CMT 314: MOBILE APPLICATION DEVELOPMENT

Lecture Eight Notes

ADAPTING TO DEVICES

8.1 INTRODUCTION

Not all mobile devices are created equal. Thus the age-old problem in mobile design and development: devices can be vastly different from each other. It would be easy if different devices simply supported different attributes—one supports CSS3 and one doesn't. But it isn't that easy. One device might support CSS3 and another device might support CSS3 poorly—or worse, incorrectly.

This might not be a problem at all if we only had a few platforms or browsers to contend with. The mobile industry is an entirely different story altogether. In mobile, you have a half a dozen or so platforms, plus a dozen or so feature phone platforms. Add the plethora of mobile web browsers that run on each of these platforms, and you can start to see that the mobile web is a very fragmented and difficult space to support.

Detecting, *adapting*, and *supporting* multiple devices has historically been the worst pain point of mobile design and development. It usually requires a software system to perform a **three-step** process in order to render the best experience per device, creating plenty of opportunities for problems. Though there are several strategies to solve this problem, each of them work in this basic way:

Detect

The system must first detect the device requesting the content, known as device detection. This typically requires a valid and up-to-date device database, with all the pertinent information on the

hundreds of devices on the market, or in some cases a very smart mobile web browser.

Adapt
It next takes the requested content and formats it properly to the constraints of the device, which is called content adaptation. This might include the screen size, device features, appropriately sized photos, supported media types, or web standards support. This is done by having an abstracted layer of content and a layer of presentation and media for each supported device.

Deliver
Finally, it must serve the content to the requesting device successfully, usually requiring testing each device or class of device that it is intended to support

This entire process is often referred to as **multiserving**, **device adaptation**, or **simply adaptation**. As you might imagine, this process requires the consideration and adaptation of many variables. It is in no way an obstacle that cannot be surpassed, but it does add significant cost and complexity to a mobile project.

So what are we, as designers and developers, to do? We know that mobile technology is important—it is the platform for the future, and we want our content in the space— but dealing with all of these devices seems like a headache that we don't exactly want or need.

Adapting for multiple devices is the greatest risk that your project can make. Too many challenges are unforeseen, there is too much knowledge that takes too much time to learn, and the hidden costs can too easily multiply beneath you. So how does one balance the necessity of multiserving while still keeping mobile investments of time and money under control? We shall be looking at **five** *basic strategies* to work around this seemingly insurmountable obstacle:

8.2 IS ADAPTATION A "NECESSITY"

Is multiserving a "necessity", and if so, why? Yes, multiserving your content in one-way or another is absolutely a necessity. If you have a website and you want to have a mobile website, web app, or even native application that shares content in some way with another

context, you are going to need to figure out a strategy to make that happen.

In Figure 13-1, you can see a simple example of the software systems that someone might need if she wanted to have a *desktop site, a mobile website, a mobile web app,* and *a native app*. Each context is creating a standalone system, because more often than not, tying them all together would be costly in time and resources.

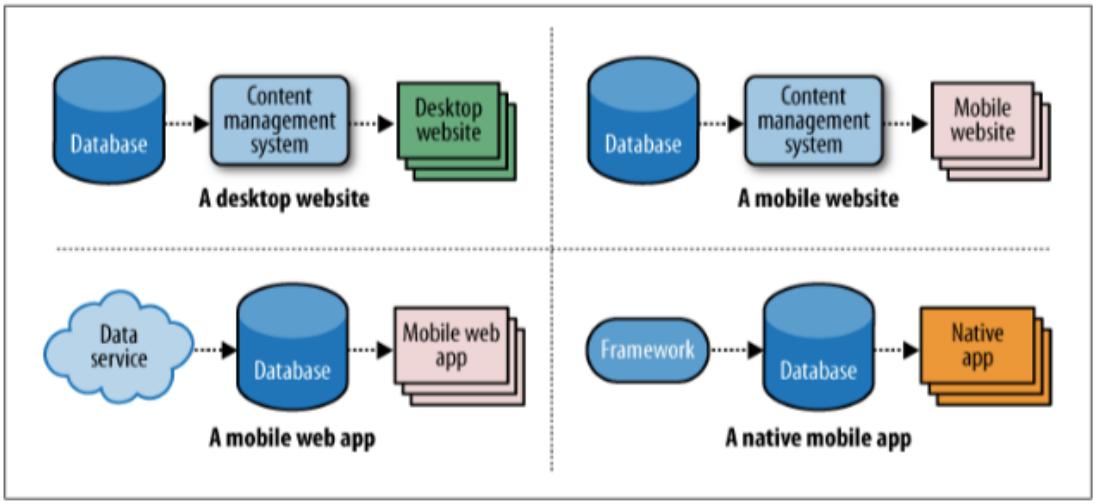

Figure 13-1. Serving multiple contexts can be redundant and expensive

Here, we create an internal API or data service to get each system talking to each other. With this strategy, getting information out of a central database through an API can be a snap, allowing users to create new sessions on the client, but storing data either locally or back into the context-specific database can be a hassle. The problems occur when trying to get each of these separate systems in sync.

