The Live Web Series

Introducing Forever

Personal Cloud Application Architectures

Phillip J. Windley, Ph.D.,
Founder & Chief Technology Officer, *Kynetx, Inc.*

May 2013

The Live Web Series

***The Live Web Series** sets forth a vision for the future of the Internet and our interactions on it. This paper is the fourth paper in that series. We recommend reading other papers in the Live Web series (http://www.windley.com/liveweb/) as background for this paper. A companion paper, Picos and Personal Clouds (http://www.windley.com/liveweb/picos) more completely describes the technology behind the Forever application discussed in this paper.*

Executive Summary

Forever is an evergreen address book. Imagine an address book where the contact information was always in sync. No matter how often your friend changes their profile, your address book is always up-to-date. That's the promise of Forever.

Forever is an *unhosted app*, meaning it does not have a database to store user data. Forever succeeds because it is based on a brand-new architecture and programming model that doesn't store any user data in the app itself.

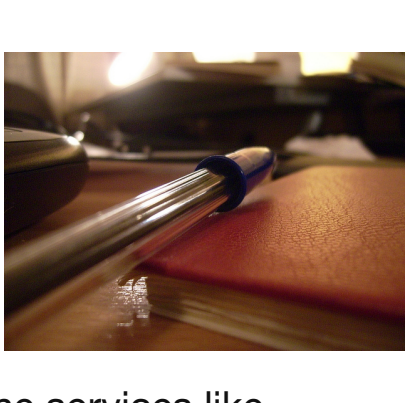
Forever accesses data about its users by linking to their personal clouds. Rather than storing user data in the app, Forever makes use of data and links that are stored in the user's personal cloud. Personal clouds are user-controlled, online systems that can connect to other personal clouds, forming a peer-to-peer network.

Forever is DNS for people. Whenever Forever needs profile information about one of your contacts, it pulls that information directly from their personal cloud. This may sound inefficient, but it's exactly how the domain name service (DNS) works. The difference is that in this case, Forever applies that architecture to a network of people, rather than a network of machines.

Forever is easier to build because it uses personal clouds. The lack of a back-end database makes Forever and apps like it easier for developers to build and maintain. Personal clouds provide a convenient platform for developers and their applications.

Forever's architecture provides significant benefits to people. Forever's architecture provides timely, up-to-date contact information to users. In addition, Forever's architecture supports what's called privacy by design, giving users more control over their data and how it's used.

Introduction



People have been writing down addresses for friends, acquaintances, and others for as long as the concept of an address has existed. But addresses rot. The longer an address has been in your address book, the more likely it is to be wrong. And the Internet and mobile has only made the problem worse.

Electronic address books help by making it easier to update the information. But they don't know when the address has changed. Online services like LinkedIn and Facebook make it easier to keep up to date, but our address books are still frequently wrong or stuck in the wrong place when we need them.

The Internet itself used to have this problem. Domain names need to be mapped to IP addresses. People used to keep their own copy of all the domain names they needed and the IP address that name mapped to in a big address book called `/etc/hosts`. Keeping the hosts file up to date was a major chore for system administrators. But we don't do that anymore.

The answer was a system called "domain name service" or DNS. DNS created a network service that allowed individual administrators to maintain the mapping for the systems they ran and anyone else to pull the information whenever they needed it so that it was *always up to date.*

This paper introduces an address book application called *Forever*. Forever is built using a novel architecture and programming model that allows Forever to function as a DNS for people. Forever is an unhosted app, meaning that it doesn't have a database of it's own. Rather it relies on an online, standards-based service to store user data. Specifically, Forever links to a personal cloud running the CloudOS (http://www.windley.com/liveweb/cloudos/) from Kynetx[1]. The personal cloud not only stores the user's profile information, it also stores channels from the user's cloud to their friends.

When a Forever user asks to see their contacts or update their profile, they are really using data stored in their personal cloud. When they add a new friend to their contacts or delete an entry, they are actually changing the channels that their personal cloud has to the personal clouds of their friends. When the app needs to contact the user, it does so using the personal cloud's notification service.

