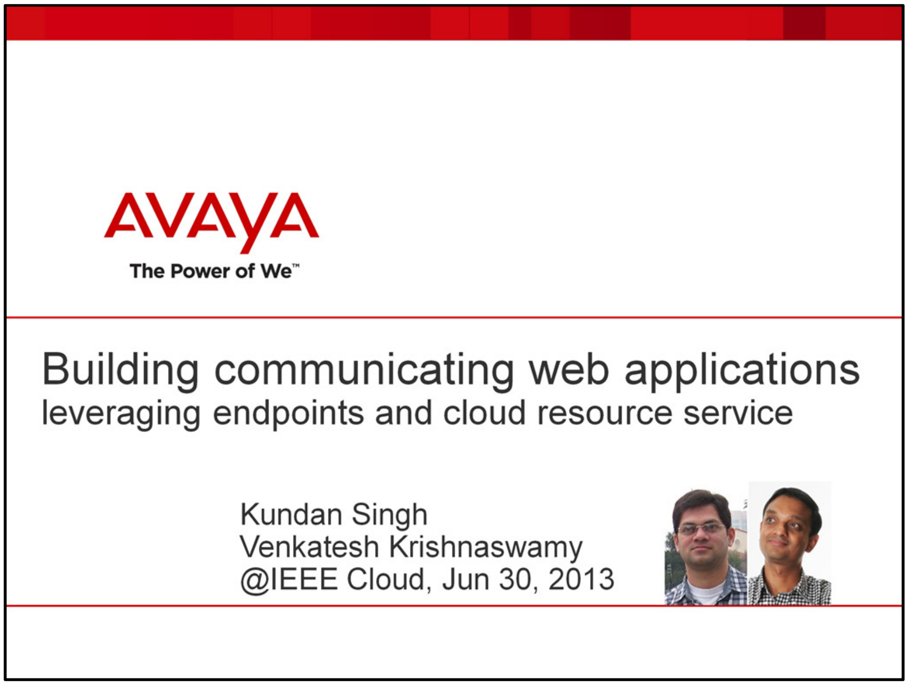


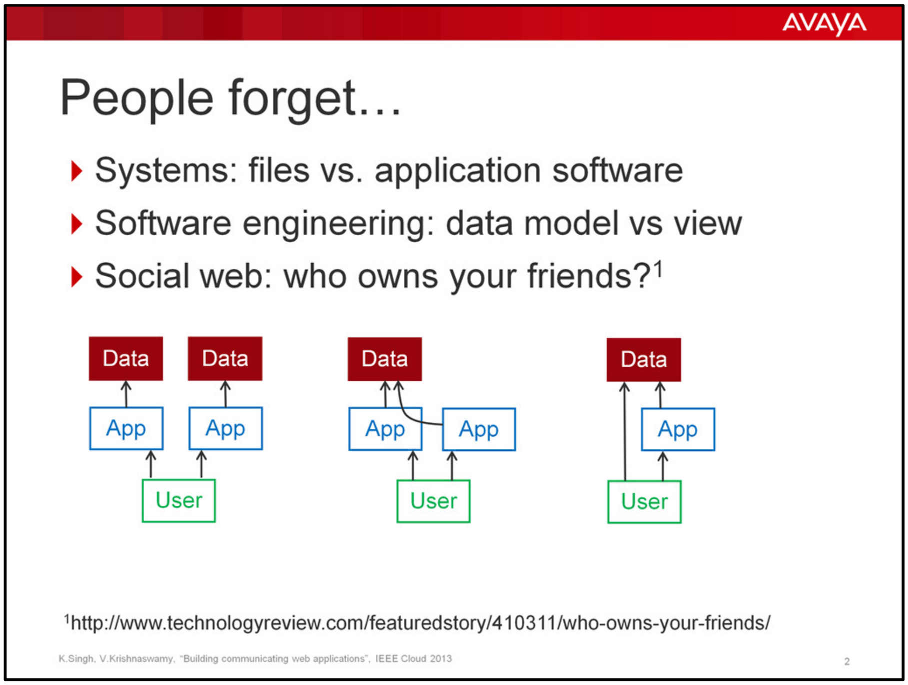
AVAYA

The Power of We™

Building communicating web applications
leveraging endpoints and cloud resource service

Kundan Singh
Venkatesh Krishnaswamy
@IEEE Cloud, Jun 30, 2013

Hello everyone. My name is Kundan Singh and today I will describe a project we did at Avaya Labs.



