Preferences for Global Access Architecture

Contents

Overview

A Framework for Preferences

Based on our experience researching and designing preference editing tools over the past decade, it is clear that no single design will meet the needs of all users. There is an incredible diversity of experience levels, comfort with technology, ability, age, and other factors that influence the design of user interfaces for preference discovery. We need to speak the language of learners, giving them an environment in which they feel comfortable discovering what they need and experimenting with new settings. The Preferences for Global Access software architecture needs to take into account this diversity, making it easier for designers and developer to create different user interfaces for different users.

To this end, the project's architectural approach emphasizes modularity and adaptability. Rather than creating the infrastructure for a single, hard-coded preferences editor, the goal is to provide the technical infrastructure that will allow for maximum diversity of user experiences while avoiding the proliferation of duplicated and mostly unshared code typical of ad-hoc UI development. The PGA architecture follows Fluid Infusion's declarative Inversion of Control (IoC) philosophy, making it easier for third-party developers and designers to refine and extend the PGA's preferences discovery environment over time. With Infusion IoC, instead of writing code with tightly-coupled dependencies, developers express their application structure as a declarative "component tree" that is instantiated and managed by a context-aware framework (Basman 1). This makes it possible to change the application later, either statically by third-party developers or at run time by evaluating contextual information such as device capabilities and physical sensors.

Given the diversity of platforms, tools, and devices that learners are increasingly using in the classroom and at home, the Preferences for Global Access architecture needs to be broadly cross-platform and compatible. Web technologies such as HTML, CSS, and JavaScript are ideally suited to delivering cross-platform, cross-device user interfaces without requiring extensive code to be written for each new platform. The PGA architecture embraces these standard tools on both the client and server (with Node.js), making it easier to integrate our preferences environment into the applications learners use the most.

In keeping with the overall AccessForAll approach, in which we understand disability and accessibility to be usability issues that we all experience, preferences are always represented functionally. They never require the user to identify medical conditions or limitations; instead, they express functional preferences that may be useful to a wide variety of users.

Linkage to the Global Public Inclusive Infrastructure

The Global Public Inclusive Infrastructure (GPII) is an international effort to create tools, infrastructure, and resources that can automatically adapt user interfaces to suit an individual's needs and preferences. With the GPII, users can store their preferences securely in the cloud, taking them with themselves as they move to different platforms and devices.

The Preferences for Global Access architecture is closely aligned with the GPII architectural effort. PGA is being designed alongside the GPII by a shared community of architects and developers; the opportunity for productive overlap is significant. In particular, the PGA architecture will be connected with the GPII cloud-based Preferences Server, Flow Manager, Matchmaking, and security/privacy infrastructure. The reuse of GPII technology will substantially lower the development effort for the PGA discovery environment while ensuring that there is a large and supportive community of developers and designers to assist.

Security and Privacy

A user's preferences may reveal personal information about their lives and needs, either directly or indirectly; privacy and security is thus of critical importance to the design of the PGA architecture. This is another case where the project will closely follow the approach established by the Global Public Inclusive Infrastructure. While the security and privacy infrastructure work is still in its early stages, the intention is to use, for both projects, the OAuth 2.0 framework to protect access to a user's preferences by third party applications such as the web sites, content delivery tools, and OER authoring environments which will implement PGA (Hardt 2012). The use of OAuth will let users approve or deny access to their information on a per-site basis, ensuring they control who gets to see and act on their needs and preferences.

In the future, an attribute release layer such as the Kantara Initiative's User Managed Access will be layered on top of the basic OAuth 2.0 authorization system (Machulak 2010). This will give users the further ability to specify that a site or application is able to see only portions of their preferences set, ensuring that the risk of "privacy leakage" is reduced by only sharing the minimum information required by a service to meet the user's needs (Krishnamurthy 2009).

The ultimate goal is to ensure that this architecture can support diverse privacy and security requirements internationally.

Architectural Components

Diagram showing the primary architectural components of Preferences for Global Access, including the UI Options framework, the GPII Preferences Server, and server side transformation components.

UI Options Preferences Framework

User Interface Options (UI Options) is a component that ships with the Fluid Infusion application framework. It is currently being extended to serve as much more than just a UI component; UI Options provides a programming framework with which new user interfaces for editing preferences can be more easily built. The goal is for UI Options to provide all the lifecycle events, configuration hooks, and persistence infrastructure required to support a variety of different editing experiences, ensuring that there is a robust and well-tested framework in which to incrementally move the PGA designs from prototype to production. The UI Options framework is described in further detail below.

