IT@Intel White Paper

Intel IT
Mobile App Development
March 2014

Optimizing the Developer and User Experiences for Cross-Platform Applications

Our proof of concept confirmed that our architecture did support our goals for optimizing cross-platform applications using a single code base.

Executive Overview

Intel IT implemented a new type of software architecture that enables us to develop large-scale, UI-optimized, cross-platform applications using a single code base with maximum code reuse between supported platforms.

With the consumerization of IT and growth of personally owned devices, we must develop enterprise applications that run on multiple devices and platforms (operating systems and browsers). For many use cases, the responsive web design approach is sufficient, where we develop applications using a single code base and render the UI based on screen size. However, some use cases require applications to take advantage of platform-specific features, such as touch, gesture, and native look and feel.

For those use cases, we recognized that we needed to implement a new type of software architecture. Our new architecture has several components:

- Open source libraries. Libraries provide features that we can integrate into an application stack. They also provide functionality specific to a device or platform.
- Model View ViewModel (MVVM). The MVVM design pattern enables us to break an application into layers, separating the business logic from the UI. This enables us to share logic and data layers while rendering views and interaction types based on platform. This modular approach also helps developers to work more efficiently on smaller pieces of code and more effectively on project teams.
- Object-oriented programming (OOP) techniques. OOP techniques provide the JavaScript* objects, inheritance, and encapsulation concepts for our architecture.
- A 70/30 reuse model. This model enables us to reuse 70 percent of the code common to all views while the remaining 30 percent is platform-specific code.

Our proof of concept confirmed that our architecture did support our goals for optimizing cross-platform applications using a single code base. Employing best-known methods, we are implementing our architecture for applications that are complex, interactive, and targeted for multiple platforms. This approach leads to improved developer and user experiences and is well situated to accommodate the devices of the future.

Samion Kuptiev
Lead Developer, Intel IT

Mark Reidman
Software Developer, Intel IT

Contents

IT@INTEL

The IT@Intel program connects IT professionals around the world with their peers inside our organization—sharing lessons learned, methods, and strategies. Our goal is simple: share Intel IT best practices that create business value and make IT a competitive advantage. Visit us today at www.intel.com/IT or contact your local Intel representative if you'd like to learn more.

BACKGROUND

In the past, Intel IT's development efforts relied on a single set of products and technologies to create web applications for a single browser. This approach often led to redundant development efforts when new platforms needed to be supported; we had to develop the same application again for each platform. Now, with the introduction of numerous devices and platforms in the enterprise, it has become critical that we develop cross-platform applications only once and then deploy them on many platforms.

To support cross-platform application development, we primarily use responsive web design (RWD). The RWD approach is aimed at developing applications that optimize the user's viewing experience, whether on a smartphone or desktop monitor. Using RWD, we develop a cross-platform application using the same code base, including all the logic for the various screen sizes in the same project. The application's UI is then rendered based on screen size. This approach enables us to write a program once while providing a "good enough" user experience—one that minimizes resizing and scrolling. However, RWD does not take advantage of the native features or capabilities of a device.

RWD is appropriate for 80–90 percent of use cases: small applications that are meant to be informative, not interactive. Typically, these applications are also targeted for one main platform. If there is more than one platform, all the form factors must be identified at the start of a project, as RWD requires.

The remaining use cases concern applications that are more complex and interactive. These applications are targeted for multiple devices and platforms, some of which may not be known at the start of the project. For those use cases, we also wanted to provide an excellent user experience, one that could take advantage of the device's or platform's capabilities and interaction types, such as a menu that appears in a certain place or a touch-enabling action on a smartphone or tablet. Users expect enterprise applications to provide the same level of experience that they get with consumer applications. With those objectives in mind, we knew that we needed to optimize the UI for the platform we were developing the application for.

Our goal was to build a cross-platform architecture that could effectively address the following:

- Create a UI that is optimized for the interaction type of each platform
- Take advantage of each platform's capabilities
- Maximize code reuse, particularly when combined with the challenge of developing UIs that are optimized for different platforms

We envisioned that implementing such an architecture would lead to excellent developer and user experiences. This architecture would also position us for supporting the applications and devices of the future (see the "Transforming the IT Ecosystem" sidebar).

SOLUTION

As we began to build our software architecture, we started with the well-established approach of decoupling the user interface implementation from the business logic and its underlying data sources. In addition, we extended the business logic layer so it would better support different platforms. This approach would enable us to switch between different UI views, libraries, and data sources to maximize platform capabilities while keeping the same code base.