The unhosted model supports other applications linking to the user's personal cloud. Those applications see changes made by Forever and Forever sees changes they make. What's more, because user profile information is always only stored in their own cloud and pulled as needed, in the same way that DNS resolves domain names, the contact information it presents to users is always up to date.

Forever has a simple value proposition: users always have access to their friend's most recent contact information. In fact, a user could change her phone number, manually or automatically, based on the time of day and people using Forever to call her would get the number she is at *right now.*

> Forever is like domain name service for people.

But Forever is really an app designed to show the power and value of personal cloud architectures. Kynetx CloudOS provides a great platform for building unhosted app. Building applications on top of CloudOS offers significant benefits:

- Users are in control of their data in a personal cloud that they own. The unhosted app, combined with a personal cloud based on Kynetx CloudOS creates a scenario that is *private by design rather than private by agreement.* That is, rather than relying on terms of service or privacy statements to assure the user that their data will be protected, the app never stores it and the data is kept under the user's control.
- Multiple apps can access the data in the personal cloud—with the user's permission—enabling network effects. Rather than the data being in yet another silo, the data is always available to the user, regardless of the apps they have installed.
- Unhosted apps based on Kynetx CloudOS are easier to build and deploy than stand alone applications. The CloudOS provides services for storing and retrieving personal data, managing and using connections between clouds, and interacting with the owner of the personal cloud. All of the services provided by CloudOS are available for the app developer.

The remainder of this paper provides further details on personal cloud application architectures.

Modular Web Applications

Ever since the Web got CGI and cookies, web apps have used a programming model like the one shown in Figure 1. In the traditional model, the application presents a Web interface to application data stored and managed in concert with the application.

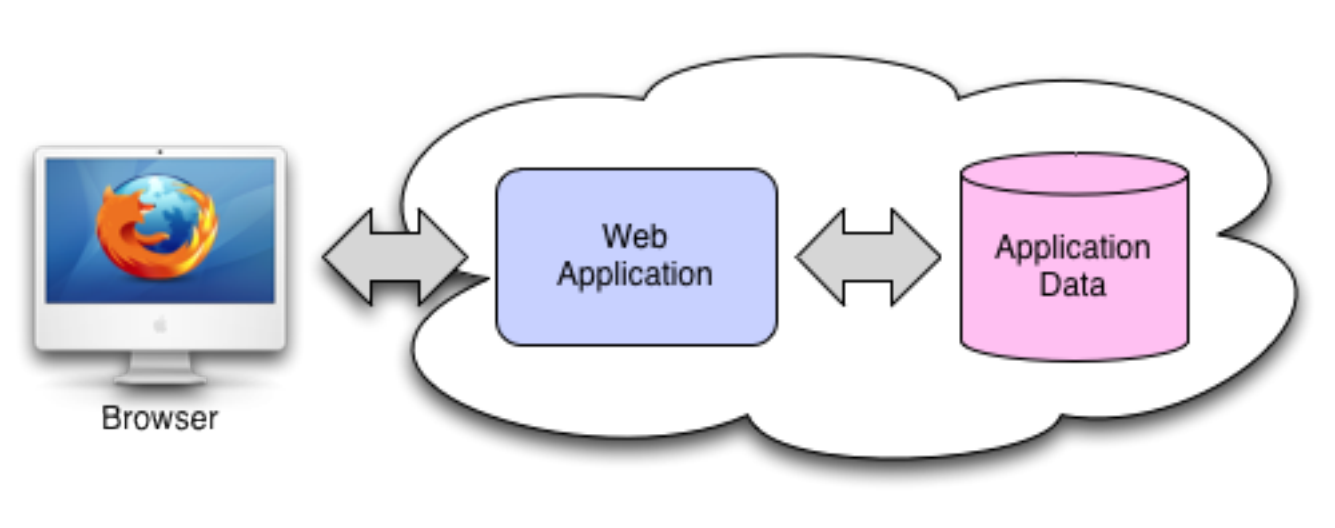


Figure 1: Traditional Web Architecture

The traditional Web application model has given us a wealth of exciting and useful online applications. While the details of how the application is structured differ greatly from app to app, the basic architecture wherein the application mediates access to data has not changed.

