Developing and Using APIs in System Design

James Conners and Mason Kortz, PAL-LTER/CCE-LTER

An Application Programming Interface, or API, is a set of code that abstracts complex operations into a simple interface. The interface provided by an API is not intended for use by end users, but rather by developers, as one way to incorporate existing functionality into an application while minimizing the amount of learning and coding necessary. Well-known APIs include the Google Maps API, which abstracts complex database calls and browser rendering into relatively simple Javascript functions, or the many database APIs that abstract the opening of sockets, sending of queries, and structuring of results into a small number of calls in your favorite language. In early 2007, the Ocean Informatics team began exploring APIs for various needs such as in-browser plotting, database abstraction, and XML transformation. As our in-house use of APIs grew, we began to organize our own code into APIs, and in November of 2007 we finished created a personnel management database schema with an accompanying API definition (and PHP implementation). Subsequently, we have created APIs for terminology lists and media collections, and have begun work on an API for the LTER Unit Repository.

Using APIs as part of the development process provides several benefits. By utilizing existing code, developers save time that would be spent generating and testing thousands of lines code. Some APIs provide interfaces into technologies so complex that it could take years for an individual or small group to recreate them - as is the case with the Google Maps API or the Microsoft DirectX API. In this way, APIs enable the development of richer applications on shorter time scales. Another benefit of using APIs is that the developer only needs to learn the interface into the technology being supported, not the technology itself. Database interface APIs, for example, allow developers to create networked database applications without knowing the ins and outs of sockets, message headers, and responses. The incorporation of high level modularity results in applications that are more easily shared and extended because of their transparent compartmentalization of logic and functionality.

In addition to the benefits of using existing programming interfaces for application development, there are also reasons for using APIs as a local development model - producing local work based on core code bases and storage resources with associated APIs. One possible benefit of this model is the resulting implicit community standards of development practices. API-oriented development begins with a defined set of capabilities required for an interface to be useful. This set of capabilities includes both immediately necessary functions and functions which seem likely to be useful in the future. Because the goal of API development is code reuse over time by developers within or across communities, documentation and clear code are encouraged, being primary factors in the useability of a programming interface. Having a well-defined, documented interface into a resource allows developers throughout the community to utilize them without concern for back-end changes deprecating their code. These factors result in a community of developers producing a well-documented and structured core of shareable works that provide enough stability for implementations built upon them to maintain a degree of flexibility and be developed with agility. These core resources are consequently well suited for sharing within or across development communities.

API-oriented development practices have both positive and negative ramifications. One such trade-off is the loss of rapid initial development periods in favor of more efficient program implementations in the future. More time in the beginning phases of development is required, but the benefit of this is a facilitative code interface that provides a substantial base structure on which to build additional applications, each providing access to a shared technology. An example would be an API developed for a media gallery backend that stores and queries photos, movies and documents. Implementation of an application using the media gallery technology begins with a large part of the development already completed. The application development cycle is limited to providing end-user access to the existing capabilities of the API, cutting development time substantially. Implementations can be across platforms, each with a different look and purpose, while all relying on the same code base. The benefit of this trade-off, then, depends on the anticipation of multiple applications using a common code base.

Code abstraction is another implication of API development, with the granularity of program control strongly influenced or restricted by the design of the programming interface, establishing a lower bound for optimization. APIs may provide interfaces to resources that include modularized storage which may affect performance of data retrieval or enforce weak relational references across databases. Additionally, abstracted code is less fluid than procedural code, as the API developer is committed to encapsulating the functionality of a code base in a set of established functions. Abstracted code is also much easier to read, and an abstract interface into a complex code base has a much shorted learning period than the code base itself. APIs are more easily adopted into new or existing development cycles, whether within or outside of the community that developed the API.

API-based development also entails certain changes in development practices. A longer period of prototyping before coding begins is beneficial to an API, as it allows the developer to consider future uses of the interface beyond immediate needs, and to discuss the requirements of the interface with other developers who may use it. An organized code repository, with agreed-upon standards for quality and completeness, increases the mobility of APIs, working towards the goal of community reuse. As emphasis on code reuse increases, so does emphasis on documentation - because abstracted code is not self-describing, time must be taken to describe it clearly for it to be useful beyond the immediate developer.

The decision to develop an API frequently occurs when a technology or resource exists from which one or more communities would benefit. Providing a programming interface into this resource encourages not only the sharing of code but the usage of a common technology that creates in effect a coordinating mechanism that makes more solid the basis for collaboration. Efforts across communities combine where they would normally conflict. Though not all undertakings' criteria justify the need to develop with such a scope, we have found that many of our local projects have benefited substantially from an API-oriented development model, both functionally and organizationally.