Enabling Architecture: How the GPII Supports Inclusive Software Development

Colin Clark[1], Antranig Basman[2], Simon Bates[3], and Kasper Galschiøt Markus[4]

[1] OCAD University, Toronto, Canada
cclark@ocad.ca
[2] Raising the Floor - USA
amb26@raisingthefloor.org
[3] OCAD University, Toronto, Canada
sbates@ocadu.ca
[4] Raising the Floor - International, Geneva, Switzerland
kasper@raisingthefloor.org

Abstract. The Global Public Inclusive Infrastructure is an international effort to build tools, components, services and a sustainable community to support personalized digital inclusion[1]. The GPII is building the critical infrastructure needed by developers to produce the next generation of low-cost assistive technology and highly flexible applications that can adapt to the needs and preferences of individuals across web, desktop, and mobile platforms.

To deliver on these ambitious goals, the GPII architecture team has created an evolving suite of development tools, idioms, and resources to support the creation of an inclusive infrastructure.

Keywords: Accessibility, inclusive design, GPII, development tools, assistive technology, JavaScript, Node.js, Inversion of Control.

1 What Is the GPII Development Platform?

The GPII Development Platform provides reusable frameworks that are employed extensively throughout the GPII ecosystem. These frameworks are designed to reduce the time, cost, and complexity of developing core services that conform to the GPII architecture, as well as ensuring that software is more easily testable, scalable, and capable of accommodating diverse user needs and preferences.

Currently, the GPII Development Platform consists of the following core framework technologies:

1. **Infusion**, a JavaScript application framework built by the Fluid Project to support highly flexible, model-driven, and personalizable applications and authoring environments on the web [7]
2. **Kettle**, a companion framework to Infusion that supports the creation of RESTful, JSON-oriented server-side applications and services[8]

C. Stephanidis and M. Antona (Eds.): UAHCI/HCII 2014, Part IV, LNCS 8516, pp. 368–377, 2014.

3. The **real-time framework**, which provides the core components and lifecycle support for the GPII's "personalization from preferences" functionality[2]
4. The **preferences framework** is built on top of Infusion and the real-time framework and supports the creation of diverse preferences editors and discovery tools that are tailored to users[6]

2 Overview of the Frameworks

2.1 Infusion

Fluid Infusion is a JavaScript library that primarily comprises the Infusion Inversion of Control (IoC) system and the Fluid Renderer. Infusion runs both on the client-side in a web browser as well as on the server and local devices using Node.js[16]. In a web browser, it uses the popular jQuery library[20] as a foundation library. Inversion of Control is a powerful software development technique popularised by Martin Fowler[5] and others, which helps to bridge the worlds of developers, integrators and end users by deferring to a framework the responsibility for wiring together parts of an application. In the case of Infusion, the dependencies between components are specified in a declarative form. Declarative programming as well as the related topic of aspect-oriented programming (AOP) will be discussed in sections 5 and 7.

2.2 Kettle

Kettle, another JavaScript library, lies one level above Infusion in the architectural stack. It is a piece of server-side infrastructure that makes use of Infusion's facilities for declarative programming to help developers express server applications in terms of easily authorable and sharable JSON documents. Kettle is capable of expressing server endpoints using standard web protocols. Kettle currently supports two types of endpoints: traditional RESTful services using HTTP, and WebSockets[3] endpoints that are suitable for realtime, bidirectional communication between client and server. As well as being based on Infusion, Kettle makes use of the popular Node.js server-side JavaScript platform[16] and the Express middleware library[17] for foundational HTTP and WebSockets protocol support.

2.3 The Real-Time Framework

The GPII's real-time framework is itself layered on Kettle and hence also on Infusion. It is responsible for orchestrating the personalization workflow both on local devices (for mobile and desktop operating systems) and in the cloud (for browser-based applications). The real-time framework provides the events and extensible lifecycle points that enable a user to be recognized, their preferences fetched from the cloud, and the appropriate applications and settings to be configured automatically for the user. The real-time framework is composed

of several major components including the Preferences Server, Flow Manager, Solutions Registry, Matchmakers, and the Lifecycle Manager. These components are discussed in detail in [18]. When the real-time framework runs locally on a Windows, Linux, or Android device, it interfaces directly with the operating system's applications, access features, and settings storage mechanisms. On the web, it produces a specification that describes the settings required by the user, delegating to the web application itself to actually enact the necessary adaptations.