But the traditional architecture of Figure 1 has significant limitations. Storing data for users in the app's datastore creates "data silos" where the app has exclusive access to the data. This results in inconveniences to the user:

- users must fill out profile and other standard information for every application they use.
- applications that die take the user's data with them.
- more significantly, data silos prevent network effects for data where one app's data exhaust becomes the input for another app.
- the app necessarily has access to all of the user data it mediates, creating privacy concerns.

In contrast, a modular Web architecture provides for independent ownership and hosting of the data portions of the application and its business logic and presentation layers. For example, an unhosted app[2], use a architecture where the browser, not the Web application, mediates the data as shown in Figure 2.

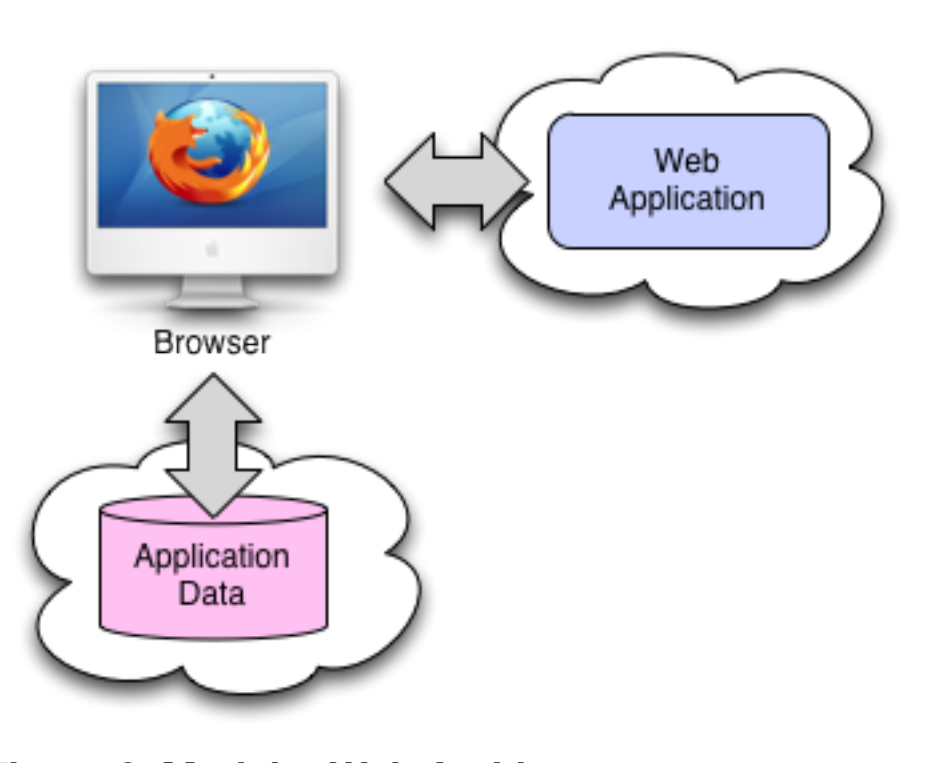


Figure 2: Modular Web Architecture

Web apps built so that data and application can be treated independently free developers from hosting and managing the data while giving the user more choice and greater control. Unhosted architectures are enabled by modern browser support for HTML5, CSS, and JavaScript. The browser receives the app's source code over HTTP and connects to the data using an API. Unhosted app architecture separates the app hosting from the data hosting making the entire system more modular and more loosely coupled.

> Modular Web architectures allow for independent ownership and hosting of data and applications.

Of course, it's not just about apps written in JavaScript. Any Web application written in any language can *and should* be built using an architecture that allows for separate ownership and control of personal data. Even mobile apps could reference data from a data source under their user's control rather than storing in their own proprietary silo.

Forever is built as an unhosted app—meaning that there is no server-side code or data store. When you got to Forevr.us (http://forevr.us) , your browser downloads the HTML, CSS, and JavaScript that make up the application. The user interface and business logic (the code that defines how the app works) are all there, running in your browser. But Forever won't function unless it is linked to a personal cloud because it has no back end infrastructure of its own for storing user data.

Personal Clouds

If unhosted apps don't store the data themselves, where is it stored? In the user's personal cloud.