We chose to break applications into well-structured, component-based layers. Breaking applications down this way separated the application logic from the view, enabling us to focus on developing a reusable code base and implementing a method for changing the UI views depending on the platform.

Our platform-optimized architecture includes several components:

- Open source libraries that serve different purposes, such as navigation and storage
- Model View ViewModel (MVVM) as a design pattern or foundation
- Object-oriented programming (OOP) principles in JavaScript*
- A 70/30 reuse model

The following sections describe our decision making process for including each of these components in our architecture.

Open Source Libraries

To build our architecture, we evaluated both open source frameworks and libraries as candidates for supplying the application building blocks.

We discovered a number of disadvantages with the leading frameworks. The main disadvantage was that we could not reuse UI components for different views, which meant that those frameworks would not support cross-platform optimization. In addition, the leading frameworks were not comprehensive enough to address all the requirements of an application. These frameworks restricted developers to their own implementation and architecture, forcing a rewrite of the applications whenever a new framework was introduced.

Next we evaluated libraries and found that they would support cross-platform optimization. We could use different libraries for building the UI layer on different platforms. To address an application's requirements, libraries could be added into an existing architecture without restricting developers to a particular implementation. To maintain applications, developers could replace libraries with newer versions without needing to rewrite entire applications.

Therefore libraries were the clear choice for the needs of our architecture. We decided to use open source libraries, instead of writing our own, to take advantage of libraries that the worldwide developer community is maintaining and enhancing. We assembled application building blocks by selecting a set of libraries:

- Data binding
- Design patterns
- Storage capabilities
- Common tasks, such as data manipulation and Asynchronous JavaScript and XML (AJAX) communication
- Date and time manipulation
- Navigation
- Mobile UI layer
- Desktop UI layer

Each library we selected served a particular purpose in our architecture.

Model View ViewModel

Next we needed to select an effective design pattern for our architecture. A design pattern describes the high-level structure of an application. We chose the MVVM design pattern, which is largely based on the Model View Controller pattern. The MVVM is targeted for UI development platforms leveraging event-driven programming.

The MVVM breaks the application into three layers: model, view, and view model.

- Model. The model is the lowest layer of the application that provides the content, or business logic. In this layer, we implemented a data source, which is the abstraction layer over a particular source, and a data provider, which is the implementation for a specific source. The model layer enables us to support different platforms using different data sources without affecting the logic.

Transforming the IT Ecosystem to Support the Applications and Devices of the Future

Developing a cross-platform architecture that optimizes both the developer experience and the user experience supports Intel IT's larger initiative to enable enterprise applications to support the devices of today, such as touch-enabled, Intel® architecture-based business Ultrabook™ devices, 2-in-1 devices, and tablets. The cross-platform architecture also supports emerging interaction methods, such as voice, gesture, and perceptual computing.

We have identified five criteria that lead to a better end-user experience:

- Security
- Ease of use
- Platform independence
- Device independence
- Support for emerging devices and interactions

Focusing on these criteria also increases an application developer's productivity and efficiency and contributes to a better overall experience.

HTML5 and the Platform-Optimized Architecture

HTML5 is a key component of Intel IT's mobile application development strategy. The platform-optimized architecture uses HTML5 in the view layer, taking advantage of mobile device capabilities to provide an enhanced user experience. The supported capabilities that users have come to expect from mobile apps include the following:

- **Global Positioning System (GPS) and geolocation.** Apps can activate the GPS and consume geolocation information.
- **Camera.** Apps can activate the device's camera.
- **3D graphics.** Apps can display 3D graphics.
- **Contacts.** Apps can edit and update local contacts.
- **Local storage.** Apps can read and write to local storage.
- **Short message service (SMS).** Apps can initiate SMS messaging.
- **Web intents.** Apps can invoke the functionality of another application.

- **View.** The view layer presents the data and UI controls on the screen using HTML5 and CSS3 (see the "HTML5 and the Platform-Optimized Architecture" sidebar). The view layer is logic free and enables us to add UI libraries to optimize the view for a specific platform.
- **View model.** The view model layer converts values between the model layer and view layer, which allows the data objects from the model layer to be managed and displayed in the view layer. The view model layer uses the data binding mechanism to glue together the logic and the view, enabling us to create functions for specific devices. For example, we can specify how an application will respond to a click event on a desktop, or a touch event on a smartphone or tablet.

