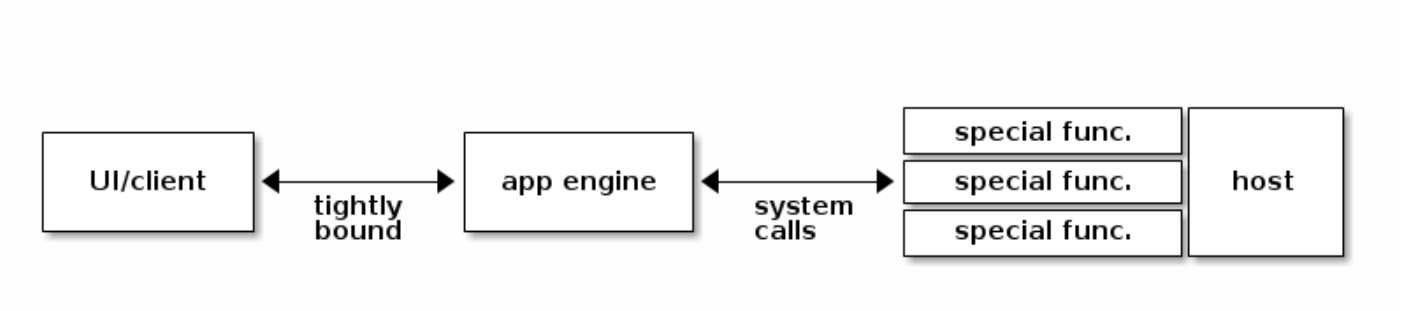
An Alternative Architecture for Hybrid Applications

Hybrid applications seem to be gaining traction now. In what follows, I'd like to present thoughts on an alternative to the emerging standard hybrid app architecture.

The conventional hybrid architecture



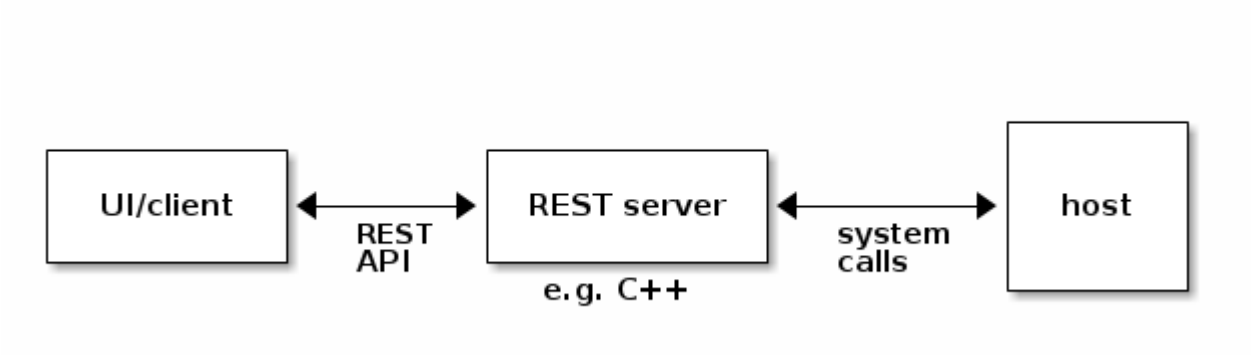
In this model, which to the best of my knowledge is used by Electron, NW.js and others, the user interface is rendered as HTML using whatever HTML, CSS, and front-end JS frameworks you desire. The overriding goal of this architecture is to use Web technologies for the UI.

The UI is tightly bound in a one-to-one relationship with the app engine.[1)] The app engine is implemented with Web back-end technology, typically Node.js. The app engine makes system calls through the engine's baked-in features or through generic `child_process.exec()`-like calls. This means custom and platform-specific behaviors that the app may require will need to be implemented as external `child_process.exec()` callable units.

This architecture does a good job of leveraging Web technologies to create secure, conventional desktop apps. In addition, frameworks like Electron have matured to the point that developing hybrid apps that use many desktop app conventions is relatively easy.

A REST-based hybrid architecture

One alternative to the conventional hybrid approach is a REST-based architecture. A REST-based approach requires more carefully considered design but offers the potential for greater flexibility.



In this model, the tightly bound user↔app engine connection is replaced by a REST API. The app engine becomes a REST server, possibly embellished with whatever superpowers are needed for accessing host resources, and the UI is supported through a client. Because the app interface is API driven, any REST client technology can be used for the interface, including HTML/CSS/JS clients, native mobile clients, terminal clients, etc. In addition, the client need not be local, which makes remote-controlled apps almost trivial to implement. As with any network-based communication, adequate measures must be taken to assure secure communication with the REST server.

The REST server can be implemented in any language on which a REST server can be built. Thus the “app” language can be one that also supports the necessary system manipulations the application requires (file access, etc.) or is otherwise best-suited to the app's requirements.

One obvious requirement is that the language chosen for the REST server language must be able to run on the target host platform. This isn't a significant issue for desktop deploymen. However, it does currently present a problem for mobile deployment as few mobile platforms provide native support for more than one blessed development language.

Some development notes.

1)

I'm using “app engine” generically here, not as a reference to Google's AppEngine.