GPII Preferences Server & Flow Manager

The GPII Preferences Server represents a cloud-based service for storing and retrieving user preferences in a secure manner. Preferences are stored within the preferences server in a JSON format that is compact and easy to parse by a wide variety of tools. In order to enable broad compatibility and "available anywhere" portability, the GPII preferences server will be used as the primary means for persisting user preferences within the Preferences for Global Access architecture. Further information about the architectural considerations of the preferences format and server can be found in (Clark and Basman 2012).

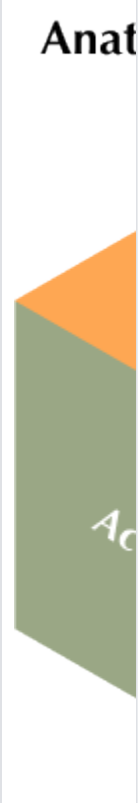
Server-Side Transformation Services

Many user preferences can be enacted using client-side web technologies (for example, large print or high-contrast themes can be applied using CSS), and this will be the default approach for the PGA architecture. However, there are cases where current browser technology is unable to accommodate a user's needs and preferences directly with HTML, CSS, or JavaScript. Text-to-speech is an example of this, where browsers don't yet have the ability to synthesize speech directly. In these cases, server-side transformation services such as a text-to-speech server using the open source Festival engine will be built. The key architectural approach here will be to leverage server-side JavaScript with Node.js and Fluid Infusion so that, as browsers rapidly improve, logic can be more easily ported from the server-side to the browser.

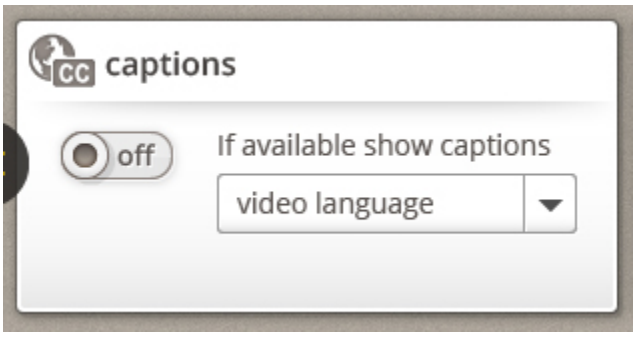
A Framework for Extensible Preferences

The UI Options framework is intended to give developers the ability to supply custom appearances, behaviours, and schemas for any preference, including new preferences that aren't part of the default UI Options package. This is achieved architecturally by modeling each preference as three discrete objects: a **View** (presentation) component, an **Enactor** (action) component, and a **Persistence** component. This three-part model ensures that developers and designers who either are integrating UI Options into their site *or* who are using it as a general framework for building their own preferences editor (such as the PGA discovery tool) have complete control over how a preference 1) appears to the user, 2) is enacted, and 3) is stored.

Diagram showing the architecture of preferences within the UI Options framework, including presentation, action, and persistence



For example, let's consider the process of adding a new preference to the framework: closed captions and subtitles for videos. The user interface this preference might be designed with two controls in it: a stylized on/off switch that allows the users to turn captions on and off, and a drop-down menu allowing them to select their preferred language. Here is an illustration:



From a technical perspective, when the Captions preference is enabled, all video players should automatically show closed captions if available. In practice, this might be accomplished by creating a JavaScript component that automatically finds supported video players on a page and invokes their API to turn on captions or subtitles.

So, from an architectural perspective, the developer who is creating this new preference will need to specify three things:

1. A View component that represents the panel. It will specify the markup, styling, and rendering required to show the dropdown menu and switch /checkbox.
2. An Enactor component, which will do the work of finding all video players on the page and enabling captions or subtitles for them.
3. A Schema, which defines default values for each of the settings. In this case, the preference might default to "off" and the language might default to user's browser locale.

In each case, the UI Options framework provides a declarative JSON data structure for registering custom components, enabling easy customization of a preferences editor by third-party developers and integrators without having to modify any built-in code. As a result, the job of the developer is to compose preferences together into a tree of JSON configuration that is passed to the UI Options framework at instantiation time.

By separating the responsibilities of view and action, this architectural approach will enable developers to more rapidly create diverse preference editing experiences. For example, a designer may want to provide both a highly simplified user interface that is more fun and approachable for children as well as an in-depth experience for more technical users. In this case, the same Enactor code can be reused while different Views can be written and swapped in without having to change any core application code.

The UI Options framework also provides support for pluggable previews, enabling a user to see the effect that a preference change will have before committing to it. Developers can provide their own custom preview strategies. The default "live preview" strategy shows the changes applied to a live view of the application, allowing users to see their preferences in context. Another built-in strategy in include a "sample window" preview showing a small selection of typical content. We also envision specialized previews where the user can try out the preferences in practice in a safe environment before committing to them globally.

