lacking for input-centric devices, those that create content and allow users to share content. This lack of infrastructural support makes input-centric devices hard to use and less useful.

The Appliance Data Services project seeks to explore a vision of an appliance computing world where users move data seamlessly among various devices. Based on this vision, we formulate three principles that guide the design of an architecture that helps realize this vision: bring devices to the forefront, minimize the number of device features, and place functionality in the network infrastructure. We evaluate our implementation of the ADS architecture based on these principles, and build applications using the ADS framework to evaluate the ease with which appliance computing applications can be built using the framework. We find that it is relatively simple to build and extend applications on ADS that make using digital devices easier, and results in the devices themselves becoming more useful.

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. For example, digital camera users can save a trip to the film developing studio by creating prints on a home printer. Furthermore, digital camera users can perform tasks that are unimaginable with traditional cameras, like instantly displaying pictures on a digital photo frame across the country. These attributes are not true only of digital cameras. Other digital devices such as PDAs, digital audio recorders, and handheld scanners are designed to have similar time-saving and performance benefits.

Many tasks that can be performed using digital devices are made possible by infrastructure services. One example is handheld Web browsers; user-transparent transformation proxies allow unmodified Web pages to be viewed on these devices even though their small screens and low-bandwidth connections were not originally designed for Web browsing. Another example is TiVo™, a device that makes recording TV shows exceedingly simple by daily downloading TV broadcast information and recommends shows users are likely to enjoy by correlating all TiVo™ users' feedback on the shows they watch. The simplicity and power of these devices is possible because they are able to 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".

In the past, several factors prevented digital devices from gaining widespread popularity; one of these factors was high prices. Today, prices are dropping to a level where everyday consumers can afford to replace their traditional appliances with new, digital counterparts. Therefore, digital devices should enable all consumers to accomplish traditional and advanced tasks more easily and in a shorter amount of time.

However, in reality, users are often unable to accomplish even the simplest tasks with their new digital devices. 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.

This usability problem does not end with the device. To extract the data from these devices, users must often deal with installing, configuring, and learning how to use new software on their PCs or handheld devices. For example, when mobile users enter a collaborative setting where they want to share information, they often need to reconfigure the network settings on their mobile devices (e.g., laptop, PDA, etc.). In these situations, the users take steps to gain full network connectivity even when all they want to do is move data from a laptop to a shared wall monitor, or save meeting notes posted to a digital whiteboard onto their PDAs. To the experienced user, these steps may not be a problem; however, for everyday users, installing, configuring, and upgrading software, along with learning how to use the software itself can be extremely difficult.

One reason users must deal with PC software and go through numerous steps to accomplish even simple tasks is that input-centric devices, those that create digital information, lack infrastructure support. We hypothesize that the utility of input appliances and the ease with which users are able to operate them will be greatly improved if they too were infrastructure enabled.

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**. Thus, such devices are unable to achieve the status of "consumer devices" and the appliance computing world is still unrealized.

To address the deficiencies in appliance devices, we introduce the Appliance Data Services (ADS) project. The goal of ADS is to identify principles that make appliance computing possible and build a testbed for exploring the validity of those principles.

2 An Appliance Computing World

Typically, people express high-level tasks in terms of concrete artifacts (e.g., digital devices, Web pages, wall monitors, etc.) and the data residing on them: "I want to display the notes I've taken on my PDA on this wall monitor to share with the others in the meeting" or "I want to put this picture that I took using my digital camera on my Web page." Our vision, which describes an important aspect of an appliance computing world, is based on this observation:

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

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

1. making observations on attributes that are inherent in an appliance computing world;
2. identifying the principles that underlie these attributes; and
3. building a framework based on these principles to produce a testbed for exploring their effectiveness.

In the following subsections, we will use an extended example to help make our discussion of the appliance computing vision more concrete; this example will be based on the process of taking pictures and developing film. We wish to emphasize that our focus is on all digital devices, and not just cameras; using a concrete example simply makes the discussion easier.

2.1 Bring devices to the forefront

If appliance computing is to become a reality, the number of steps required to perform a high-level task should be few and the individual steps kept simple. A key observation that provides insight into making the steps simple is this: people find it easier to use concrete artifacts to move data. This observation suggests that an appliance computing world has the following attribute:

> **Attribute 1:** People move data using concrete artifacts.

Most people do not understand how a 35mm film camera works or how the film is developed; however, they have a mental model of pictures being "stored" on the film and being able to take the film to a studio to have photos produced from the film. In this example, the camera and the film are tangible artifacts that produce and store pictures.

In a digital world, tangible artifacts include digital cameras, flash memory cards, wall-mounted displays, and Web pages. In these examples, the users perceive the data as residing on artifacts that are concrete and permanent, which makes the tasks involving these artifacts easier to reason about.

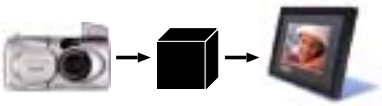
One reason digital devices are hard to use is that they force users to perform too many steps on objects they do not really understand. Today, to perform many basic tasks, users must think in terms of computers and their digital representation of the data. Extracting information from a digital device usually requires interacting with a PC. Furthermore, once the data is extracted, moving the data to the desired destination involves interacting with more computers. Continuing with our example, posting a picture on a Web page requires the user to know the server on which the page is hosted. As another example, placing notes on a wall monitor requires the user to know to which computer the monitor is attached.