Personal clouds take many forms. On one end of the spectrum as simple file stores like Dropbox or Apple's iCloud. At the other end are more sophisticated personal servers that not only store data, but also run applications and link to other personal clouds.

The personal cloud technology that makes Forever possible is the Kynetx CloudOS. CloudOS provides a personal cloud back end with the following capabilities:

- CloudOS uses OAuth so that people can link applications to their personal cloud.
- CloudOS provides a personal data store for storing, updating, and retrieving data that the app might need.
- CloudOS manages connections to other personal clouds, creating a network of decentralized, protocol-mediated clouds.
- CloudOS has a notification service that the app can use to interact with the cloud's owner.
- CloudOS is extensible by developers to provide other services that their app might need.

While other personal cloud technologies provide the first two in varying degrees, CloudOS is unique in providing the remaining features. Forever makes great use of the connections between personal clouds. Indeed that is where the real work of Forever gets done. Because CloudOS manages the connections, Forever doesn't have to store that information. What's more, there is significant business logic around initiating, using, and tearing down connections between clouds that CloudOS manages for the Forever application.

> Personal clouds based on CloudOS provide owners with independent data management and app services.

The combination of personal clouds with ideas from Unhosted apps result is a brand new application architecture with personal clouds providing important services for applications.

At present, Forever uses the SquareTag personal cloud system, but any backend that runs CloudOS would work as well. In the near future, Forever will support multiple personal cloud providers.

Personal Channels

One of the most important features of the Kynetx CloudOS is its built-in support for personal channels. Personal channels are described in detail in an an earlier white paper in the Live Web series (http://www.windley.com/liveweb/pchan) . This section is adapted from that paper. Please refer to the full paper to understand the full benefit list of personal channels.

When they first appeared in the early 1980 s, personal computers were powerful tools in their own right. That changed in the 90 s with the emergence of widespread network connectivity. Nowadays, a PC that s not connected to the Internet is non-functional for many of the tasks that people perform every day (e.g., email, web browsing, social networking). To test that assumption, just try turning off the network on your computer for a day. And of course, the very newest personal computer—the smartphone—makes connectivity the very foundation of the platform.

Even more so than personal computers, personal clouds are only interesting when they are connected. The connection between two personal clouds—or between a personal cloud and anything else it is connected to is called a **personal channel**. The network of people and organizations linked via personal channels is called a **relationship network**.

On an open standard relationship network, the attributes, permissions, and capabilities of a relationship are standardized and extensible. Every relationship is a **link**. A link may be a simple one-way (asymmetric) subscriber relationship that does not require involvement of the second party, or it may be a stronger two-way (symmetric) relationship in which both parties may act as publishers and subscribers.

In either case, when data and messages can flow in one or both directions across a link, it is a **channel**. The control each party has over the channel—the terms and conditions to which they agree about how it will work—is called a **link contract**. Figure 3 shows two personal clouds connected via a channel controlled with a link contract. Note that both parties store a copy of the same link contract just like they would a paper contract.