People forget…

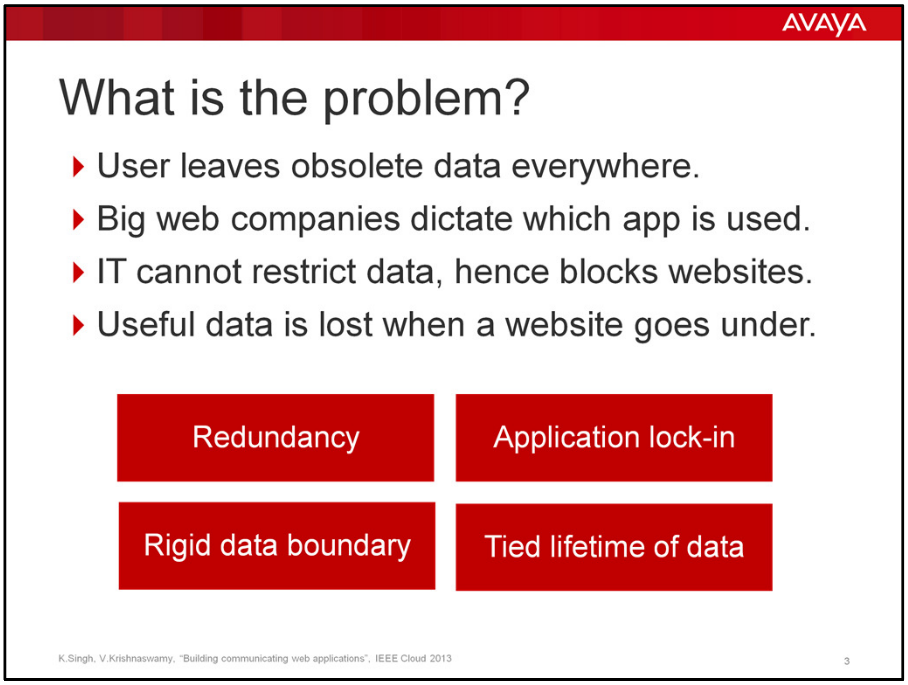
- Systems: files vs. application software
- Software engineering: data model vs view
- Social web: who owns your friends?[1]

[1] http://www.technologyreview.com/featuredstory/410311/who-owns-your-friends/

K.Singh, V.Krishnaswamy, "Building communicating web applications", IEEE Cloud 2013

Let me start by saying that people often forget the importance of separating data from the application logic. Social websites often manage and control the user’s data such as his connections. Even though the data belongs to the user, it is often difficult for him to use it on another website. Who really owns your friends?

If another application needs to access the user data controlled by the first application, it must follow the custom API provided by the first application, as shown in the second diagram. An obvious solution shown in the third diagram is to let the data be controlled by the user, who gives permission to individual applications to access the data on his behalf. Many people have tried to create such socially aware cloud storage to be shared by websites.



AVAYA

What is the problem?

- User leaves obsolete data everywhere.
- Big web companies dictate which app is used.
- IT cannot restrict data, hence blocks websites.
- Useful data is lost when a website goes under.

Redundancy

Application lock-in

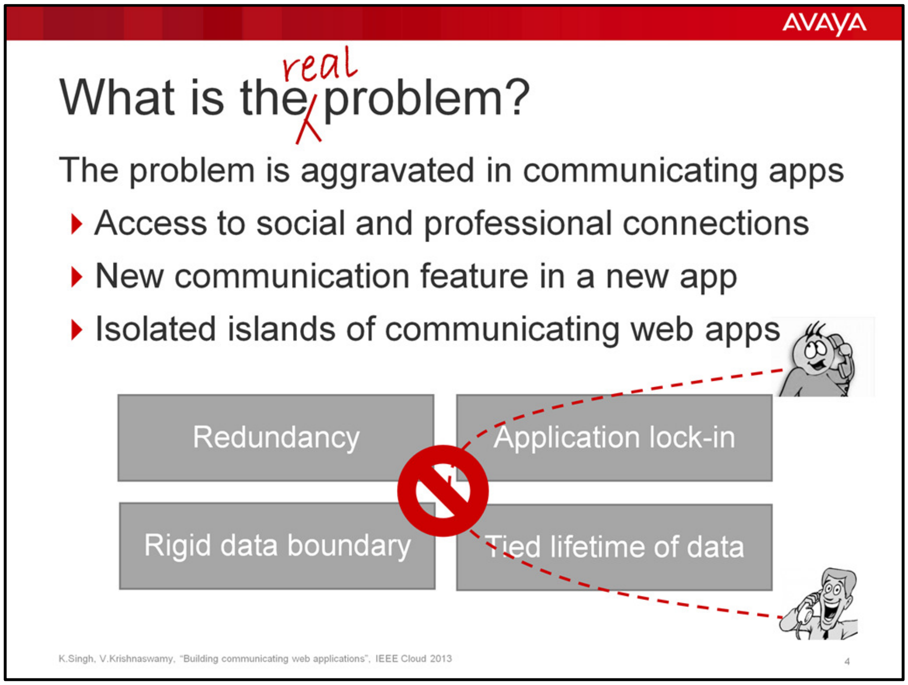
Rigid data boundary

Tied lifetime of data

K.Singh, V.Krishnaswamy, "Building communicating web applications", IEEE Cloud 2013