The user not only has to deal with file locations, but possibly also file format conversions. File format conversions are even harder to reason about for some users who are unaware that files have different formats. One real-life example is posting images produced by an HP Capshare™ handheld scanner onto the Web. This particular scanner produces TIFF files; therefore, to view scans on the Web, the user must convert the scans into either JPEG or GIF. A user focused on Web publishing should not be required to fix the broken icon that results from trying to display the TIFF file on a Web page.

To allow people to focus on their tasks, the appliance they use should be sufficient to complete the task. For example, a mobile user who has taken notes on his Handspring Visor™ should be able to easily display his notes on any given digital display in a meeting room. It should not be necessary to use a secondary computer, like a PC, to complete the desired task:

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 [17]. It allows the user 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. To the user, everything that occurs between the artifacts is a "black box."



One consequence of this principle is that the traditional notion of a file should become invisible to the user; this means that a file's actual location and format must be hidden from everyday users. After all, using only simple, single-purpose data appliances, everyday users should not be expected to deal with a file system hierarchy or incompatible file formats. In practice, this means that file routing and format transformation are hidden from the user.

2.2 Keep devices simple

One approach some device manufacturers have taken to eliminating the experience of using a computer is to push functionality onto the devices. This observation is verified by the high-end cameras that have networking and image-editing capabilities. However, taking computers out of the picture by moving the computer into the device does not solve the problem. As Alan Cooper explains, such devices have not made tasks any easier. In fact, 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 easy to use if they are to be used by everyday consumers at all. In terms of our appliance computing vision, devices must 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, all the device requires is a mechanism for extracting the data from or placing data onto the device.

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

Besides aiding the usability issue, placing fewer features on devices helps address the mobile device problem of power consumption. This problem is so serious that some device vendors find it necessary to place sophisticated power consumption programs on their devices [2]. Simpler devices and with fewer computations reduce the power requirements of devices.

2.3 Place software in the infrastructure

Up to this point, we have focused on making the steps required to perform a high-level task simple. This has been accomplished through removing features from devices and eliminating the need to use general-purpose, hard-to-use devices, namely PCs. The question now 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.

So where does the functionality lie to perform the high-level tasks the users demand? Many tasks require file format conversion and data routing. More advanced tasks may require application-level data transformations (e.g., placing a drop-shadow on a picture) or the ability to coordinate data from multiple devices.

As this question suggests, there is a tension between the third attribute of our vision and the first two. We want to minimize the set of features placed on the device, but we want to allow users to control data movement with 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.

One possible location for the software that drives the high-level tasks 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. By "infrastructure" we refer to the deployed collection of hardware and software accessible directly or indirectly via an Internet (usually HTTP) programmatic interface. 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 the PCs and devices into the supporting infrastructure, has other added benefits. Having logically centralized software makes upgrades and administration much simpler. Furthermore, availability and reliability can more easily be reasoned about in a centralized approach. Finally, 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:

1. Bring devices to the forefront.
2. Keep the number of user-controllable features on devices to a minimum.
3. Place the 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, part of making ADS general is designing the framework such that it is amenable to new user interfaces and usage models. The same applies to issues such as authentication and security. While these areas of research are not actively explored, our design does not preclude the use of general solutions for these problems. Furthermore, where there are clear areas of exploration such as these, we intentionally add hooks in the ADS framework so that, for example, a new authentication module can be plugged in.

In the following sections, we describe the ADS framework architecture, its current implementation, and how the framework meets the three design principles outlined above.

3 The Architecture

Prior to entering a discussion of the ADS architecture, we first describe its basic data unit. The basic data unit is a triple, composed of a user identifier, the command to be executed, and the data that is to be operated on: (userid, command-tag, data). Although each component of the triple can be of an arbitrary type, it is simplest to begin by thinking of the userid and command-tag as text strings. The triple is used as the basic data unit for the following reasons:

1. Application selection: The command-tag names the high-level application that the user wants to perform on the data (e.g., "Send picture to my Web site"), using a binding mechanism described later. However, the command-tag alone is not sufficient to define the application since different users may have different meanings for the same tag, or result in different semantics in the interpretation of the tag (e.g. "my Web page" maps to different URLs for different users). Thus, a user identifier is required to fully specify the desired application.

2. Access control: The user identifier is also required to determine what credentials are to be attached to the application request. For example, a user should be allowed to push data into her own public_html directory, but not necessarily those of other users. Furthermore, some services may be accessible only to authorized users. Thus, the system needs some identifier to attach credentials to the request.

3. Other service features: Services that require a command-tag and user identifier include billing, security, and personalization. Although we have not investigated the implementation of such features, we have left the user identifier as a necessary "hook" for adding these capabilities later.

The remainder of this section describes the architectural components of the ADS framework shown in Figure 1. The architecture is split into three main components: Data Receive stage, Application Control stage, and Services Execution stage. Each stage corresponds to a high-level function that must be performed on the input data. In the following sections, the functional and architectural roles of each stage's subcomponents will be described.