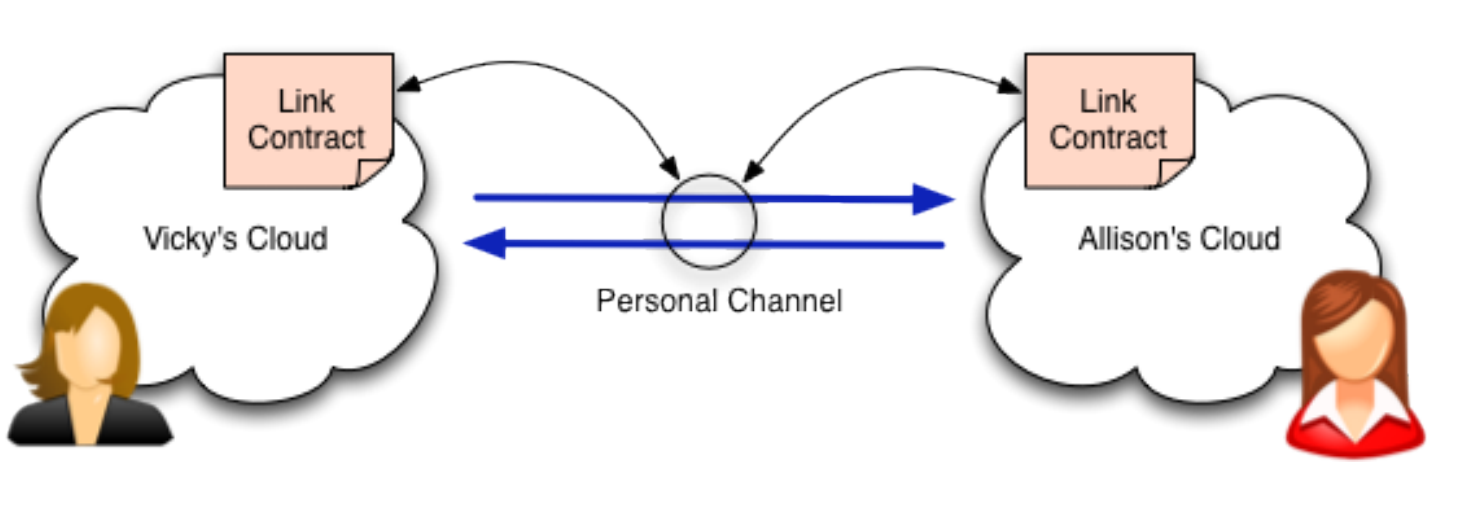


Figure 3: Every personal channel is controlled by a link contract.

Like email, personal channels all speak the same protocol, forming a point-to-point network between personal clouds. However, unlike an email server, whose sole function is usually email processing, a personal cloud is more like a general-purpose computer in the cloud, i.e., it has an operating system that runs applications, processes events, and manages data on behalf of and under the direct control of its owner. So personal channels can be much smarter communications links than ordinary email or text messaging.

> Channels give personal cloud owners control over how, when, and with who data is shared.

- A personal cloud may have any number of inbound and outbound channels. And two personal clouds may share multiple channels for different purposes.
- Personal channels ensure accountability for shared data because the authority to share can be granted, modified, and revoked on a channel-by-channel basis.
- Link contracts are a flexible means of declaring fine-grained access control to data and services. Link contracts specify the nature and behavior of a channel.
- Channels may pass any type of message between personal clouds, not just text, images, and attachments. Messages may include event notifications, data queries, and data transfers. Messages may also be orchestrated into workflows.

In sum, personal channels on an open-standard relationship web can be dramatically more useful to individuals and businesses than ordinary email or Web connections. Forever makes use of personal channels by using them as the conduits over which permissioned access to profile information for the user's contacts occurs.

Pull, not Push

The channels between personal clouds are the magic behind Forever. Without a network of personal clouds operating as peers, we lose vital aspects of the architecture. Forever depends on two important principles:

1. Users are responsible for maintaining their own profile data.
2. When a user needs contact information, her personal cloud *pulls the information from her friend's cloud.*