3

Such trend by social websites poses several problems as shown here. For example, user often leaves obsolete data around because modifying redundant information at many different social websites is cumbersome. Typically, the big website that has most number of users, dictates what web applications their user can use, even if a better or more feature rich application exists elsewhere. In an enterprise network, the IT would like to keep the social data and interactions private within the organization – some way to bind the social website to a private enterprise database while the user accesses the website from the enterprise network. Finally, the lifetime of the data gets tied to the application. For example, when many people moved from friendster or myspace to facebook, they had to pretty much recreate their social graph and profile.



AVAYA

What is the real problem?

The problem is aggravated in communicating apps

- Access to social and professional connections
- New communication feature in a new app
- Isolated islands of communicating web apps

Redundancy

Application lock-in

Rigid data boundary

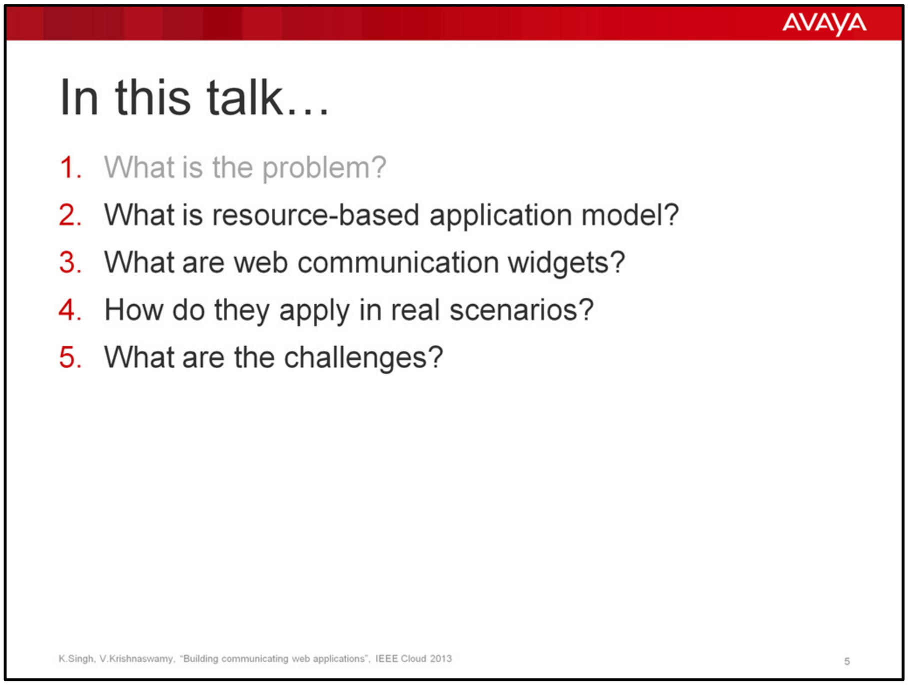
Tied lifetime of data

K.Singh, V.Krishnaswamy, "Building communicating web applications", IEEE Cloud 2013

4

These problems are aggravated when dealing with communicating applications – those that allow you to interact with others in real-time (and sometimes asynchronously). For example, one would like to be able to reuse their social connections to call out from different websites.

With the emergence of web-based communication technologies such as WebRTC (or web real-time communications) it has become relatively easier for anyone to create communicating web applications – but the problem is every website wants to define its own way of call control and session establishment, creating islands of non-interoperable web applications.