3.1 Data Receive

In the Data Receive stage, data is taken from various devices, collected, and sent along once a complete *(userid, command-tag, data)* triple is received. The components in this stage, Access Point and Aggregator, interface with the devices the system supports to receive data and output the completed triple, which the ADS system can operate on.

3.1.1 Access Point

The Access Point consists of necessary hardware and software to receive data from appliances. Examples of hardware are IR transceivers, RF basestations, or cradles or cables for "docking" an appliance. The software is made up of device adaptors, which allow the Access Point to communicate with devices speaking a particular protocol. The Access Point can be implemented as a commodity PC outfitted with the appropriate hardware interfaces or it can be designed as a special-purpose "network appliance."

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

Role: Deals with device heterogeneity.

We make dealing with device heterogeneity simpler by isolating this concern in this one architectural component. Isolating device heterogeneity to the Access Point 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 functional role of the Access Point involves selecting the appropriate protocol for communicating with a particular device and communicating with the device to receive its data.

The key challenge in designing the Access Point is extensibility. This requirement arises from the lack of standardization among device vendors and the increase in the variety of devices being introduced. One who believes standardization is right around the corner needs only look at the continued existence of operating system device drivers. Furthermore, since general-purpose devices are often difficult to use for everyday consumers, the trend towards single-purpose devices is likely to continue; this only increases the number of devices ADS must support. As long as vendors continue to use their own protocols for the new devices that come out, building a usable system means supporting a sufficient number of the devices available. Thus, it is crucial to make adding support for new devices simple.

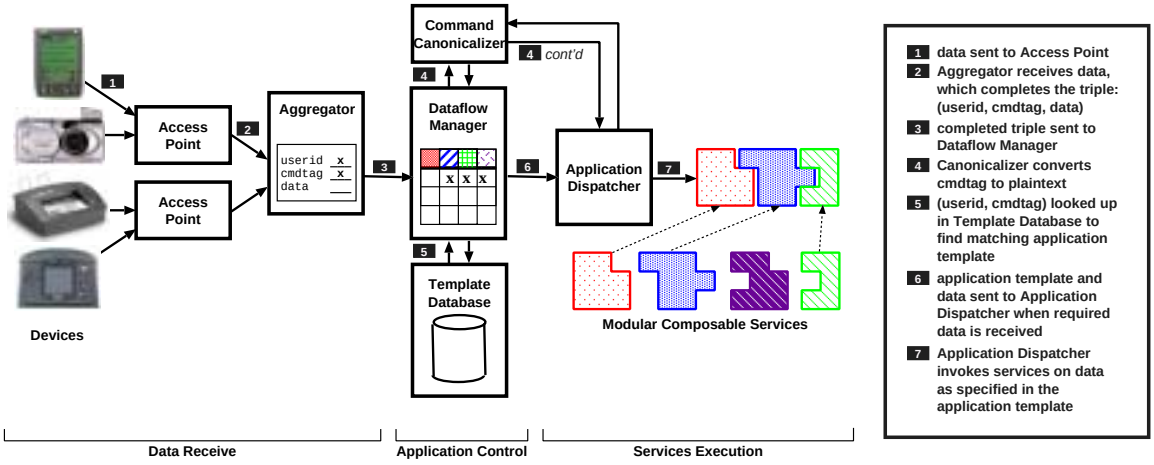


Figure 1: The ADS Architecture

3.1.2 Aggregator

One way to simplify adding support for new devices is to make the Access Point stateless. A stateless Access Point makes device adaptors easier to code because adaptor writers do not have to worry about state management. Furthermore, adaptors that are stateless can more easily be made robust since they can arbitrarily restarted.

The Aggregator manages the state in the Data Receive stage so that the Access Point can be made stateless. Thus, the architectural role of the Aggregator is to make the Access Point more extensible:

> **Role:** Simplifies adding support for new devices and protocols.

The functional role of the Aggregator is to gather data sent from the Access Point, and send the data off once all pieces of the triple (userid, command-tag, data) are received. 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 received from the Data Receive stage are used to determine the desired application. The data portion of the triple is then added to the list of parameters required for the selected application. Once all the parameters for a given application are filled, all the data collected is sent on to the Services Execution stage.

3.2.1 Command Canonicalizer

The Command Canonicalizer facilitates designing devices with "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 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, records the desired command-tag "send to my Web site," and pushes a button to send the data and command-tag into an Access Point. 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.

3.2.2 Template Database

The canonicalized command-tag 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.

Another benefit of using the Template Database to bind command-tags to templates is that it provides a level of indirection between application selection and application specification. This level of indirection serves to separate the concerns of users of ADS applications and creators of ADS application templates. The database provides an easy way for third-party developers to make their templates available to ADS users. Furthermore, with the proper authentication mechanisms in place, a third-party template provider can easily and effectively restrict access to the templates it has developed.

For example, say Kodak™ wants to make a set of ADS applications available to consumers who purchase a Kodak™ camera. If the set of templates is 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 cameras made by different vendors. With the Template Database, the user's act of registering gives that user the credentials to view and select the ADS application templates Kodak™ has developed. Furthermore, upgrading applications or adding new ones is simple since the application provider needs only modify or add templates stored in the Template Database rather than ship upgrades to all subscribers.