For example, say we wanted to create a grocery shopping application, allowing users to create shopping lists from home, then use their mobile device as a shopping list at a physical grocery store. As we really only need to send data one-way, it might not be too problematic—a fairly simple multiserving strategy would most likely work. But what if we wanted the application to work in the reverse as well, allowing users to create a shopping list from their mobile device while they are out and about, then when they get home, to use it to place an online order for delivery? The more the user changes his context, the more complex the system needs to be to adapt.

Ideally, we could use one system for multiserving all of our content, as shown in Figure 13-2, with one central source of data outputting

content to multiple contexts, using whatever external tools are needed.

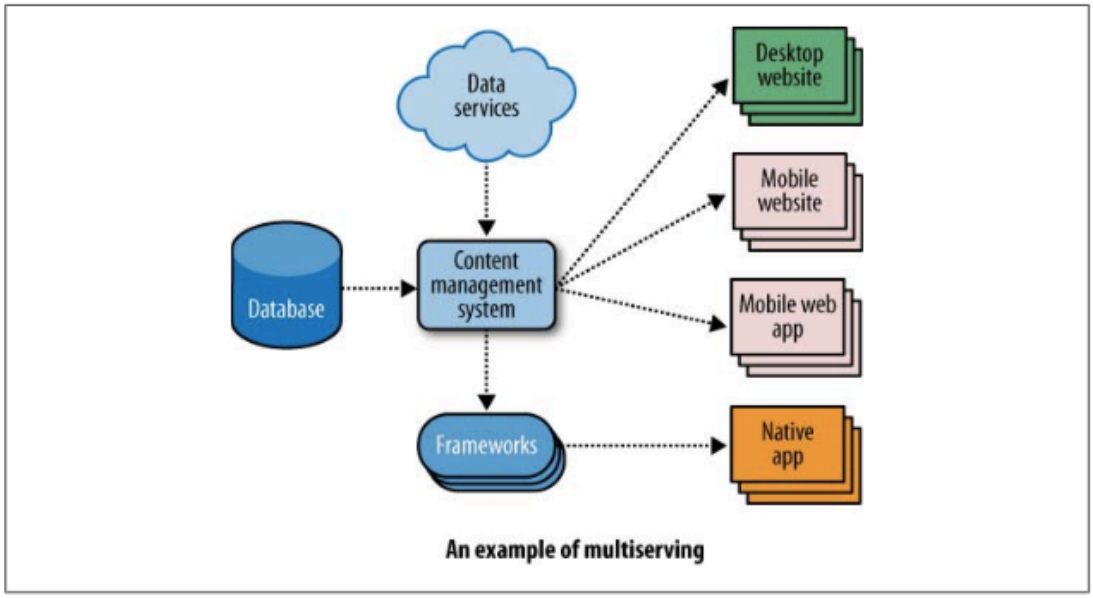

Figure 13-2. A simple example of a multiserving system

The question isn't whether it is a necessity; in today's environment, and with the growth of mobile devices starting to move beyond just phones, this isn't a problem that is going away anytime soon. The question is how to adapt for multiple devices with a minimal number of headaches. This brings us to our first philosophical debate: the adaptation / multiserving conundrum, and our first adaptation / multiserving strategy.

8.2.1 STRATEGY # 1: DO NOTHING

Our first of four multiserving strategies is to simply do nothing, or rather to wait for the technology to adapt to our principles—which actually isn't as foolish as it might sound.

In 2005, the W3C created the Mobile Web Initiative (MWI) to attempt to bring the standardization it achieved with the desktop web to mobile. This was a difficult challenge, because the W3C had not issued any specifications for mobile standards. The announcement of the effort included an introduction from Tim Berners-Lee about the importance of "One Web," a concept that has been a hotly debated topic in both the web and mobile communities ever since.

Initially, the W3C defined One Web this way:

One Web means making the same information and services available to users irrespective of the device they are using.

In other words, content should be published once and the device should be smart enough to know what to render; effectively, we wouldn't need to do anything more than ensure that our content meets the content standards. This actually makes a lot of sense, as we'd be able to write once and publish everywhere to multiple devices and multiple contexts.

This do-nothing approach is actually a multiserving system in its own right: it detects devices, though without the need of a device database; it adapts content based on the requested device through thoroughly separated content and presentation and renders it to the requesting device. The multiserving system in this scenario is the *browser*; assuming that it can intelligently render the appropriate experience that we define for it.

Five Assumptions About One Web

This approach can work really well when you design your content with it in mind, but it isn't very flexible or robust, and it isn't without flaws:

- It assumes that your content for multiple contexts will be the same, when it usually isn't.
- It assumes that cost per kilobyte to the user is minimal or nonexistent. In other words, it assumes that the user is willing to pay for content not designed for her context.
- It assumes that a persistent and high-speed data network will always be available.
- It assumes that mobile browsers are smart and will support the same standards consistently, which isn't the case, at least today.
- It assumes that a technology-based principle should come before the needs of the user.

Modern mobile browsers have been designed specifically to address the first four assumptions.

The problem with the One Web principle is that fifth assumption. It can be dangerous to put the desire for efficient technology before the

needs of users. Creating a great experience means finding the sweet spot between the constraints of the technology and the needs of the user.