2.4 The Preferences Framework

The preferences framework is a further JavaScript library that occupies an intermediate position in the technology stack. It is layered on top of Infusion and is capable of interacting with the real-time framework in order to mediate support for editing, transforming, and enacting user preferences and settings. It includes several browser-based user interface components implemented using Infusion, which can be used both to visualize and adjust a user's preference set. It also includes components to store and retrieve a preference set from various sources, including local browser storage and the preferences server component of the real-time framework. It also has the capability to allow a user or an integrator to create their own preference editing interfaces from elements authored by others, rather than requiring the involvement of a developer to write a new application. In order to do this, it makes use of the declarative programming capabilities offered by Infusion's IoC.

3 How the Development Platform Is Used

The Development Platform is currently in active use by a number of teams and projects working to build the Global Public Inclusive Infrastructure and other tools. As mentioned above, Kettle and Infusion are used foundationally across a large number of GPII components, and are used to build the other parts of the Development Platform such as the real-time framework and the preferences framework.

Beyond this, the Development Platform is being used to automatically configure desktop, mobile, and web platforms. The real-time framework can be installed locally on any Windows, Linux, or Android device. Web-based applications can invoke the cloud-based Flow Manager as a web service.

The Cloud4All project is currently developing two applications that enable a user to edit her preferences and device settings. The *Preferences Management Tool* provides a comprehensive web-based user interface for creating, editing, and saving a needs and preferences set to the GPII Preferences Server. The *Personal Control Panel* resides on the user's local device and allows for the adjustment of current settings on-the-fly. For example, the PCP can be used to quickly toggle captions on when the user enters a noisy environment.

The PMT and the PCP use the preferences framework to:

1. render user interface controls that allow the user to change preferences and settings
2. enact in-page previews that show the effect a particular change will have on the system
3. retrieve, transform, and save user preferences to a data source

Further information about the preferences framework and how it can be used by developers of preferences editors is discussed in [21].

Infusion, as the most mature framework in the GPII Development Platform, has also been used for a variety of other applications and projects that preceed the GPII, such as uPortal [22], Fluid Engage [23] and CollectionSpace [24].

The following diagram illustrates the dependency relationships between components of the GPII Development Platform as well as applications that use it. Solid lines represent integral dependencies, while dashed line denote optional or "soft" dependencies that can be swapped out by specifying an alternative configuration. It is worth noting that all dependencies occur downward in a acyclic

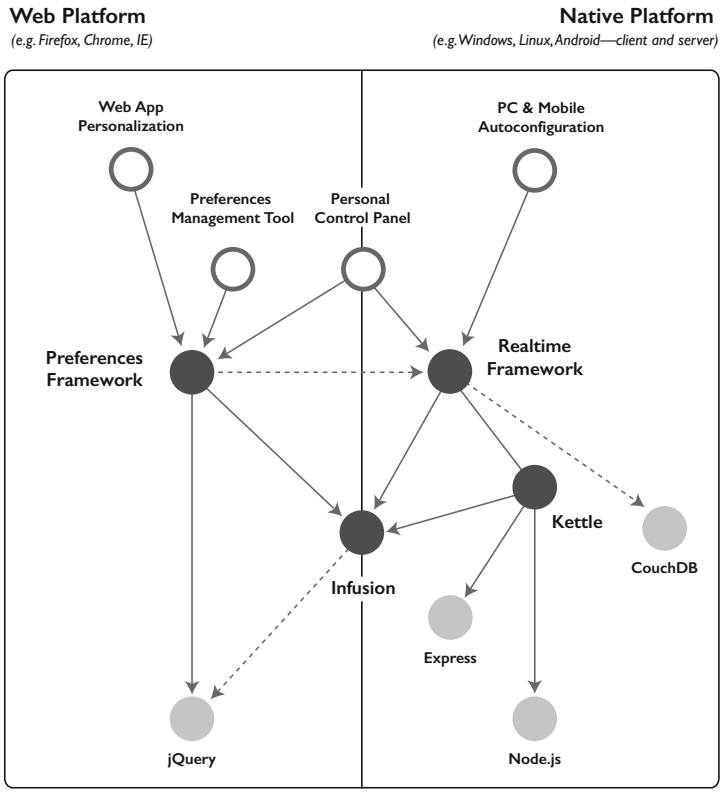

Fig. 1. The components and dependencies of the GPII Development Platform

graph, illustrating the fact that the frameworks of the GPII Development Platform are effectively layered. This ensures that a developer can use a lower-level component without requiring a dependency on higher or more specific layers of the system.

