What does the Common Navigator Framework help me do?

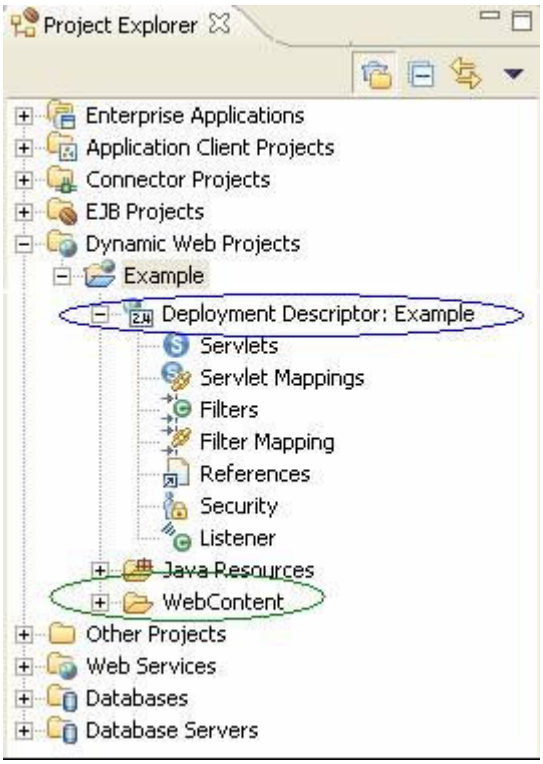
A framework should *help* you do something, not *make* you do something. In the following article, I'll go over the scenarios that motivated the creation of a generic, viewer integration framework. I'll also discuss a little of the background of the framework and how it evolved to an open source solution for the Eclipse Platform.

The details of this article roughly follow the flow from the presentation given at EclipseCon 2006.

Use case 1: Expose various types of information in a viewer to provide a cohesive user experience with editors.

One of the greatest features of Eclipse is its inherent extensibility. However, in some cases, the way in which the extensibility of Eclipse is utilized can come at the cost of the user experience. One problem that many products have come across when extending Eclipse is viewer *explosion*, where many views for different tasks are created to satisfy the requirements of each individual component. Depending on the overall product architecture, the views may sometimes duplicate information (such as each view rendering the project and its resources, plus one other piece of value-added content for each project), or alternatively the views may render just their data, which means that if the user is exploiting function from multiple components within the product, s/he may have to keep multiple views open.

The pattern that many components teams follow tends to be a viewer that exposes a custom model (like a J2EE Deployment Descriptor, as in the Web Tools Platform) plus the resource or Java(tm) representation of the project.

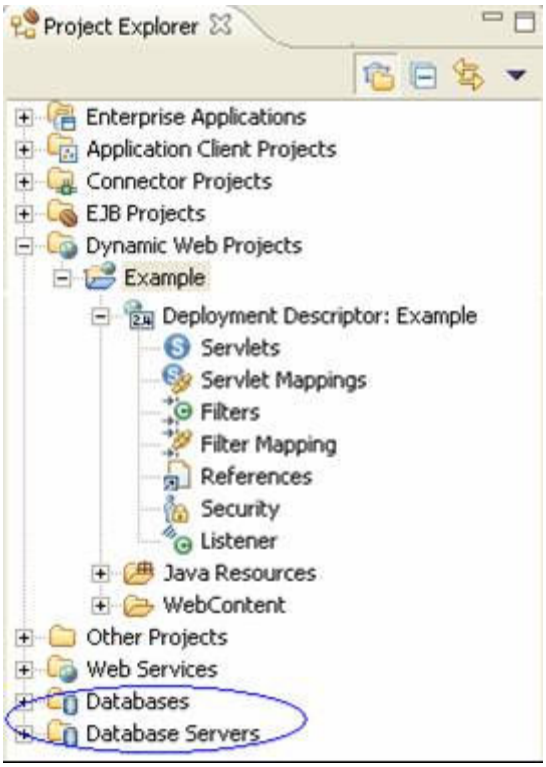


It turns out that this pattern is really driven by integrators incorporating the editor model into their viewers. Often, the extra content in a viewer is really a representation of some model that an editor can operate on, which is exposed to the user to make it easier for them to manipulate or navigate. So the first requirement of fight club .. err the CNF is:

Requirement: Support a single viewer for all editor model integrators.

Use Case 2: Expose model elements that are not tied directly to the workspace.

Not all viewers follow the editor model pattern; some viewers follow the pattern of incorporating data from outside the workspace environment to make it easier for users to have a "one-stop" shop for all of their development needs. The database extensions in WTP are an example of this:



Now merging all of the content of these views in one place can overwhelm the user, so the CNF must allow users to choose what is relevant to them.

