LAYERED ARCHITECTURE DELIVERS MORE RELIABLE AUTOMOTIVE APPLICATIONS, FASTER

By Roland Krause,
Integrated Computer Solutions

Today's consumers are used to a rapid pace of innovation. Mobile and web apps can be developed and updated quickly so there's always something new to appeal to consumers—a reality that heightens their expectations. Unfortunately for automotive manufacturers, traditional approaches to developing in-vehicle-infotainment (IVI) systems are often characterized by long development times. And IVI updates can take months if not years. The result? Automotive head units, expensive and time-consuming to develop, are somewhat outdated by the time they first reach the market.

Unimpressed, consumers often spurn these systems, putting OEMs in a precarious position. So how can automotive OEMs find market acceptance? How do they and their Tier-1 providers quickly build appealing, competitive and secure infotainment systems without relying on mobile technologies that don't meet automotive safety and security requirements? An effective solution is to rely on a layered software architecture.

With a layered software architecture, developers create independent components that address a specific part of the functionality of the whole system. These components communicate through well-defined and stable interfaces that allow for high-performing implementations. This approach necessitates not only understanding requirements but also completely understanding the user experience (UX). The most successful software projects lead with UX by allowing UX design to guide and dictate all steps of the implementation process.

Lead with UX

One important requirement for modern applications is that they look intuitively beautiful with meaningful animations and simple, easy-to-recognize graphics and iconography. That makes applications easy to use, recognizable and appealing to consumers. This is especially important for products marketed worldwide as user interfaces (UI) must be easily translatable into different languages and adaptable to different cultures.

And often they need to function on different devices with a variety of screen sizes and orientations. Modern UI toolkits have the capabilities to deal with these challenges. HTML5 with CSS, iOS Swift or the Qt toolkit are very popular choices for cleanly building a software layer that addresses the look and feel of the application.

Whichever toolkit is used, leading an automotive project with UX design makes all this possible—and easier than with more-traditional approaches. Modern UX design supports intuitive, practical workflows that allow users to quickly do what they need to do without having to actually "learn" how an application works by studying instruction manuals. This is often achieved by organizing applications with simple hierarchies, hiding expert-level functionality behind access levels, and taking lessons from the way popular mobile-phone applications flow.

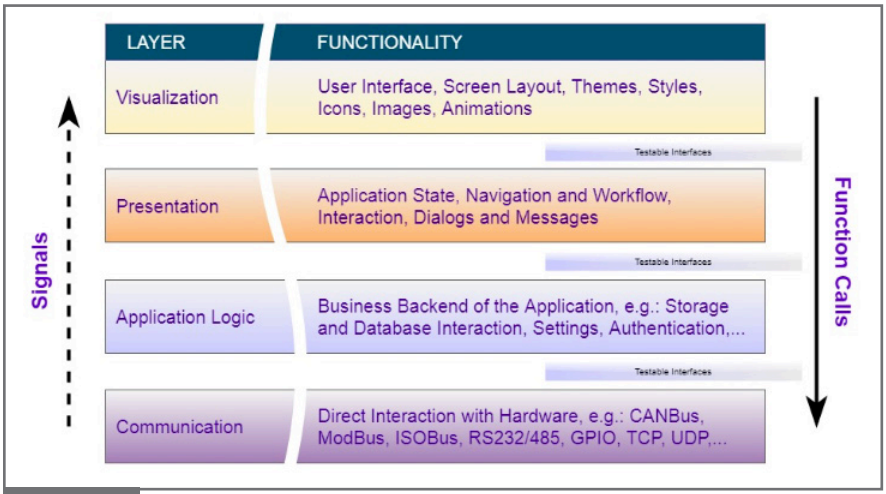

FIGURE 1 Four-Layer Software Architecture

The implementation of this workflow requires a software architecture that provides a programming backbone to the "facade" of the topmost layer.

To make the complex look and feel simple, a systematic approach must be chosen to allow the user to switch between pages or groups of items in an application, navigate back, reach the "settings," be interrupted and then easily find their way back to focus. Using a state-machine architecture allows for the flexibility needed while preserving the required robustness and testability.

UX-First Layered Approach

Following a UX-first, layered-architecture approach is one way to ease development of an auto IVI that appeals aesthetically to consumers while providing sought-after functionality. This type of approach, which includes layers on visualization, presentation, business logic and communication, relies on components to ease the process.

For instance, the visualization and presentation layers provide the user with a way of interacting with the application. But there is still the implementation of the actual functionality, for example while the user interface wants to display a list of contacts in an address book, this data must be retrieved by the application from storage, be it in a database, on a connected phone, from a simple file, a network connection or elsewhere.