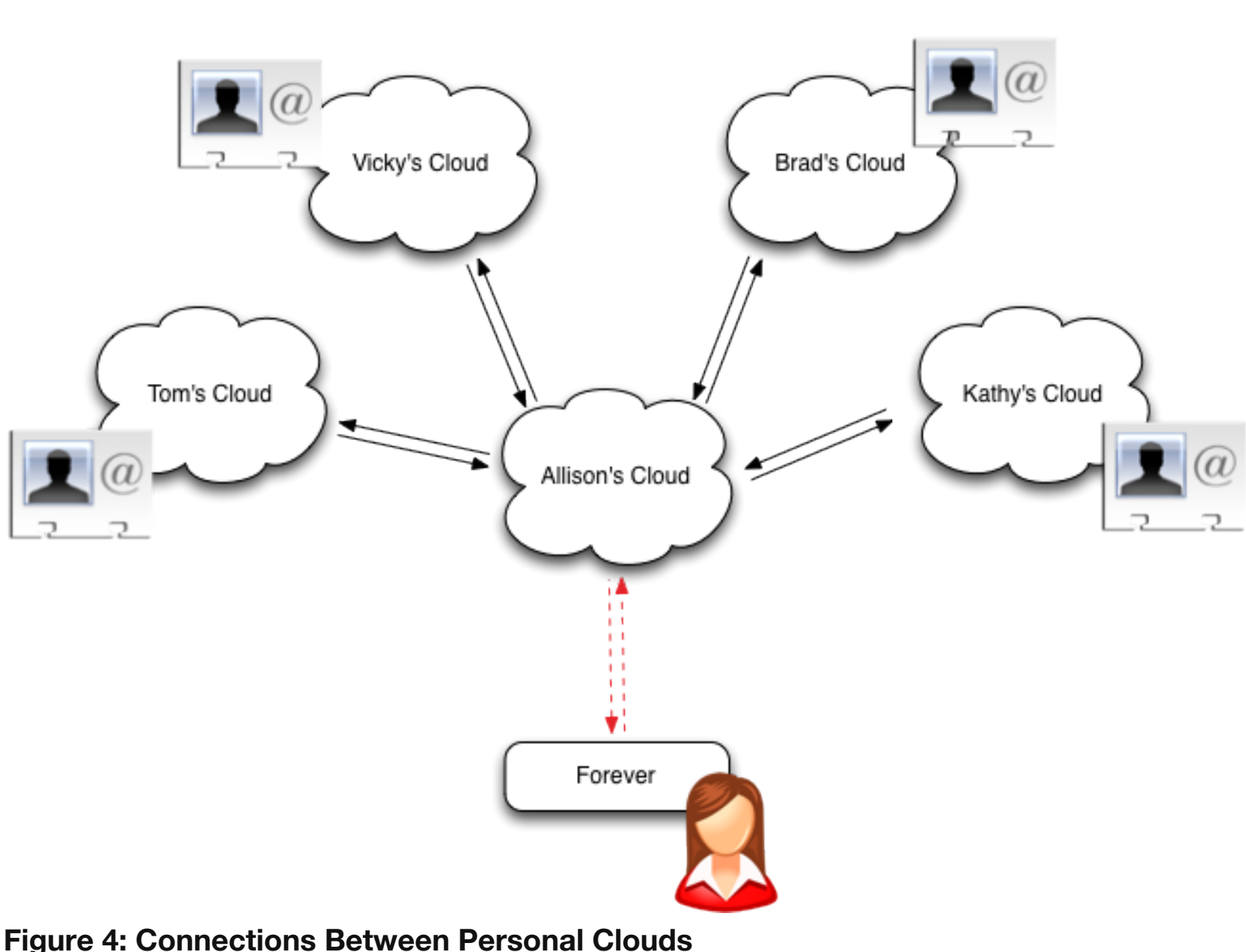


Figure 4: Connections Between Personal Clouds

To understand how Forever achieves these principles, consider the following scenario and Figure 4.

1. Allison uses Forever to manage and use the connections in her personal cloud.
2. Forever links to Allison's personal cloud. Forever *does not have links* to the other clouds (at least not on Allison's behalf—although those users may use Forever as well).
3. When Allison asks Forever to show her Vicky's contact information, it sends that request to Allison's personal cloud.
4. Allison's cloud requests updated contact information from Vicky's cloud and returns it to Forever for display.

Note that Allison's cloud manages the interactions with her friends' clouds. Whether Allison is requesting contact information, establishing a connecting, or deleting one, her cloud is responsible for intermediating the request. This has important privacy and security implications because it puts Allison in control of the data and connections in her cloud. She isn't relying on policy or code in Forever.

> Forever's design pulls contact information from the person's cloud just when it is needed.

Of course, Allison's cloud might cache data for efficiency purposes—as does DNS—but she sees contact data maintained by its owner.

This architecture presents Allison and her friends significant choice and freedom. While Allison is using Forever, her contacts might use other contact management applications and yet they will all work together because personal clouds and channels present a standard way of connecting. This is exactly the model that IMAP and SMTP represent for email. Two people can exchange email regardless of their choice of email client or email provider[3].

Talk to the Cloud

We have been discussing Forever as if all of the connections are of equal weight and trustworthiness. That needn't be the case. Suppose that Allison has connections to friends and to some companies she does business with as shown in Figure 5.