Finally, application templates give a large degree of flexibility to ADS users. While basic users simply select from a set of templates provided by the device vendor, advanced users can customize applications by modifying or creating their own application templates. In doing so, advanced users can select the exact services and parameters to be invoked on their data. At the same time, these users do not have the burden of dealing with the protocols required to route and transform the data among the various services used.

3.2.3 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.

4 Current Implementation

In this section, we describe the current implementation status of the architecture as described in the previous section. We begin by discussing the implementation details of the Access Point, followed by the remaining components. The reason the Access Point is discussed separately from the remaining components is that it is the only component that does not logically reside in the centralized network infrastructure. Instead, Access Points are likely to be deployed as publicly-accessible Web kiosks or as appliances within people's homes. The framework implementation discussion will be followed by a description of two applications built on the framework.

During the discussion of the implementation, it is helpful to note that existing technologies (e.g., HTTP, XML, etc.) are used in ADS whenever possible. In general, these technologies allow us to leverage current or prior work so that we can focus on the research agenda of ADS.

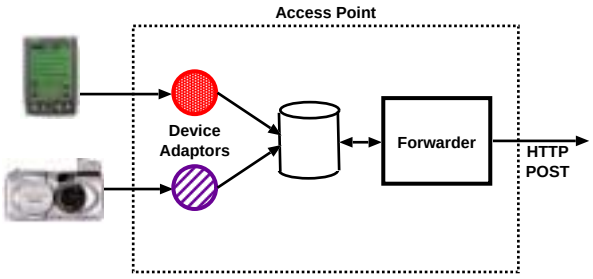


Figure 2: Access Point implementation

4.1 Access Point

As stated in the architecture section, the main challenge in designing the Access Point is extensibility; i.e., making it easy to support new devices. The Aggregator plays an architectural role in facilitating extensibility by allowing the Access Point to be stateless. In the implementation, we go a step further by making each device adaptor autonomous.

4.1.1 Device Adaptors

In the Access Point implementation shown in Figure 2, autonomous, stateless device adaptors interact directly with the particular devices or protocols they were designed for. The Forwarder then has the job of forwarding data received by the adaptors on to the Aggregator. Autonomy in the adaptors is achieved by eliminating the programmatic interface between the adaptors and Forwarder, and using the file system as the communication channel. Using such a simple interface is possible because a device adaptor need only notify the Forwarder whenever it receives new data. Hence, the Forwarder need only poll a shared directory for new files.

Using stateless, autonomous device adaptors that are only required to write data to the file system greatly simplifies the task of writing new device adaptors. Adding a device adaptor to ADS does not require any knowledge about the structure or APIs of ADS.

4.1.2 Forwarder

When the Forwarder detects a new file in the shared directory, it sends the data to the Aggregator using the POST method of HTTP. The POST request includes any meta-information available along with an ADS-specific *x-data-class* header. This header specifies the type of data being sent; i.e., userid, command-tag, or data. Each of these components can arrive at any time and in any order; it is the Aggregator's job to keep track of session state (i.e., current userid and command-tag) for each Access Point.

Using HTTP POST as the API between the Access Point and the Aggregator provides more flexibility than a strongly typed API. First of all, the HTTP POST API allows the Access Point to send any metadata it receives from the device without knowing what information the Aggregator can handle. Secondly, this API facilitates backward compatibility. New Access Points send newly defined parameters in headers or multipart MIME, while old ones continue sending the same information as before. Thus, only the Aggregator, rather than all Access Points, needs to be changed to handle both the new and old APIs.

The Access Point as described here is stateless and configuration-less, which makes it appealing to deploy as a publicly available kiosk, home data appliance, or Web-centric service.

4.1.3 Status

The current implementation of the Access Point runs on an IR-equipped, commodity PC running Windows 2000™. Device adaptors for the following devices have been implemented:

- Palm™, WinCE™, and other devices that support the IR-FTP protocol *(IR)*
- HP CapShare™ handheld scanner, HP digital cameras, and other devices that support the HP JetSend™ IR protocol *(IR)*
- CardScan™ business card scanner *(parallel port or USB)*

4.2 Infrastructure Components

The infrastructure components run on one or more Linux PCs and are built on the Ninja service framework developed at UC Berkeley [8]. The research goals of Ninja include providing a framework in which its services automatically gain the desirable system attributes of reliability, fault tolerance, and scalability. Ninja also provides services commonly found in service frameworks such as service discovery, as well as other useful services such as an XML database and distributed data structures with transactional properties. Using Ninja allows us to focus on building the system rather than the system issues that are important for all large-scale systems, but that have orthogonal solutions [6].