4 Infusion and Architectural Flexibility

The GPII's architectural approach and philosophy is inspired by Infusion, which serves as the technical backbone on which the Development Platform is built. Infusion is an ongoing project of the Fluid community, a key contributor to the GPII effort.

Infusion provides a comprehensive framework and Inversion of Control system that provides developers with a means to weave together many small, self-contained modules into a whole application without requiring hard-coded relationships between each module[4]. This helps to ensure:

1. a high degree of testability; each unit in a program can be tested in isolation, and mock objects can be inserted into a program to support integration testing
2. maintainable long-term growth; individual modules can be changed or swapped out with alternative implementations without requiring an exponential explosion of changes throughout the code base as new features are added
3. adaptability and reconfigurability; a program can be changed or modified freely by others simply by defining context-aware configuration that adds, removes, or overrides an application's functionality

These values of testability, scalability, and end-user adaptability are foundational to the entire GPII Development Platform, and require a different mindset when writing code. Traditional object-oriented programming techniques and conventional web development frameworks are rarely capable of supporting software "ecosystems" in which others are free to adapt, configure, and modify an application without having to fork and change its inner workings. In contrast, we envision a world where programmers, integrators, and, most importantly, end-users are all able to:

1. modify existing software
2. assemble new software from component pieces
3. develop and share novel authoring and customization interfaces

In other words, users should be able to make themselves "feel at home" in their digital environment by being able to adapt their environment to suit their needs, accessibility requirements, and personal habits. The history and motivations for this approach are described below.

5 The GPII Platform in Context of Software Development Trends

Typical programming code is designed for a limited audience. Imperative programming of the '70s and '80s was designed primarily for the compiler. If you wanted to change the behaviour of a program, you needed to find its source code, modify it, and submit it to the compiler again. Object-orientation, a scheme for promoting greater code reuse that became popular in the '80s and '90s, increased this audience marginally; it was possible to *derive* from the originator's implementation without necessarily having access to its source. However, this code still needed to be compiled and then submitted to the user in place of the originator's version.

Later programming developments such as *aspect-oriented programming* (AOP) promised to lengthen the chain of possible creative networks by one more link[12]. With AOP, it was possible to *advise* an existing implementation from the outside in order to change its behaviour, making use of global specifications that match pieces of implementation wherever they may be. Despite this improvement in reusability, these pieces of advice could not themselves easily be advised, creating yet another limitation for extensibility.

We argue that in traditional development environments, users and integrators who want to modify an originator's implementation, but who are more than a certain critical distance from the originator's community, are typically locked out of being able to assemble and distribute modified versions. In this situation, the only recourse is to make a *fork* — to take the original implementation and create a copy that diverges from a snapshot taken of the original at a particular instance in time. The risk of forks in open source software is well-documented[9]. Although tools increasingly exist to reduce the costs of resolving such forks, members of one community are cut off from the benefits of innovation in another. This causes a loss of effort that is proportional to the number of different communities involved. With the GPII, our goal is to enable a "one size fits one" model of accessible development. This entails the creation of many different "versions" or adaptations of an application that can productively coexist and interact. We call this a *community of software creativity*. Such communities require a solution to the problems of large-scale reuse and forking.

6 Supporting Creative Communities

With the GPII Development Platform, we imagine an unbounded sphere of creativity that stretches from the communities of our developers to the communities of our users. A crucial touchstone of our approach is that *any action performed by one creator should be undoable by another*. That is, there should be no limitations in the system that causes the intention of some creators to be privileged above the contributions of others.

To accomplish this technically, the GPII Development Platform attempts to model applications as *documents* encoded as JSON-based *component trees* that

can be shared, aggregated, modified, and re-shared without breaking the informational chains which link together diverse communities of interest.

7 Declarative Programming

Above, we described the GPII Development Platform as being declarative in nature. Although there is significant debate about which characteristics are intrinsic to a formally "declarative" system[13], J.W. Lloyd's informal description of the approach is useful as a pragmatic definition. He states that declarative programming entails "stating *what* is to be computed but not necessarily *how* it is to be computed" [14]. The emphasis is on the logical or semantic aspects of computation, rather than on low-level sequencing and control flow. More importantly, Paul Graham identifies the essential characteristic of declarative programming as representing program logic in data structures that can be manipulated by other programs. Discussing Lisp, he says that it "has no syntax. You write programs in the parse trees ... [that] are fully accessible to your programs. You can write programs that manipulate them ... programs that write programs"[19]. This characteristic is essential to Infusion and many other parts of the GPII Development Platform.