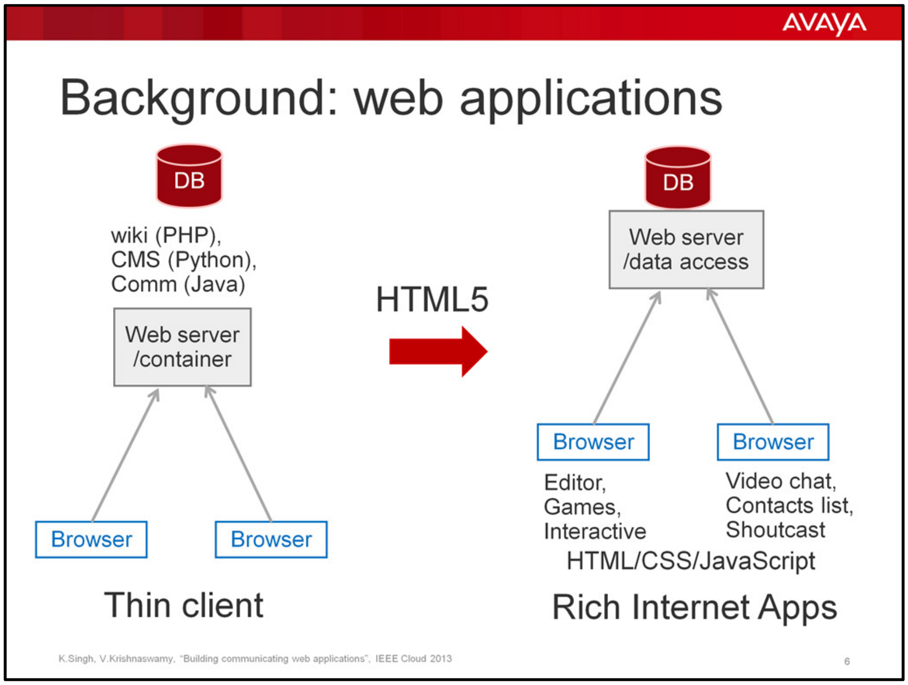
AVAYA

In this talk…

1. What is the problem?
2. What is resource-based application model?
3. What are web communication widgets?
4. How do they apply in real scenarios?
5. What are the challenges?

K.Singh, V.Krishnaswamy, “Building communicating web applications”, IEEE Cloud 2013

In this presentation, I will describe the our project that aims to solve these problems. This is a brief agenda for the talk.



Background: web applications

DB

wiki (PHP),
CMS (Python),
Comm (Java)

Web server
/container

Browser

Browser

Thin client

HTML5

DB

Web server
/data access

Browser

Browser

Editor,
Games,
Interactive

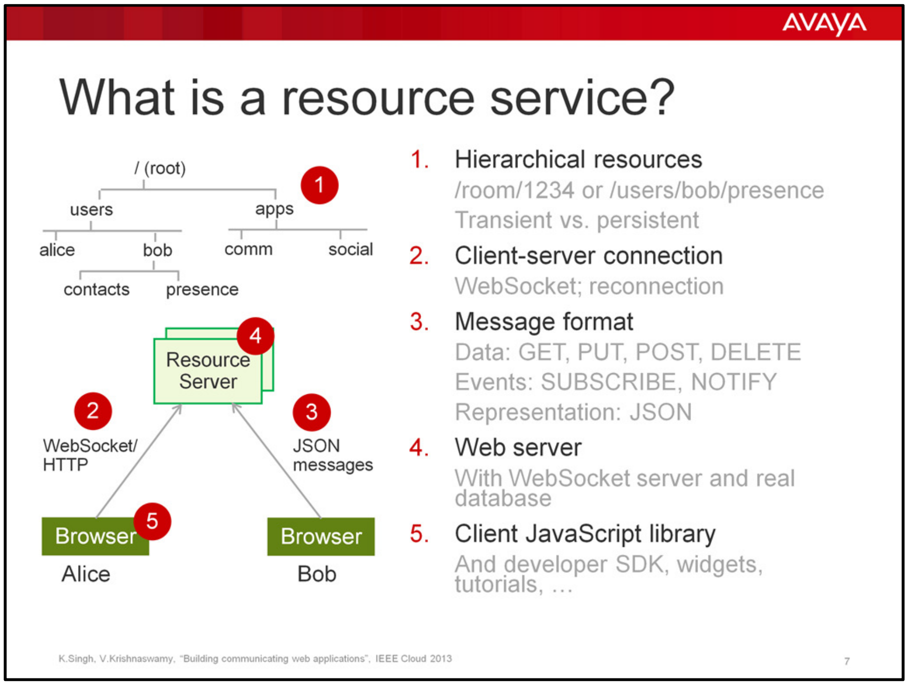
Video chat,
Contacts list,
Shoutcast

HTML/CSS/JavaScript

Rich Internet Apps

K.Singh, V.Krishnaswamy, "Building communicating web applications", IEEE Cloud 2013

Let us take a quick look at the background of web applications. With the feature rich HTML5 browsers both on desktop and mobile, many applications are moving their logic from the backend webserver to the browser. The concept of rich internet apps is not novel, but can be taken to the extreme in the resource-based application model.



What is a resource service?