Each component runs as a Ninja service that can be independently replicated and restarted. The

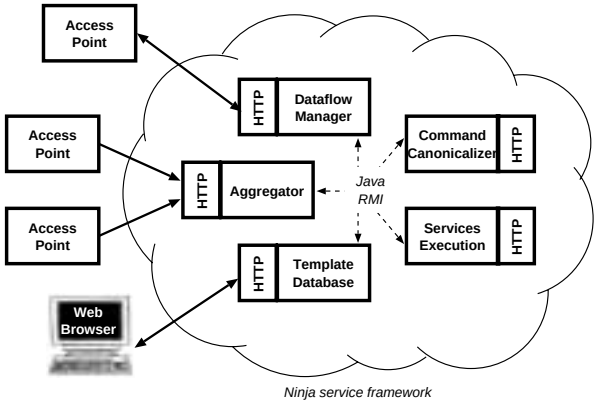


Figure 3: Infrastructure implementation

components communicate with each other using Java RMI, which eases the programming of these distributed services. Furthermore, Ninja places minimal constraints on its services; all that is required is that services inherit from the `iSpaceService` class and implement `init()` and `destroy()` methods.

Each of the components also exports an HTTP interface to the outside world, which makes it possible to directly access information about each component. For example, accessing the Template Database via a Web browser allows the user to view and edit command-tags mappings and template specifications. Web access to the Dataflow Manager allows the user to view the previously input data (e.g., pictures from a digital camera) and enter the data that is required to complete the chosen high-level task (e.g., text descriptions entered via a Web form). We choose HTTP as the external communication protocol because Web browsers are available on nearly every computer and HTTP has become a de facto standard for communication on the Internet.

The high-level organization as described above is shown in Figure 3. The following sections describe the implementation status of each individual component in the infrastructure.

4.2.1 Aggregator

As discussed in the architecture section, the Aggregator manages state so that Access Points can remain stateless. Since the session state of the Data Receive Stage is made up elements of the basic data unit, the Aggregator stores the current userid, command-tag, and data for every Access Point. Thus, each Access Point is assigned a unique ID so that the Aggregator can keep track of the data associated with each Access Point. Once a complete triple is received, the Aggregator sends the triple on and clears the data field. Clearing the userid and command-tag occurs when the user "logs out," which causes the Access Point to send explicit "clear" commands for each field.

4.2.2 Dataflow Manager

The Dataflow Manager stores a table for each (userid, command-tag) pair. The columns of the table correspond to the parameters required by the application specified by the (userid, command-tag) pair. Each row of the table corresponds to a set of parameters, or *data bundle* pending application execution. When a data bundle is complete, the Dataflow Manager sends the complete bundle and template to the Services Execution stage.

The number of outstanding data bundles allowed is specified in the template using the *maxbundles* tag. For example, if $maxbundles = 1$, a piece of data sent by the user overwrites previously sent data if it occupies the same column. However, if $maxbundles = -1$, which corresponds to unlimited bundles being allowed, for any data that arrives that does not have an empty slot in an existing bundle, a new bundle is created.

4.2.3 Command Canonicalizer

Currently, for most practical purposes, command-tags are specified via a Web form interface. Thus, ADS receives most command-tags in canonicalized form (i.e., plaintext). However, the current implementation of the Canonicalizer also has the ability to handle a WAV file command-tag. The service that converts speech to text is a Ninja service originally written for UC Berkeley's Iceberg project [16]; the service utilizes IBM's ViaVoice™ SDK [9] to perform the conversion.

Our experience using spoken command-tags is limited since it requires using an extra device (i.e., digital audio recorder). Since current digital devices do not have embedded audio recorders, it is easier to select command-tags from the Web interface. However, we have found that the main issue in using speech-to-text converters is that the WAV file must be high-quality and have low noise. Thus, it is unlikely that recorders embedded on digital devices will be used to select command-tags using current technology. One way to address this problem without requiring high-quality microphones to be placed in devices is to limit the vocabulary of the speech-to-text conversion dictionary to set of valid command-tags.

4.2.4 Template Database

The current implementation of the Template Database uses to file system to enable simple (userid, command-tag) lookups. Templates are stored in files named by command-tag under a directory named by the userid. Thus, template lookup is performed by following the path: `<userid>/<command-tag>.xml`. In the future, an XML database such as the Ninja framework's XSet [18] may be used to enable queries on the information within the templates.

The templates themselves are represented using XML. XML is a natural choice for representing structured data because of its widespread use, which makes utilities such as XML parsers and XML databases readily available. Thus, representing our structured data using XML allows us to leverage the work of others, and allows our work to be used by others.

Application templates fully describe ADS applications; it specifies the data the application requires and the services to invoke on the data. Therefore, building a new application on the ADS framework using existing services simply involves creating a new template. The general format of templates is shown in Figure4.

In our own development and testing, we have found it useful to use macros that describe commonly-used functions. For example, rather than including the service description of the `StoreService` in every template, we place a reference to another XML file that contains that description. The process of building applications can be simplified even further by creating a template editor application. Such an application would present a graphical user interface that allows its user to choose services and set service parameters.

4.2.5 Services Execution

The Application Dispatcher reads the application template and invokes each service on the corresponding data in turn. Each service returns a receipt that contains an optional return data value and service execution information (e.g., "success," errors, etc.). Although the it is not used except by ADS developers, the receipt will eventually be used by the Dispatcher can convey status and error information back to the user.