There is a certain block of logic that all applications must implement, the "business logic" layer. This is where, for instance, the algorithms of a supplier in the agricultural industry determine whether and where the level of product sprayed on a field can be optimized. Organizing this logic and decoupling it from the presentation and visualization layers is of highest importance for the effectiveness of the application-development process. For companies aiming to provide superior user experiences, this layer is where they have the greatest potential for differentiation.

During the development process, certain components that require specialized hardware or long-running processes must be "mocked out"—replaced by simulated functionality. A system based on interfaces and plugins can be deployed to achieve this goal where the presentation layers of the application access the business logic through a well-determined set of interfaces, and hence isolate from changes or the ongoing development process in these layers. Usability of an application can then be tested with the help of mocked-up data and simulation. This process has been shown to lead to more cost-effective and more usable, complete and better applications.

That's where the communication layer comes in.

To achieve the aforementioned goals, nearly all modern applications communicate in some way. Automotive IVIs often use a CANBus layer to determine the state of various vehicle subsystems, while medical devices have verified and validated communication libraries that could, for example, make a robot arm perform a highly precise movement during a surgery.

In addition, nearly all applications need to be updatable, able to deliver information about their use for aggregation and marketing purposes, or simply need online database access to be fully useful. Grouping this part of the application's logic into a communication layer is therefore a logical choice and brings many advantages with it.

Communication channels can be exchanged without having to rewrite the application. Safety and security can be focused and implemented in few places, which minimizes the attack surface. And layered implementation allows for optimizations that keep an application "alive" while important processes happen in the background. Non-blocking communication patterns are much easier to achieve when the architecture is built on a communication layer.

Sample 4-Layered Architecture

Here's an overview of Integrated Computer Solutions' version of a layered approach showing clear separation of functionality suitable especially for all projects that must comply with safety regulations. Engineering in this way enforces clean and maintainable software ideally suited for integration and unit testing.

The four independent layers of this architecture are defined by testable and mockable interfaces. Loose coupling is enforced for all communication from the lower to the upper layers. This guarantees independence of the backend from the frontend. So-called signals

can be implemented easily using many common frameworks. All top layers can by convention make calls to the layers below through APIs and interfaces. This creates tight coupling top to bottom where it is important to ensure deterministic communication patterns and execution of application logic.

These are the roles of the individual layers:

1. *The Visualization Layer* is responsible for everything the user sees and interacts with on screen. It is responsible for displaying all text, images, icons, themes, styles, animations and more.

   The layer can be implemented using many different technologies. (We often recommend the use of the Qt framework and in that case this layer would be implemented using Qt's QtQuick module.)

2. *The Presentation Layer* maintains application and user interface states. It is responsible for storing values shown on the screen. For example, a value set on a dial is stored in the presentation layer, but is displayed in the visualization layer. The presentation layer implements the application's workflow and all interaction of the workflow with the backend (e.g. if an error message from the backend arrives, the presentation layer logic displays the dialog on the screen that contains the error message).

3. *The Application Logic Layer* implements the business logic of the application. It is responsible for data storage, database interaction, handling of events from the backend and interaction with the frontend. This layer handles the "domain knowledge" needed for the implementation of the user interface. For the current application this layer will, for instance, implement the business logic of "Test Runs," "View Results," "User Management" and nearly all other tasks defined in the Software Requirements Specification.

4. *The Communications Layer* implements the communication with hardware devices, networks, or other systems. Hardware partners will often be responsible for providing interfaces for the actual hardware and will implement the communication routines necessary using mutually developed APIs. This layer specifically allows the use of mockable interfaces that can be tested independently before the actual hardware is completed and available.

This well-defined architecture facilitates use of reusable code, and decoupling the layers enables the division of work between UX designers, UI implementation, business logic providers and hardware partners. It further allows easy assignment of tasks to the right skill set. Taken as a whole, these characteristics greatly reduce project development risk. Further, this layered architecture approach allows companies to build prototypes much faster and iterate workflow with UX designers immediately.

To keep pace with consumer demand, automakers must accelerate the speed at which they build appealing, secure infotainment systems. Relying on a layered software architecture that incorporates reusable code rather than leaning on less well-defined mobile technologies can dramatically shrink time to market so automakers can speed across the finish line in record time.

Roland Krause is the Engineering Director for Integrated Computer Solutions, Inc. (ICS). He is the software architect behind the company's Automotive offerings and has more than a decade experience developing large-scale and embedded software systems. He received Master's and Doctoral degrees in engineering from the Universität Dortmund and is fluent in both German and English.