section, applications themselves become a data service of a sort, supplying the content for Facebook to display under *http://apps.facebook.com*. A URL such as *http://apps.facebook.com/fettermansbooks/...* would no longer map to Facebook-generated data, logic, and display, but would query the service at *http://fettermansbooks.com* to generate the application's content.

We must simultaneously keep in mind our assets and our constraints. On one hand, we have a highly trafficked social system for users to discover external content, and a great deal of social data to augment such social applications. On the other hand, requests need to originate on the social site (Facebook), consume the application as a service, and render content such as HTML, JavaScript, and CSS, all without violating the privacy or expectations of users on Facebook.

First, we show some *incorrect* ways to attempt this.

Applications on Facebook: Directly Rendering HTML, CSS, and JS

Imagine an external application's configuration now include two new fields named `application_name` and `callback_url`. By entering in a name like "fettermansbooks" and a URL like *http://fettermansbooks.com/fbapp/* respectively, *http://fettermanbooks.com* declares that it will service user requests to URLs like *http://apps.facebook.com/fettermansbooks/PATH?QUERY_STRING* on its own servers, at *http://fettermansbooks.com/fbapp/PATH?QUERY_STRING*.

A request to *http://apps.facebook.com/fettermansbooks/...* then simply fetches the HTML, JS, and CSS contents on the application servers and displays this as the main content of the page on Facebook. This renders the external site as essentially an *HTML web service*.

This changes the n-tier model of an application significantly. Earlier, a stack consuming Facebook's content as a data service served direct requests to *http://fettermansbooks.com*. Now, the application maintains a tree under its web root that itself provides an HTML service. Facebook obtains content from an online request to this new application service (which may, in turn, consume Facebook's data services), wraps it in the usual Facebook site navigation elements, and displays it to the user.

However, if Facebook renders an application's HTML, JavaScript, or CSS directly in its pages, this allows the application to completely violate the user's expectation of the more controlled experience on *http://facebook.com*, and opens the site and its users up to all kinds of nasty security attacks. Allowing direct customization of markup and script from outside users is almost never a good idea. In fact, code or script injection is usually the *goal* of attackers, so it's not much of a feature.

Plus: no new data! Although this forms the basis of how an application's stack changes, this solution solves neither of our product problems fully.