1. Hierarchical resources
   /room/1234 or /users/bob/presence
   Transient vs. persistent
2. Client-server connection
   WebSocket; reconnection
3. Message format
   Data: GET, PUT, POST, DELETE
   Events: SUBSCRIBE, NOTIFY
   Representation: JSON
4. Web server
   With WebSocket server and real database
5. Client JavaScript library
   And developer SDK, widgets, tutorials, …

K.Singh, V.Krishnaswamy, "Building communicating web applications", IEEE Cloud 2013

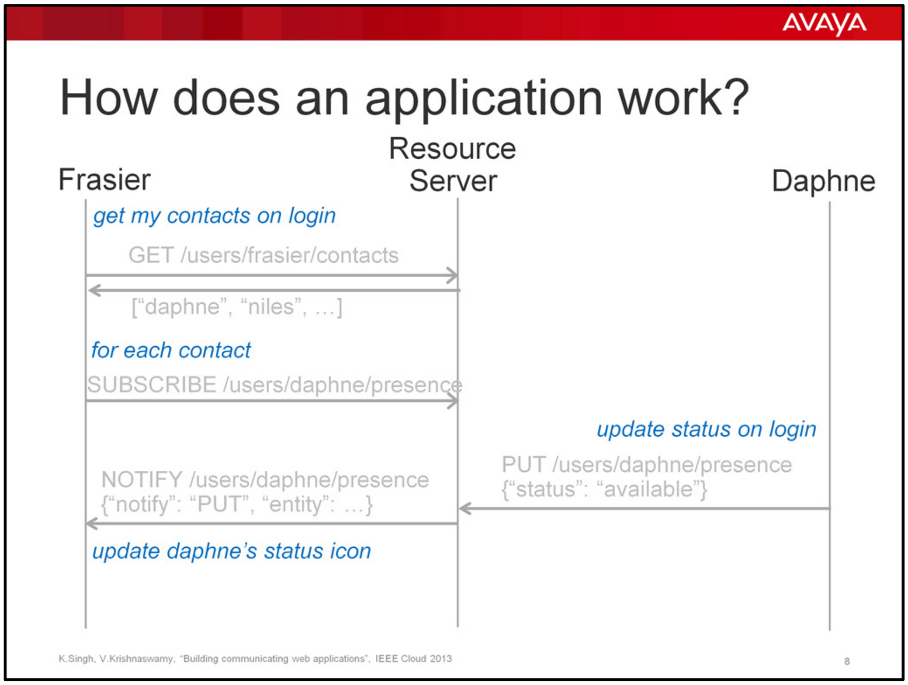
7

The main principle in resource based application model is as followed: all application logic runs in the client (i.e., the browser), whereas the server is just a simple and generic data access and event system. Since the server does not run complex or heavy application logic, it is easy to make it robust and scalable.

The resource service can be explained in five points. First, resources are pieces of data, organized hierarchically, similar to a file system. For example /users/bob/presence could be Bob's presence resource and /room/1234 could be a chat room's members list.

The client application running in the browser connects to the server, typically over a persistent WebSocket, and exchanges messages. The message format is in JavaScript Object Notation (JSON). The resource server defines generic set of data access and event messages, inspired by REST methods.

The resource server is essentially a web and WebSocket server, with a backend database to actually store the resources. Finally the JavaScript library contains the SDK and widgets used for creating communicating applications. I will describe the widgets later.



AVAYA

How does an application work?

Frasier

Resource Server

Daphne

get my contacts on login

GET /users/frasier/contacts

[“daphne”, “niles”, …]

for each contact

SUBSCRIBE /users/daphne/presence

update status on login

PUT /users/daphne/presence
{“status”: “available”}

NOTIFY /users/daphne/presence
{“notify”: “PUT”, “entity”: …}

update daphne’s status icon

K.Singh, V.Krishnaswamy, “Building communicating web applications”, IEEE Cloud 2013