Figure 1 shows an overview of an application using the MVVM design pattern.

Overall, using the MVVM provides many benefits. It provides a complete separation between the UI and business logic, enabling us to layer the application and swap out components as needed. For example, we can discard one view, bring in another one, and bind it to the same logic. We can also create separate modules with functions that are common for all views and then add more functions that are relevant for only specific types of devices.

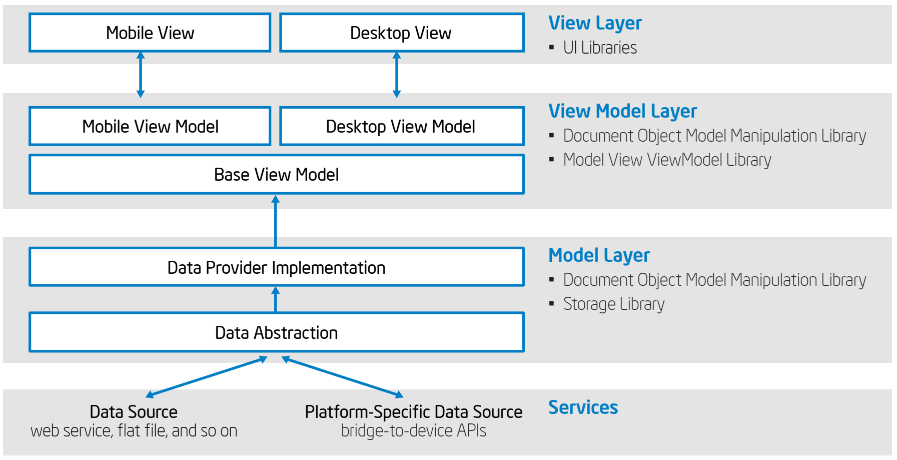
The separation that the MVVM provides helps developers create reusable code, particularly in the logic and model layers. This separation also helps the development team structure the project between developers and module owners more effectively. For example, with the MVVM approach, each developer can work on a single set of modules or files. These modules are designed as separate, pluggable components that are each maintained with its own set of files. This mode of work prevents collisions and promotes effective design patterns.

Object-Oriented Programming Principles

Another piece of our architecture is the application of OOP principles. To use libraries and the MVVM design pattern, we needed to understand and apply OOP principles and interpret the relevant ones into JavaScript.

We used the following OOP JavaScript principles:

- **Objects** are abstract ideas that are modeled in an application and perform all the work. Each object has a state and behavior. In the MVVM, objects define the application data (models), UI look and feel (views), and application behavior (view models).



Mobile View
Desktop View
View Layer
▪ UI Libraries
Mobile View Model
Desktop View Model
Base View Model
View Model Layer
▪ Document Object Model Manipulation Library
▪ Model View ViewModel Library
Data Provider Implementation
Data Abstraction
Model Layer
▪ Document Object Model Manipulation Library
▪ Storage Library
Data Source
web service, flat file, and so on
Platform-Specific Data Source
bridge-to-device APIs
Services

Figure 1. The Model View ViewModel design pattern separates the UI implementation from the model layer's business logic and its underlying data sources.

- **Encapsulation** is the concept of enclosing an object's methods and properties within that object so that they are hidden from the rest of the application. Encapsulation enables developers to abstract a specific set of functionalities on specific objects from the rest of the application.
- **Inheritance** is the passing on of methods and properties from a parent object. This concept is important for creating a view model that contains code common to all platforms, which is discussed in the next section.

All three principles are important because they support our goals for building applications with reusable, self-contained code.

70/30 Reuse Model

Our exploration showed that the data binding mechanism alone (discussed earlier in the "Model View ViewModel" section) was not enough to enable switching between different UI views for different platforms. The primary challenge was how to encapsulate the different interaction types, derived from the form factor, on a single layer. To address this challenge, we implemented a 70/30 reuse model. This reuse model creates two distinct view models:

- A shared base view model that defines the interactions common to all platforms (70 percent of code base)
- An extended view model that is specific to the platform (30 percent of code base)

The OOP inheritance principle enables applications to inherit from the base view model. This inheritance along with the declarative binding between the view and the view model make the support of an optimized UI possible, as shown in Figure 2.