Currently, the following services are available for use in building applications:

- convert – uses the UNIX command 'convert' to convert between arbitrary image formats
- scaling – scales images by a user-specified amount
- thumbnail – re-sizes images to thumbnail size
- speech-to-text – converts WAV files into text strings
- store – stores files on the local drive
- HTML publishing – publishes an application's completed HTML page to a publicly accessible page on Geocities™

As mentioned before, creating new services under the Ninja framework is a relatively low-overhead task. Thus, the programmer can focus on the service logic rather than on distributed systems or framework requirement issues.

4.3 Applications

In this section, we describe two appliance computing applications that exhibit various aspects of the ADS framework. The application descriptions are followed by a discussion of how these applications were built using the ADS framework. These sections show that:

1. appliance computing applications greatly simplify the user-experience of using digital devices;
2. ADS can be used to build applications that coordinate data from many devices; and
3. building and evolving appliance computing applications on top of the ADS framework is far simpler than applications built from scratch.

4.3.1 Web Photo Album

The Web Photo Album application exhibits the potential of ADS for making digital devices true consumer devices by simplifying the user-experience. In this application, the user conducts the following steps to create a Web-based photo album using a digital camera (note that the figures show rudimentary Web interfaces for this application; they could easily be extended, but have been kept basic for simplicity and ease of deployment):

1. take pictures with a digital camera

Template Format

```
<template>
  <data type="[mimetype of input1]">
    <action desc="[output desc]">
      <command>
        <service>[service name]</service>
        <method>[method name]</method>
        <arglist>
          <arg type="[arg1 type]">[arg1]</arg>
          <arg type="[arg2 type]">[arg2]</arg>
          ...
        </arglist>
        <return>[return mimetype]</return>
      </command>
    </action>
  </data>

  <data type="[mimetype of input2]">
    ...
  </data>
</template>
```

data1 — services to invoke

data2

Sample Template

```
<template>
  <data type="image/jpeg">
    <action desc="photo">
      <command>
        <service>ads.ScalingService</service>
        <method>convert</method>
        <arglist>
          <arg type="byte[]">$6</arg>
          <arg type="String">image/jpeg</arg>
          <arg type="String">50%</arg>
        </arglist>
        <return>image/jpeg</return>
      </command>
      <command>
        <service>ads.StoreService</service>
        <method>store</method>
        <arglist>
          <arg type="byte[]">$6</arg>
        </arglist>
      </command>
    </action>
  </data>
</template>
```

produces a thumbnail

stores the file

Figure 4: Application Template Format

2. visit a Web page to login and select the photo album application

3. transfer the pictures by pointing the camera's IR transceiver to an IR-equipped Access Point

4. enter descriptions of each photo via a Web page that displays the pictures received by ADS

5. click "add to photo album" and view the published page

Although the steps include using a Web browser, the interfaces are exceedingly simple (e.g., login, select command, etc.). These simple interfaces allow the pages to be served up by a kiosk-style, touch screen Access Point so that the user-experience is very similar to that of using an ATM machine. Thus, during the entire process, the user never directly interacts with a "PC." In contrast, without such an end-to-end application, the user must deal with a computer and one of its applications in each

of the steps ADS performs:

1. extract photos from the digital camera *(file system)*
2. create thumbnails of the photos *(image editing program)*
3. upload the thumbnails and pictures to the Web server hosting the Web-based photo album *(ftp program)*
4. edit HTML to add the thumbnails to the page, descriptions of each photo, and links to the actual picture *(text editor)*

These steps may be easy for a computer-savvy user and are simplified by online photo album applications. However, for consumers that expect to buy a digital camera and produce pictures as easily as they would with a non-digital camera, these steps and the learning required to master them is unacceptable. Only an end-to-end application solves the problem for the average consumer.

An extension to this application that saves time even for expert users involves adding an image-editing service. Often, when publishing pictures on the Web, people perform a set of image manipulations on every picture. These manipulations may include adding a drop-shadow, increasing the contrast, or decreasing the brightness. Adding a customizable image-editing service to the Web Photo Album application eliminates this time-consuming task. Thus, applications like these simplify tasks and save time when using digital devices.

4.3.2 Guestbook

The Guestbook application shows how the ADS framework can be used in a appliance computing world where people have a variety of personal and infrastructure-attached devices available. This application exhibits the framework's flexibility in dealing with a variety of different devices. An example scenario involves a kiosk placed at the entrance of conference. In this scenario, attendees register by adding their information to the ADS guestbook:

1. the attendee looks into a Webcam mounted on the kiosk and presses a button to take a picture
2. the application requests the attendee's business card and displays the following options:
   - type your contact information
   - place your business card in the business card scanner
   - beam your electronic business card using a Palm™ or WinCE™ device
3. the attendee verifies that the picture looks right and the business card information is correct, then clicks a button to add the entry to the conference attendee list

5 Evaluation

We evaluate the implementation of the framework using two criteria: how well the implementation meets the stated principles of our appliance computing vision and how easy it is for developers to build applications on the framework. To make steps towards realizing our vision, we identify three principles, which are met framework implementation in the following ways:

1. We bring devices to the forefront so that people can focus on using concrete artifacts for moving data. The Template Database plays a major role in accomplishing this by separating application specification and selection so that users can perform high-level tasks by simply specifying a command-tag. Meanwhile, the extensible Access Point makes it possible to support a variety of devices so that a large number of devices can be used to input data and select command-tags.
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 adding a lot of extra features to the device. Meanwhile, the Template Database helps minimize device configuration by allowing application creation and customization to be done independently of the user's devices.
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, as much functionality as possible is placed in the Internet infrastructure to free the user from software issues (e.g., installations, upgrades, and configurations), provide a more reliable service, and leverage existing Internet services.