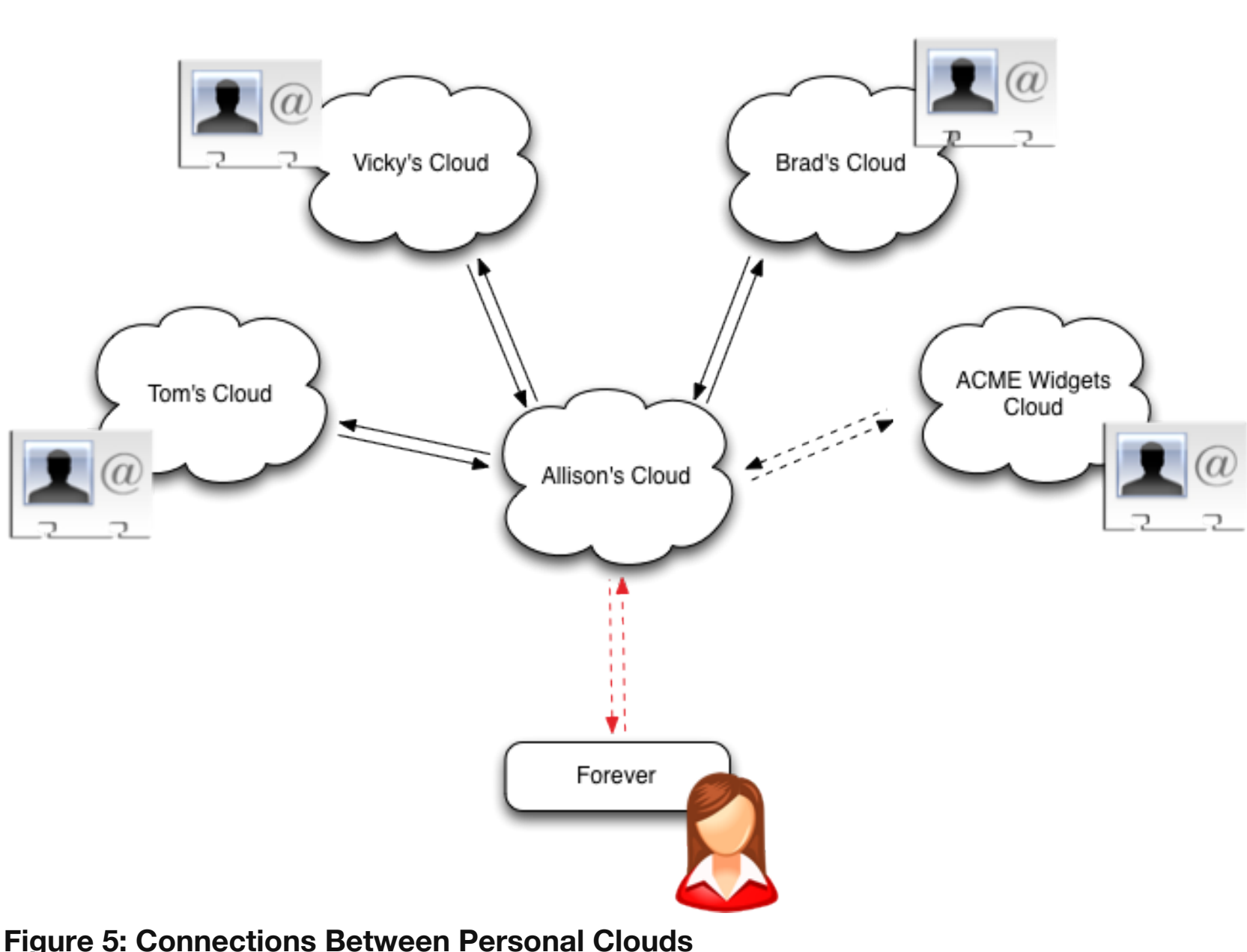


Figure 5: Connections Between Personal Clouds

Note that one of Allison's connections is *ACME Widgets*. The channel between Allison and ACME Widgets is shown as a dotted lined to indicate that it is different from the others. In this case, Allison has marked this channel as a "vendor" relation rather than "friend".