For example, our Virtual Assistant application includes an employee search feature (see Figure 3). The shared base view model executes the search functionality with paging, while the mobile view model controls how the results display on a smartphone, and the desktop view model controls how the results display on a desktop.

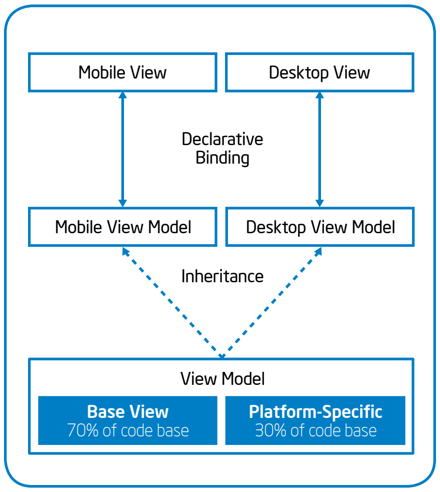
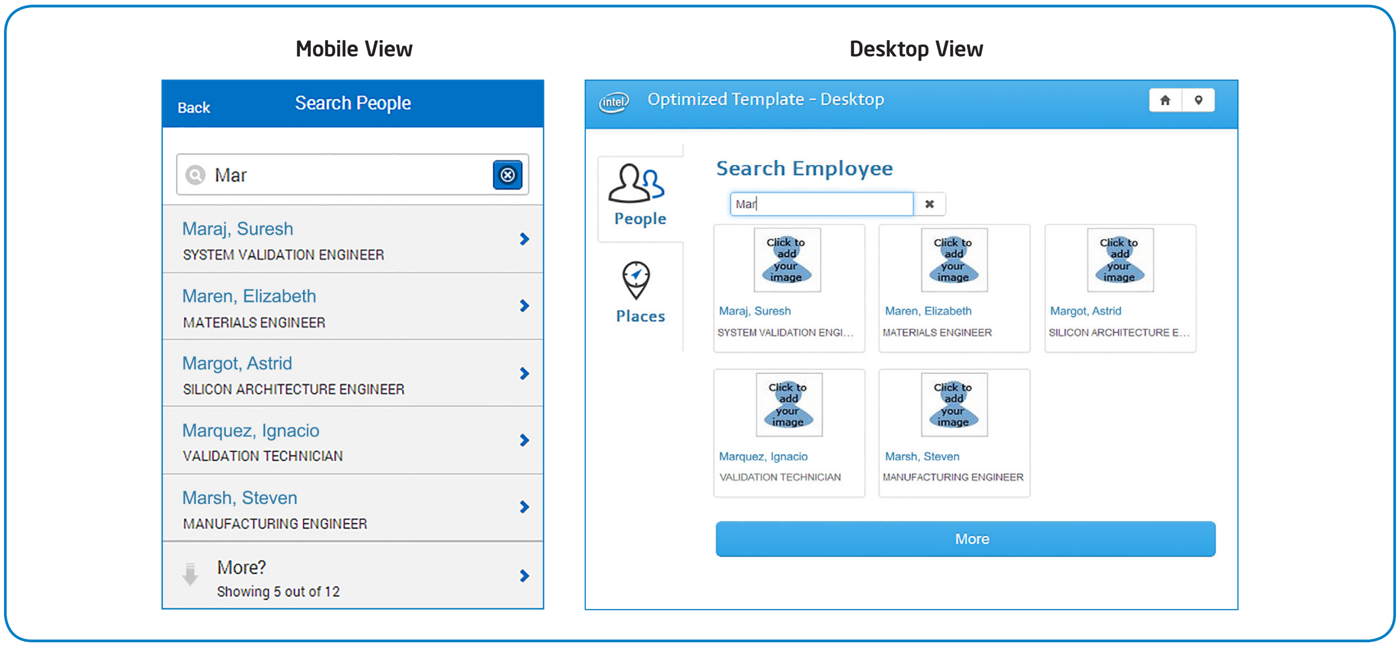


Figure 2. Inheritance and declarative binding enable views to be optimized for a particular platform.



Maraj, Suresh	SYSTEM VALIDATION ENGINEER
Maren, Elizabeth	MATERIALS ENGINEER
Margot, Astrid	SILICON ARCHITECTURE ENGINEER
Marquez, Ignacio	VALIDATION TECHNICIAN
Marsh, Steven	MANUFACTURING ENGINEER

Figure 3. The Virtual Assistant application displays search results differently, depending on whether they are viewed from a mobile device or a desktop.