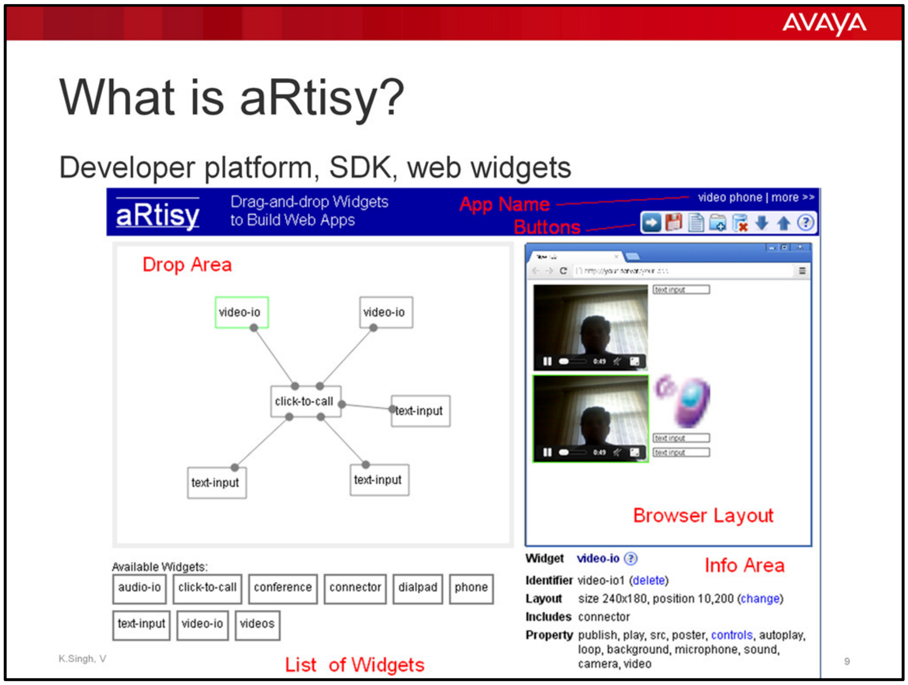
8

Let us look at one example of how a contact list with presence is implemented. Say, one user opens the webpage in his browser. The application logic in JavaScript on the page fetches the contacts list resource of this user, and for each contact it fetches and subscribes to the presence resource for that contact.

When a particular contact comes online, her application logic writes the presence resource with the status as “available”. Since the first user has subscribed for this presence resource, it gets notified, and updates the status icon of this contact in the contact list.

What is aRtisy?

Developer platform, SDK, web widgets



Widget video-io

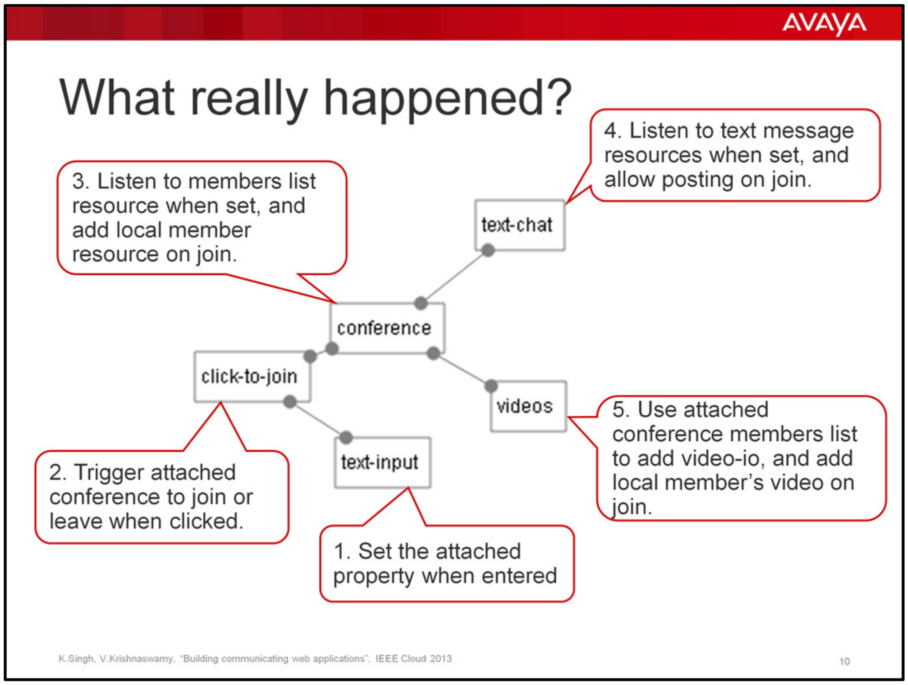
Identifier video-io1 (delete)

Layout size 240x180, position 10,200 (change)

Includes connector

Property publish, play, src, poster, controls, autoplay, loop, background, microphone, sound, camera, video

To further assist in creating different kinds of applications, we have created a developer platform called aRtisy. Let me show a quick demonstration video of the application builder in aRtisy.



AVAYA

What really happened?