Not only does the ADS framework fulfill the requirements of our appliance computing vision, it eases the development and evolution of applications that make appliance computing a reality. First, creating new ADS service modules is a relatively low-overhead task because of the services Ninja provides and the small number of requirements it places on its service. Secondly, creating new ADS applications using XML templates is simpler than building standalone applications from scratch. Finally, modifying existing ADS applications and extending them to support new devices and services is simpler than doing so for standalone versions of the same applications.

A real-life example that shows the ease with which ADS application behaviors can be modified arose when we decided to replace actual photos with thumbnailed links to the actual photos in the Web Photo Album. Changing the application's behavior simply required adding a few lines to the template to invoke the `ThumbnailService` and `StoreService`. Also, since the application is Web-based, a few lines needed to be added to the dynamic HTML file that displays the pictures in the photo album.

Another real-life example of the advantage of having a general appliance computing framework involves adding support for new devices. When we added support for the CardScan™ business card scanner, all current and future applications that use a picture instantly benefited. Without any modifications to the template, the Web Photo Album accepted the TIFF files, converted them to JPEG, and correctly displayed the scans on the resulting page. It was also easy to create a new application based on the Web Photo Album that was optimized for the image attributes (size and B/W) of scanned images. Modifying the Guestbook to accept scans of business cards as an alternative to text business cards sent from PDAs required that a few lines be added to the template. In standalone versions of these applications, it is likely that support for the handheld scanner would have to be added to both independently; this is especially true since these two applications would have been developed by two different parties.

Our experience with the CardScan™ business card scanner has shown that adding support for new devices is simple as well. The simple file system API between device adaptors and the Access Point Forwarder made it possible for a new project member to immediately write adaptors for ADS. This student did not have to learn anything about the rest of the ADS framework and could focus solely on the logic of interfacing with the scanner. Thus, it took very little effort to add business card scanner support for all current and future ADS applications.

Figure 5: A meeting at the Stanford Interactive Workspaces Room

Thus, in our experience, 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: people use simple artifacts to move data around in a variety of ways.

6 Research Agenda

This section describes three areas of research we intend to explore in the immediate future.

6.1 Integrating more infrastructure services

One of the advantages of taking an infrastructure-centric approach to designing the ADS framework is that we are able to leverage existing infrastructure services. Currently, ADS takes advantage of the Geocities™ Web page hosting service by publishing users' Web-based photo albums and guest books to Geocities. With the wealth of Internet services such as Geocities available, we intend to continue our exploration in using these services as building blocks in the modular services framework.

Immediate plans involve implementing a digital photo frame posting service that will enable users to post pictures to a Ceiva™ or Storybox™ photo frame from anywhere in the world. Currently, the Storybox service involves many of the same steps as posting pictures to a Web photo album. By adding a Storybox service, we hope to make posting pictures to a remote digital photo frame as simple as it is to post pictures to a Web page using ADS.

6.2 Exploration into "Smart Space" environments

The Interactive Workspaces Project at Stanford [7] explores new possibilities for people to work collaboratively in "meeting rooms of the future" using a variety of computing and interaction devices of many scales. These devices include laptops, PDAs, wall-mounted SmartBoard™ displays, etc. We intend to use this technology-rich space to explore ways ADS can be deployed in "Smart Space" environments. Furthermore, since the Stanford Interactive Workspaces Room (IW-Room), shown in Figure 5, is in production use for regularly scheduled meetings, the room and its users will serve as a testbed for ADS applications.

Our initial efforts involve installing a production version of the ADS framework in the IW-Room. The first application that will be deployed is the ADS Guestbook application. A kiosk at the entryway will act as the Access Point for the application so that all visitors to the room can input their business card information and have their pictures taken upon entering.

Future areas of exploration involve using the IW-Room's endpoints (e.g. wall-mounted displays) and devices to build other ADS applications that support sharing of data in this collaborative environment. For example, an IR-dongle placed below a wall-mounted display might have a fixed ADS application associated with it that results in the injected data being displayed. This application exercises the modular services framework to a point where dynamic, on-the-fly service composition is needed to invoke the appropriate application to display the data; Paths is a project at Stanford designed to do exactly that [11]. Currently, a production version of Paths, which includes an interface to make Paths appear like an ADS service, is being installed in the IW-Room.

As people begin to use ADS and Paths for day-to-day operations in the IW-Room, we expect to gain insight on new applications and deficiencies in the architecture that need to be addressed.

6.3 Conveying status and error information

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. Even when ADS is used casually among members of the project team, this issue has already begun to arise. 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 a non-traditional approach must be taken to convey status and error information to the user.