Infusion's declarative programming idiom, specifically the JSON-based *component trees* that represent the structure and relationships of a program in a semantically meaningful way, supports the creation of authoring tools and flow-based or model-driven programming environments. Where typical programming code, as described above, is one-directional and opaque to third, fourth, and fifth parties (i.e. the developers, integrators, and end-users who will ultimately customize and adapt their software), the GPII Development Platform attempts to open up the meaning and structure of an application to be editable both by humans and authoring tools.

This declarative approach is applied throughout the GPII Development Platform. For example, Kettle provides a declarative representation of the structure of server-side applications. A developer focuses on configuring her application and routing logic as trees of *server*, *app*, and *data source* components operated by a common framework based on Node.js[16], Express[17], and Infusion. Similarly, the Preferences Framework exposes a higher-level, schema-based representation of user preferences and how they should be bound to web UI controls such as sliders, buttons, or select boxes. This makes it easier for other components of the GPII infrastructure, such as matchmakers, to automatically generate personal control panels that are optimized to the particular user.

8 Third-Party Support Libraries

In addition to the frameworks that comprise the GPII Development Platform, a collection of lower-level third-party libraries is also employed. These help to provide a solid foundation that is aligned with prominent tendencies in open source web development. Aside from saving time and effort while developing

the core responsibilities of the GPII, the use of third-party libraries also ensures that the development platform is interoperable with other popular tools available to developers. These prominent third-party libraries include CouchDB, jQuery, Express, and others. A few of these libraries are discussed below.

8.1 CouchDB

CouchDB is a document-oriented database that stores data natively in JSON[10]. Queries (or rather, *Views*) to a CouchDB database are expressed in JavaScript code using a map/reduce approach to enable highly concurrent database indexing[11]. Data is saved and retrieved using a RESTful, HTTP-based API, making CouchDB an effective fit for resource-oriented web applications such as those in the GPII architecture. Notably, storing data as JSON objects avoids common application architecture pitfalls such as the "impedance mismatch" problem common to most object/relational mapping libraries[15]. The Kettle framework provides a built-in *Data Source* for accessing data stored in CouchDB. Data Sources are an abstraction representing an arbitrary source of data that implements the get, set, and delete semantics of HTTP while preserving a highly declarative interface for developers.

8.2 CouchDB

jQuery is a very popular library for managing common front-end web development tasks such as Document Object Model (DOM) manipulation, rendering, and making network requests[20]. jQuery is employed by Infusion to provide a familiar interface for web developers who are developing dynamic HTML-based user interfaces.

9 Results and Next Steps

It is difficult to objectively measure the productivity afforded by a set of development tools without costly and error-prone comparison studies. It is the opinion of the authors that such an approach rarely produces pragmatic information with which our tools can be improved. Instead, it is more effective to listen to the subjective experience of developers who work daily with such frameworks and to measure the success of these tools based on the architectural principles of reusability, extensibility, and the potential for enabling communities of contribution.

Informally, we have observed that the developers of top level GPII components such as the Preferences Management Tool, the Personal Control Panel, and the Real-time Framework's Matchmaker components are able to develop their components in a way that is more effectively isolated from changes occurring in the rest of the system. The PMT and PCP tools, for example, are currently being developed in a distributed manner across two continents and four different time zones. This social complexity is simplified by the system's separation of

concerns and layering, where a developer is able to work on their own component or "vertical slice" of an application without being impacted by parallel efforts occurring elsewhere in the codebase.

Another example of the practical benefits we have experienced when using the GPII Development Platfrom is related to a significant recent refactoring of the real-time framework, where Matchmaker implementations were migrated from inside the same runtime as the Flow Manager out into remote, cloud-deployed web services. The Flow Manager's strict decoupling of services from their dependents, provided by Infusion's IoC system, enabled this large-scale refactoring to occur with minimal impact on the Matchmakers themselves. More notably, no changes were required to the portions of the realtime-framework that depend a Matchmaker and its results.