1. Set the attached property when entered

2. Trigger attached conference to join or leave when clicked.

3. Listen to members list resource when set, and add local member resource on join.

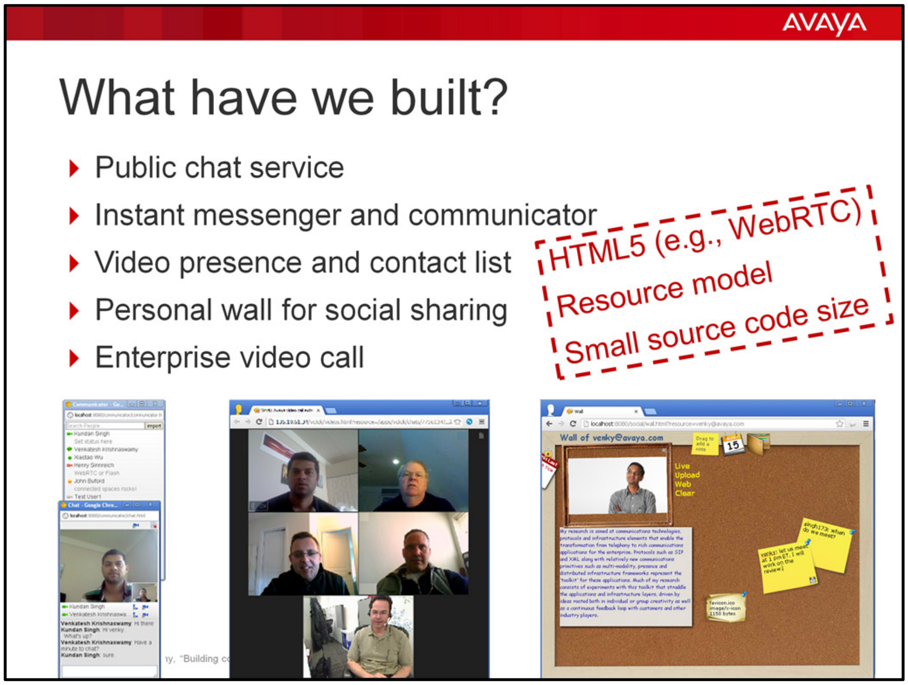
4. Listen to text message resources when set, and allow posting on join.

5. Use attached conference members list to add video-io, and add local member’s video on join.

K.Singh, V.Krishnaswamy, “Building communicating web applications”, IEEE Cloud 2013

What really happened behind the scenes? Each of the widget implemented a single application scenario, for example a button to join a conference, or layout of videos in a conference. We have many different kinds of widgets both telephony and web style.

These widgets mash up at the data level, e.g., the text chat widget uses the conference resource and local participant identifier from the conference, without knowing that videos widget is also attached to the conference.



AVAYA

What have we built?

- Public chat service
- Instant messenger and communicator
- Video presence and contact list
- Personal wall for social sharing
- Enterprise video call

HTML5 (e.g., WebRTC)
Resource model
Small source code size

Venkatesh Krishnaswamy: Hi there
Kundan Singh: Hi venky. What's up?
Venkatesh Krishnaswamy: Have a minute to chat?
Kundan Singh: sure.

My research is aimed at communications technologies, protocols and infrastructure elements that enable the transformation from telephony to rich communications applications for the enterprise. Protocols such as SIP and XML along with relatively new communications primitives such as multi-modality, presence and distributed infrastructure frameworks represent the 'toolkit' for these applications. Much of my research consists of experiments with this toolkit that straddle the applications and infrastructure layers, driven by ideas rooted both in individual or group creativity as well as a continuous feedback loop with customers and other industry players.

singh173: when do we meet?

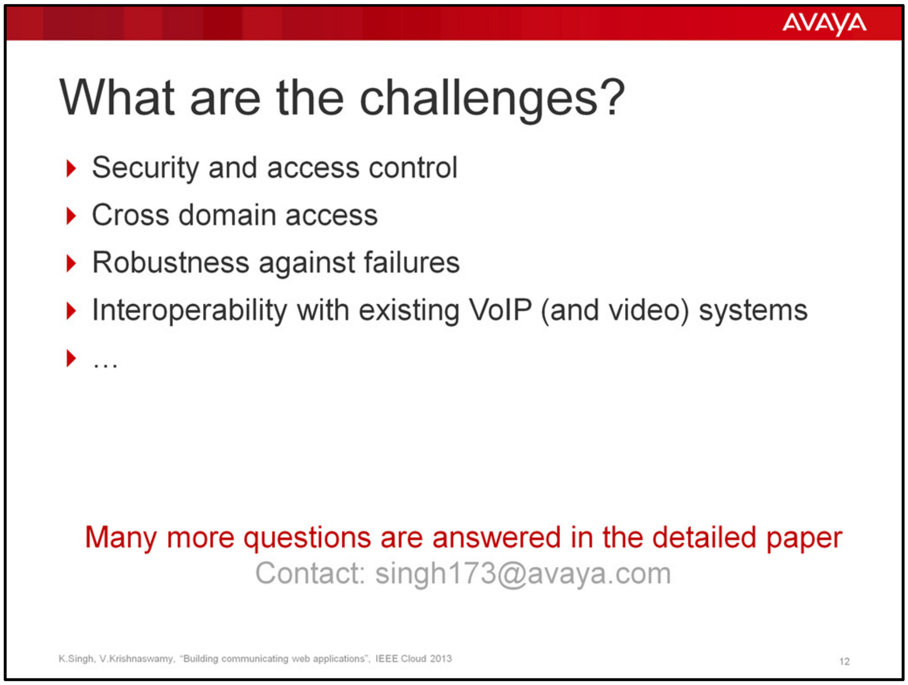
venky: let us meet at 1 pm ET. I will work on the review!