The reuse model extends beyond a single project or application. Table 1 lists how we can achieve code reuse by application layer, both in a single project and among other projects. The base view model (70 percent) can be reused in the same project by all views. Only the extended view model (30 percent) and the view layer are platform-specific and cannot be reused.

All objects in the model layer (data object, provider, and source) can be reused in other web-based projects.

BEST-KNOWN METHODS

With our architecture in place, we were ready to conduct a proof of concept (PoC) to help us refine the architecture and generate best-known methods to pass on to developers.

To test our platform-optimized architecture using the PoC, we developed the Campus Information application for the Microsoft Windows* 8 platform. This application, which was already supported on various smartphone devices, used HTML5 geolocation to provide features based on the device location. Our challenge was to reuse as much existing code as possible while implementing an MVVM structure with cross-platform UI logic layers and a platform-specific layer. Equally important was the need to deliver the user experience that Windows 8 users would expect.

First, we worked on the model layer of the Campus Information application, which represented the data coming from the back-end systems through web services. Next, we worked on the base view model that would contain the interactions common to all platforms. We built each layer so it could stand alone with clear and minimal dependencies. This enabled us to reuse these layers, focus on the Windows 8 user experience, and deliver the application in a shorter period of time.

Overall, the results were successful. Moving from a monolithic structure to a layered, component-based application led to increased code quality, maintainability, and extensibility. The structure also supports the delivery of a user experience tailored for a specific device.

The PoC also enabled us to identify several best-known methods to help developers use our platform-optimized architecture:

- **Table of requirements.** We created a table for developers to help them choose the architecture that would best meet their use case requirements (see Table 2).
- **Project templates.** We created project templates that contain sets of JavaScript libraries optimized for different platform types. Developers can use the templates to create an entire skeleton for an application.
- **Internal social coding site.** We used our internal social coding site to share our platform-optimized templates and encourage developers to use this approach. Developers can access the latest versions of our templates on this site.

Learning our architecture and applying it effectively takes time and has a learning curve. These best-known methods enable developers to identify when to use our architecture and give them a good starting point for applying the architecture to their application development.

Table 1. Code Reuse by Application Layer

Application Layer	Object	Code Reuse
View	UI view	Platform-specific
View Model	Extended view model (30 percent of code base)	Platform-specific
	Base view model (70 percent of code base)	All views can reuse the code in the same project
Model	▪ Data object ▪ Data provider ▪ Data source	Any web-based project can reuse the code

Table 2. Choosing between Responsive Web Design and Platform-Optimized Architecture

	Responsive Web Design	Platform-Optimized Architecture
Use Case	Default architecture	When device features are required as part of an enterprise mobility application framework
Multiplatform	Yes	Yes
Rendering Target	Web	Web browser/hybrid app
Device Features	Some	All (when in an enterprise mobility application framework)
Code Base	Single code base	Multiple code base

CONCLUSION

Our platform-optimized architecture is built on cross-platform application layers that consist of libraries, the MVVM design pattern, a 70/30 reuse model, and OOP principles. This architecture is designed to provide a device-tailored user experience for the supported platforms. More than just adjusting for a device's screen size, applications take full advantage of a device's features and interaction modes, behaving in a manner consistent with what the user is expecting on that device.

Additionally, the developer's job is streamlined: code is reusable within a project, and some code is reusable among all web-based projects. The architecture ensures a more efficient way to develop and maintain applications, easing development when additional platforms are supported. It also provides a structure that enables teams to divide up work more effectively.

Looking to the future, our architecture is well-suited to meet the introduction of more devices, platforms, and interaction types. As more applications are expected to take advantage of each platform's capabilities, the use of this architecture may become more widespread and vital to our strategy for cross-platform application development.

RELATED INFORMATION

Visit www.intel.com/it to find content on related topics:

- "Accelerating Our Path to Multi-Platform Benefits"
- "Building a Mobile Application Development Framework"
- "Implementing a Cross-Platform Enterprise Mobile Application Framework"
- "Learnings from Early Native Apps Improve HTML5 and Hybrid Apps"

ACRONYMS

AJAX	Asynchronous JavaScript and XML
GPS	Global Positioning System
MVVM	Model View ViewModel
OOP	object-oriented programming
PoC	proof of concept
RWD	responsive web design
SMS	short message service

For more information on Intel IT best practices, visit www.intel.com/IT.

intel® Look Inside.™