The View

The View component is responsible for specifying the markup, styling, and template required to render the panel. Here's an example of what the declarative view specification might look like for this CaptionView component if it used the Infusion Renderer to present its markup:

```
fluid.defaults("fluid.uiOptions.views.captionView", {
    gradeNames: ["fluid.uiOptions.view", "autoInit"],
    strings: {
        language: ["English", "French"]
    },
    controlValues: {
        language: ["en", "fr"]
    },
    produceTree: "fluid.uiOptions.views.captionView.produceTree",
    resources: {
        template: "{templateLoader}.resources.captionView"
    }
});
```

The Enactor

The Enactor does the actual work of accommodating the user's preference. In the case of web application, Enactors are typically registered with the UI Options framework's **UI Enhancer** component, which is responsible aggregating multiple Enactors together and managing the interaction between these Enactors and the page's Document Object Model (DOM).

Enactors can be of arbitrary complexity. In many cases, an Enactor may well manipulate DOM elements directly (such as a content simplifier that removes HTML5 and ARIA landmarks such as "banner," "footer," and "navigation" while leaving the "main" content untouched). In other cases, such as this closed captions example, the Enactor may delegate the actual work of performing the action to another module (such as an HTML5 video player component). Here's an example of how a developer might configure a UI Enhancer component with this new closed caption and subtitles Enactor along with others:

```
fluid.defaults("fluid.uiEnhancer.defaultActionsWithCaptions", {
    gradeNames: ["fluid.uiEnhancer", "fluid.uiEnhancer.browserTextEnhancerBase", "autoInit"],
    components: {
        captions: {
            type: "fluid.uiOptions.enactors.captionsEnactor",
            container: "body"
        },
        textSize: {
            type: "fluid.uiOptions.actionAnts.textSizerEnactor",
            container: "{uiEnhancer}.container",
            options: {
                fontSizeMap: "{uiEnhancer}.options.fontSizeMap",
                sourceApplier: "{uiEnhancer}.applier",
                rules: {
                    "textSize": "value"
                }
            }
        },
        ...
   }
});
```

Persistence Information

The Persistence component provides schematic information about the preference---its value types, ranges and default values. Developers can also plug in custom **Data Store** components, which enable the preferences editor to connect to different data stores. Several default Data Store objects ship with UI Options out of the box, including one that stores the user's preferences in a cookie or in HTML5 local data storage, one that saves the preferences directly into the GPII cloud-based Preferences Server, and a temporary one that is suitable for testing.

Here's an example of what a simple schema component might look like for our closed caption and subtitles example, including a set of default values and a JSON Schema (Zyp 2013):

```
fluid.defaults("fluid.uiOptions.mediaSchema", {
    gradeNames: ["fluid.uiOptions.schema", "autoInit"],
    defaultModel: {
        captions: false, // boolean
        language: "en" // ISO 639-1 language code
    },
    schema: {
        properties: {
            captions: {
                type: "boolean"
            },
            language: {
                type: "string"
            }
        },
        required: ["captions", "language"]
    }
});
```

References

- Basman, A., Lewis, C., Clark, C. "To Inclusive Design Through Contextually Extended IoC". In C. Videira Lopes and K. Fisher (eds): Companion to the 26th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2011. pp 237-256. Web: http://wiki.fluidproject.org/download/attachments/1707985/Infusion-Splash-Wavefront-2011-Paper.pdf?version=1&modificationDate=1317768364132
- Clark, Colin and Antranig Basman. *GPII: Architecture of Preferences and Data*. 2012. Web: http://wiki.gpii.net/index.php/Architecture_of_Preferences_and_Data
- Fielding, R.T. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation (2000), University of California, Irvine. Web: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
- Hardt D., ed. *The OAuth 2.0 Authorization Framework*. Internet Engineering Task Force, 2012. Web: http://tools.ietf.org/html/rfc6749.html
- Krishnamurthy, Balachander and Craig E. Wills. _On the Leakage of Personally Identifable Information Via Online Social Networks_. Proceedings of ACM SIGCOMM Workshop on Online Social Networks, 2009. Web: http://www2.research.att.com/~bala/papers/wosn09.pdf

- Machulak, Maciej P., Eve L. Maler, Domenico Catalano, and Aad van Moorsel. *User-Managed Access to Web Resources*. Proceedings of the 6th ACM workshop on Digital identity management, 2010: 35-44. Web: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.161.7027&rep=rep1&type=pdf
- Zyp K., ed. *JSON Schema: core definitions and terminology*. Internet Engineering Task Force, 2013. Web: http://json-schema.org/latest/json-schema-core.html