This distinction allows her cloud, by policy, to treat requests for contact from ACME Widgets differently than requests from her friends. For example, she may not allow ACME to pull her contact information from her cloud, but only allow them to contact her through an in-cloud notification, a feature built into CloudOS. Later, if she severs the connection to ACME Widgets by deleting the channel, they will not be able to contact her.

> Channels provide owners control of their personal clouds.

On the other hand, they might show themselves to be a valued and trustworthy partner and request access to certain information in Allison's personal cloud, perhaps even her browsing history, driving history, or some other information that will help them serve her better. Allison can grant increased access.

One of the great benefit of connections based on personal channels is that the owner of the cloud is ultimately in control of how connections work and what data is shared. This needn't be a management burden, these changes in policy can be made by the system in response to rules in Allison's cloud or gestures she makes in application interactions.

Conclusion

Forever is designed to illustrate the architecture and benefits of unhosted applications running against a personal cloud infrastructure based on Kynetx CloudOS. Forever is significant because it uses services from personal clouds beyond mere data storage.

Developer Benefits The unhosted application architecture provides significant benefits to developers.

1. Developers no longer have to build and maintain the data and service back ends necessary to make their apps work.
2. Development time is reduced because applications are simpler.
3. CloudOS provides services beyond data storage such as personal channels that developers can access and use.
4. CloudOS gives developers an extensible platform that they can mold to suit the needs of their application.

Because unhosted applications are more modular than monolithic Web 2.0 applications, they create an ecosystem where apps and data can be combined in new ways to create more powerful applications. For example, the Web couldn't exist without DNS and the inventors of DNS did not anticipate the Web. Modular, protocol-based solutions have a power that is much greater than the individual apps.

User Benefits Personal clouds provides significant benefits to users.

1. Users are in control of their data and how it is used. This results in increased privacy.
2. Personal clouds break down data silos so that one source of data can be used by different applications.
3. Users have more freedom over the apps they use because the ultimate source of the data is their personal cloud.

The model that personal clouds and unhosted applications create is familiar because it resembles the application and data ownership models of the personal computer. At the same time, it brings significant benefits of the Web because apps and data are hosted.

Forever is just one of many applications that can run on CloudOS. We invite developers to work with us to build other compelling applications.

Finding Out More

Forever is available as an unhosted app at Forevr.us (http://forevr.us) . The code is open source (http://github.com/kynetx/Forever) (MIT License).

You can discover more information about the concepts and technologies we've discussed from a variety of sources including the Phil Windley's blog, Technometria (http://www.windley.com) , the Live Web white paper series (http://www.windley.com/liveweb/) and Phil Windley's book The Live Web (http://www.amazon.com/exec/obidos/ASIN/1133686680/windleyofente-20) .

If you're interested in creating picos and personal clouds, the Kinetic Rules Engine is open source (https://github.com/kre/Kinetic-Rules-Engine/) . However, the easiest way to get started is using the online service provided by Kynetx. You can try out personal clouds and the KRL programming model for free by creating an account at SquareTag.com (http://squaretag.com) . SquareTag accounts are free and you can develop multiple applications and run them without charge for non-commercial use. Examples and documentation (http://developer.kynetx.com) are available online.

Endnotes

1. The Kynetx CloudOS is based on the open source KRE project hosted on Github (https://github.com/kre/Kinetic-Rules-Engine) .
2. You can read about unhosted apps and alternative back ends at the unhosted.org Web site (https://unhosted.org/) .
3. For more on this analogy of personal clouds and email, see IMAP as the Proto Personal Cloud (http://www.windley.com/archives/2013/04/imap_as_the_proto_personal_cloud.shtml) .

About the Author

Phillip J. Windley is the Founder and Chief Technology Officer of Kynetx. Kynetx is a personal cloud vendor, providing the underlying technology for creating, programming, and using persistent compute objects, or picos, using KRL. He is also an Adjunct Professor of Computer Science at Brigham Young University where he teaches courses on reputation, digital identity, large-scale system design, and programming languages. Phil writes the popular Technometria blog (http://www.windley.com) and is a frequent contributor to various