7 Related Work

The Portolano Project [3] at the University of Washington shares a similar goal of exploring ways in which appliance computing can be made a reality. They too recognize that it is necessary to shift the focus away from general-purpose computers and towards simple, easy-to-use devices. Furthermore, they take an infrastructure-centric design approach and aim to build a general framework on top of which many appliance computing applications can be built. The main difference between ADS and Portolano is in the scopes of the projects. Certain Portolano research issues, such as user interfaces, service deployment, and resource discovery, are beyond the scope of ADS or are provided by the service framework we have chosen.

At the Sony Computer Science Laboratory, there are a number of projects that explore how transferring data between devices can be simplified. The Pick-and-Drop [14] and InfoStick [10] projects are based on the observation that users find it easier to deal with concrete artifacts than with abstract objects like files. This system allows users to move objects between multiple displays using a pen-like device to provide the illusion that the pen can manipulate physical objects. The Augmented Surfaces [15] project addresses the need for users to share data from their mobile devices with the preinstalled devices in the environment. The project explores how the user's mobile devices can be integrated into the current computing environment.

Much of the work on post-PC devices has focused on accessing the Internet from data appliances such as PDAs, cell phones, and palmtop computers. Some examples include transformational proxies [6, 5, 13] and wireless protocol gateways [12, 4], both of which enable these devices to leverage the enormous installed infrastructure of servers, content, and interactive services. Conversely, and partially as a result of infrastructure enablement, the Internet has begun to adapt to these devices and we are now seeing services tailored for their use, such as Yahoo Mobile™ and a variety of sites that feature "Palm-friendly" pages in addition to their desktop content.

8 Conclusion

While the devices required for an appliance computing world exist, users often cannot even perform the simplest operations with current devices. These de-

vices and the supporting infrastructure are simply too difficult to use. Our vision for appliance computing is a world in which everyday users move data seamlessly and effortlessly among handheld and infrastructure-attached artifacts. As is the goal of this project, we identified three principles for realizing this vision and implemented a testbed for affirming their validity. Applications built on the ADS framework showed that these principles are helpful steps in realizing our appliance computing vision.

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] Mike Esler, Jeffrey Hightower, Tom Anderson, and Gaetano Borrielo. Next century challenges: Data-centric networking for invisible computing. In *Fifth ACM Conference on Mobile Computing and Networking (MobiCom 99)*, Seattle, WA, USA, August 1999.

[4] WAP Forum. Wireless application protocol (WAP) forum. `http://www.wapforum.org`.

[5] Armando Fox, Ian Goldberg, Steven D. Gribble, Anthony Polito, and David C. Lee. Experience with Top Gun Wingman: A proxy-based graphical web browser for the Palm Pilot PDA. In *IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, Lake District, UK, September 15–18 1998.

[6] Armando Fox, Steven D. Gribble, Yatin Chawathe, Eric A. Brewer, and Paul Gauthier. Cluster-Based Scalable Network Services. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP-16)*, St.-Malo, France, October 1997.

[7] 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.

[8] Steven D. Gribble, Matt Welsh, Rob von Behren, Eric A. Brewer, David Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Joseph, R.H. Katz, Z.M. Mao, S. Ross, and B. Zhao. The ninja architecture for robust internet-scale systems and services. In *To Appear, Special Issue of Computer Networks on Pervasive Computing*.

[9] IBM. Viavoice sdk. `http://www-4.ibm.com/software/speech/dev/sdk_linux.html`.

[10] Naohiko Khotake, Jun Rekimoto, and Yuichiro Anzai. InfoStick: An interaction device for inter-appliance computing. In *Handheld and Ubiquitous Computing (HUC'99), First International Symposium*, Karlsruhe, Germany, September 1999.

[11] Emre Kiciman and Armando Fox. Using dynamic mediation to integrate cots entities in a ubiquitous computing environment. In *Handheld and Ubiquitous Computing (HUC 2000), First International Symposium*, September 2000.

[12] Metricom Corp. Ricochet Wireless Modem, 1998. `http://www.ricochet.net`.

[13] ProxiNet, Inc. ProxiWeb Thin Client Web Browser, 1998. `http://www.proxinet.com`.

[14] Jun Rekimoto. A multiple device approach for supporting whiteboard-based interactions. In *CHI'98 Proceedings*, Los Angeles, CA, USA, April 1998.

[15] Jun Rekimoto and Masanori Saitoh. Augmented surfaces: A spatially continuous work space for hybrid computing environments. In *CHI'99 Proceedings*, Pittsburg, PA, USA, May 1999.

[16] Helen J. Wang, Bhaskaran Raman, Chen nee Chuah, Rahul Biswas, Ramakrishna Gummadi, Barbara Hohlt, Xia Hong, Emre Kiciman, Zhuoqing Mao, Jimmy S. Shih, Lakshminarayanan Subramanian, Ben Y. Zhao, Anthony D. Joseph, and Randy H. Katz. Iceberg: An internet-core network architecture for integrated communications. In *IEEE Personal Communications: Special Issue on IP-based Mobile Telecommunication Networks*, 2000.

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

[18] Ben Y. Zhao. The xset xml search engine and xbench xml query benchmark. Master's thesis, University of California, Berkeley, May 2000.