Although the Development Platform has provided a number of observable benefits to developers of the GPII, there have also been significant challenges in adopting the new mindset that accompanies these frameworks. Developers have often found it intially difficult to navigate the unfamiliar landscape of JSON-based configuration, particularly in cases where they are accustomed to writing tightly-coupled code, such when developing user interfaces directly with jQuery. The Development Platform's greater abstraction and emphasis on declarative idioms provides significant production-oriented benefits, but it also represents a learning curve in these situations. This difficulty is compounded by a lack of documentation for many parts of the system, particularly Kettle. To date, we have attempted to address this problem by pairing new developers up with experienced mentors. However, this is not scalable and documentation represents a weak aspect of the system and a major point for future improvement. The GPII community is actively working to improve the comprehensiveness of the Development Platform's documentation, including tutorials, API references, and conceptual background information.

10 Community-Based Sustainability

The GPII Development Platform is open and evolving. It has benefitted from the more than six years of active development invested in frameworks such as Infusion and Kettle, and continues to grow under the stewardship of a federation of open source communities and funded projects such as Fluid, Cloud4All, Prosperity4All, Preferences for Global Access, Floe, and more. The software is available under an "open/open" license, and can be extended, modified, adapted, and commercialized freely.

References

1. GPII: The Global Public Inclusive Infrastructure, `http://gpii.net/`
2. Clark, C., et al.: A Detailed Tour of the Cloud4All Architecture, `http://wiki.gpii.net/index.php/A_Detailed_Tour_of_the_Cloud4all_Architecture`

3. The Mozilla Foundation: WebSockets, https://developer.mozilla.org/en-US/docs/WebSockets
4. Basman, A., Lewis, C., Clark, C.: To Inclusive Design Through Contextually Extended IoC. In: Videira Lopes, C., Fisher, K. (eds.) Companion to the 26th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2011, pp. 237–256 (2011)
5. Fowler, M.: Inversion of Control Containers and the Dependency Injection pattern, http://martinfowler.com/articles/injection.html
6. Clark, C., et al.: Preferences Framework Overview, http://wiki.gpii.net/index.php/Preferences_Framework_Overview
7. Fluid Project: Fluid Infusion combines JavaScript, CSS, HTML and user-centered design, http://fluidproject.org/products/infusion/
8. Fluid Project: Kettle is a framework for building server-side Web applications using JavaScript and Node.js, http://wiki.fluidproject.org/display/fluid/Kettle
9. Viseur, R.: Fork impacts and motivations in free and open source projects. International Journal of Advanced Computer Science and Applications 3(2) (2012)
10. The Apache Software Foundation: Apache CouchDB is a database that uses JSON for documents, JavaScript for MapReduce queries, and regular HTTP for an API, http://couchdb.apache.org/
11. CouchDB Wiki: Introduction to CouchDB Views, https://wiki.apache.org/couchdb/Introduction_to_CouchDB_views
12. Kiczales, G., et al.: Aspect-oriented programming. ECOOP Springer Verlag (1997), http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.115.8660&rep=rep1&type=pdf
13. C2 Wiki: Declarative Programming, http://c2.com/cgi/wiki?DeclarativeProgramming (accessed January 28, 2014)
14. Lloyd, J.: Practical advantages of declarative programming. In: Joint Conference on Declarative Programming, GULP-PRODE 1994 (1994)
15. Neward, T.: The Vietnam of Computer Science, http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx
16. Joyent: Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications, http://nodejs.org/
17. Holowaychuk, T.J.: Express is a minimal and flexible node.js web application framework, http://expressjs.com/
18. Clark, C., et al.: A Cloud-Scale Architecture for Inclusion: Cloud4all and GPII. In: Assistive Technology: from Research to Practice, AAATE (2013)
19. Graham, P: Beating the Averages, http://paulgraham.com/avg.html
20. The jQuery Framework, http://jquery.com
21. Clark, C., et al.: How the Preferences Framework Works, http://wiki.gpii.net/index.php/How_the_Preferences_Framework_Works
22. uPortal, http://www.jasig.org/uportal
23. Fluid Engage: Transforming museum content for mobile, web, and in-house experiences, http://fluidengage.org/
24. Goodman, C. et al.: Architecting CollectionSpace: A Web-Based Collections Management and Information System for 21st Century Museum Professionals. In: Trant, J., Bearman, D. (eds.) Museums and the Web 2010: Proceedings. Archives & Museum Informatics, Toronto (2010), http://www.archimuse.com/mw2010/papers/goodman/goodman.html