There are two requirements derived from this scenario:

Requirement: Allow non-resource driven model content.

Requirement: Allow users to choose what they want to see in their integrated viewer.

Use Case 3: Open Source Platform must allow integration from Commercial Products.

One of the biggest value propositions of Eclipse is for vendors to pool resources to produce the commodity

layers; then vendors can differentiate themselves to provide value-added content based on their customers' needs. The Web Tools Platform is one of the first big examples where many different kinds of viewer content would be required for end-users, both from standard and proprietary models, and therefore vendors must not be restricted from extending the primary navigational viewer.

A similar problem exists on the level of commercial products, where value-added content from Independent Service Vendors (ISVs) must be able to integrate seamlessly to enhance the user experience. Often, the integration paths for ISVs are not as well documented as an open source platform, so providing a solution on the open source layer allows the ISVs to use the same consistent, public, documented API that the community has helped hone.

So the general use case here is to allow extensibility across layers of software; downstream layers must be able to integrate seamlessly to maintain a cohesive user experience.

Requirement: Allow Platforms (open source and commercial) to reuse a consistent viewer integration framework and API.

So in summary, there are four basic requirements of the CNF:

1. *Support a single viewer for all editor model integrators.*
2. *Allow non-resource driven model content.*
3. *Allow users to choose what they want to see in their integrated viewer.*
4. *Allow Platforms (open source and commercial) to reuse a consistent viewer integration framework and API.*

In addition to the requirements above, there were a few goals driven from experience with developing Eclipse viewers from release to release:

1. *Enhanced reusability*. Clients should be able to re-use Platform content (Resource model, Java(tm) model, etc) with little or no code; and clients should be able to define their own viewers an "absorb" content easily.
2. *Mitigate client reaction from release to release.* Eclipse continues to evolve at all levels as an effective Platform. Often enhancements are made at the lower levels (such as the enhanced working set support from JDT in 3.1). For downstream integrators to adopt this functionality, it often requires an expensive porting or coding effort, in addition to whatever value-added content is slated for that product's own release. The CNF should make it easier for clients to absorb these enhancements without coding efforts.

The CNF began as product solution in Rational® Application Developer (RAD) v6.0. It was originally derived from the General Purpose Navigator proposal for Eclipse 3.0. The GPN was a great start, but much of the original code has been re-written; however some of the concepts still remain.

When IBM Rational decided to contribute some of the lower integration layers of RAD v6.0 to the then newly formed Eclipse Web Tools Platform project, the CNF was included in the contribution. Product solutions often have very different focuses from Platform solutions, as the measure of value in a product solution is on the user experience, not on the ease of extendibility. Some initial API definition was done in the context of WTP, but there was uncertainty at the time about the proper home for the CNF. Like many of the frameworks in WTP, it is not specific to any particular domain, and solves a general problem required for products built on Eclipse. Therefore, it was expected that the proper home for the CNF was in the Platform. At the time, it wasn't clear that the Platform would be willing to accept the CNF or if another version or solution would present itself, so the effort invested to harden the CNF API in WTP was minimal.

In Eclipse 3.2, it was agreed that the Platform would adopt the CNF, with the proper API finalization that was required. I've had requests to provide some thoughts and feedback on what it takes to turn a product solution into a platform solution, and I hope to address that in a future post.

So in a nutshell, the CNF is designed to help you integrate document models into an navigator experience, integrate content that isn't specific to the workbench, allow you to absorb other content seamlessly (in particular resource and Java(tm) models), and mitigate your expense in time and effort to absorb incremental enhancements from release to release from layers beneath you. The CNF began as a product solution for a general problem in Rational Application Developer v6.0, and has been contributed to the

open source Eclipse Platform in 3.2 to allow the community to better integrate their navigational viewers and provide a more cohesive user experience across software layers and products.

ABOUT THE AUTHOR

MICHAEL ELDER
RESEARCH TRIANGLE PARK, NORTH CAROLINA, UNITED STATES

Michael has been a Java developer since 1999 and currently contributes to Eclipse in open source and commercial venues. Most recently, Michael has contributed the Common Navigator Framework (CNF) to Eclipse 3.2. Prior to that, he helped design several of the frameworks and API in the Web Tools Platform (WTP), and continues to contribute fixes to WTP through the open source process. Michael is currently part of the Services Oriented Architectures (SOA) Tools Team for IBM Rational® based in Research Triangle Park, NC.