ıy, "Building c

We have built many more real applications using the resource-based application model and the widgets. This is just a list of applications and some screenshots, but the paper describes these applications in more details.

The public chat service is a simple cloud based multiparty video conference with text chat. The entire communicator functions such as contact list, presence, IM, file sharing, emoticons, video call, voice call, offline messages are implemented in our communicator, entirely using HTML5 and resource model, without any legacy Jabber or SIP systems. We have also built social network applications with profile, wall post and video presence.

Some points to remember about these applications: the application logic is written with HTML and JavaScript, using some HTML5 technologies such as WebSocket and WebRTC for real-time media. These applications are usually very small – few hundred to few thousand lines of source code.



What are the challenges?

- Security and access control
- Cross domain access
- Robustness against failures
- Interoperability with existing VoIP (and video) systems
- …

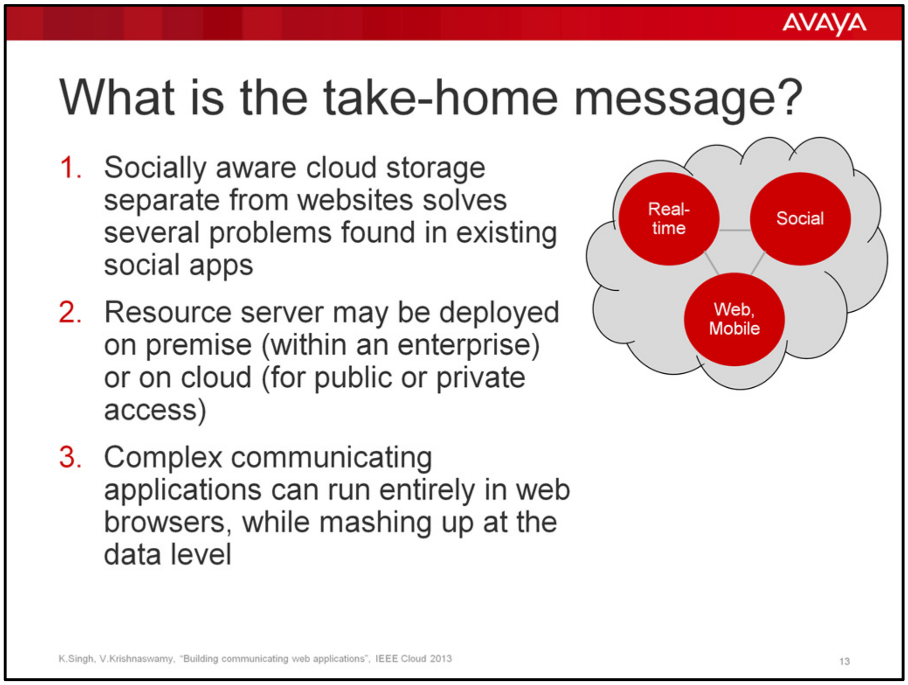
Many more questions are answered in the detailed paper

Contact: singh173@avaya.com

K.Singh, V.Krishnaswamy, “Building communicating web applications”, IEEE Cloud 2013

12

There are several challenges discussed in the paper. Particularly, for cloud deployment of the resource server, the security and robustness is very important, whereas for on-premise enterprise use case, interoperability with existing communication systems is useful.



AVAYA

What is the take-home message?

1. Socially aware cloud storage separate from websites solves several problems found in existing social apps
2. Resource server may be deployed on premise (within an enterprise) or on cloud (for public or private access)
3. Complex communicating applications can run entirely in web browsers, while mashing up at the data level

K.Singh, V.Krishnaswamy, "Building communicating web applications", IEEE Cloud 2013

13

So what should you take home from this presentation? First, socially aware cloud storage which separates the data from the application logic of the social website solves many problems found in existing social and cloud applications.

The resource model contains a resource server with a very generic data access and event message. The resource server can be deployed on premise within an enterprise, and can potentially be bound from public social websites. aRtisy is a web and cloud based developer platform to create such communicating web applications.

Finally, many complex communicating applications can be built in the resource model, where the application logic runs entirely in the client (browser), and mashes up at the data level. It solves the four problems we discussed in the beginning – redundancy, application lock-in, rigid data boundary and tied lifetime of the data.

In summary, our experiment presents a new way of cloud application development that involves real-time, social and web.