8.2.2 STRATEGY # 2: PROGRESSIVE ENHANCEMENT

The technique of using web techniques in a layered fashion to allow anyone with any web browser to access your content, regardless of the browser's capabilities.

Though progressive enhancement is normally thought of as a development strategy, it can also be used as an adaptation strategy as well, going through each of the three steps of multiserving. Similar to "doing nothing," this approach assumes that the browser will be smart enough to detect, adapt, and deliver the right experience, but in this strategy we design a site to have several fallback points, supporting a larger number of devices and not requiring support of media queries.

For example, take Figure 13-3, where we can see our presentations layered on one another. At the bottommost level, we have a rich CSS3-based experience for our Class A browsers, including support for media queries. Above it, we have good CSS2 support, for Class B browsers, and so on, until we get to the top level, or our last fallback position of no styling whatsoever, for our Class F browsers. In this case, as long as our markup is semantically coded, it should still be usable on any device that can render HTML.

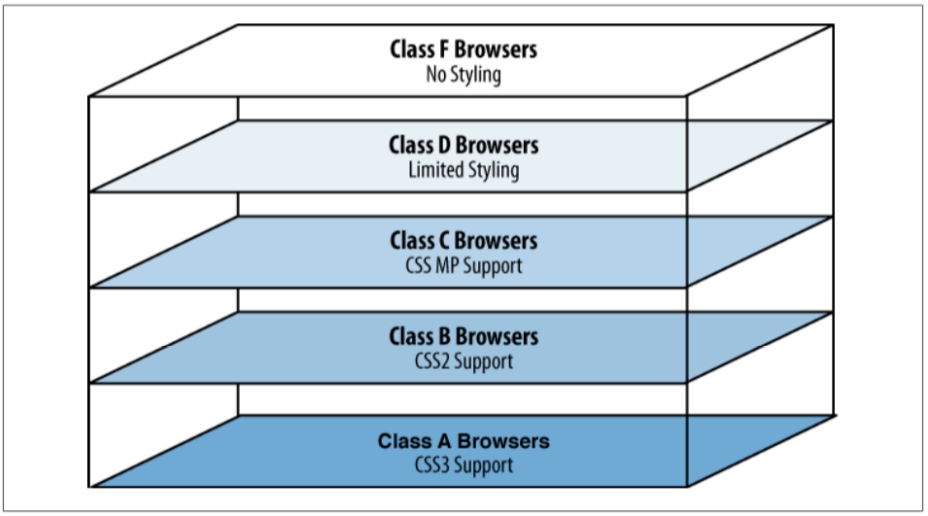

Figure 13-3. Using a progressive enhancement technique to establish several fallbacks to our presentation

8.2.3 STRATEGY # 3: DEVICE TARGETING

The third multiserving strategy is device targeting, or calling out specific devices by class or model and delivering an experience designed with that device in mind. In this strategy, we don't assume that browsers are trustworthy enough to get to where they need to be. Therefore, in this strategy, the first step of multiserving is to reliably detect the device.

Once the device and browser are detected, you route the user to the best experience. This can be done by checking the HTTP headers, starting with the User-Agent string, in order to recognize the device and browser and then deliver a device-specific site.

One of the more common examples here would be wanting to treat a device like the iPhone differently than the rest of your mobile devices; the markup and styles used to create the iPhone experience could be quite different than what you might use for all the other mobile devices. In these cases, each experience is often built as a standalone product, with little to no adaptation-taking place.

The Device Detection Dilemma

Large publishers with experienced developers can certainly hack their web server configurations to detect and route devices, even though it still requires having an up- to-date device database with all the device profiles and appropriate techniques to recognize a device. And in larger organizations it can take months to prepare, test, and QA any major changes to the server directives. For smaller companies using shared hosting, altering the server configuration at the levels needed isn't even an option.

8.2.4 STRATEGY # 4: FULL ADAPTATION

The fourth and final multiserving strategy is full content adaptation, or the process of making extremely unique mobile experiences based on the device that is requesting the content—almost always dynamically. Like the other multiserving strategies, it starts with detecting the device that is requesting content and matching that to a valid user- agent string; then the system outputs markup, styles, and images generated exclusively for that device.

Let's say, for example, that we want to support 20 devices across multiple operators. These devices carry browsers that range from Class A to Class D, each with different screen sizes and device capabilities. Using a full adaptation strategy, we detect, adapt, and render, but then we take it a step further. For each request, we detect each of those device's user-agent strings against our device database and then dynamically adapt our base templates to suit that device class and render to the device.

With full adaptation, we dynamically create four specific experiences for each of our classes based on a number of templates and assets designed to degrade or adapt that would render a unique experience to each device.

In addition, we may have interclass optimizations. An example of this might be when you have the same device deployed on two different operator networks. This is a more common problem in the U.S. market, where we have more than just one type of network standard. Occasionally, multiple devices can share the same model number and even have similar user agents but render content differently. In this case, we need full adaptation to do detailed lookups and make sure that we detect the right device on the right network, and then do interclass optimizations for that particular device.