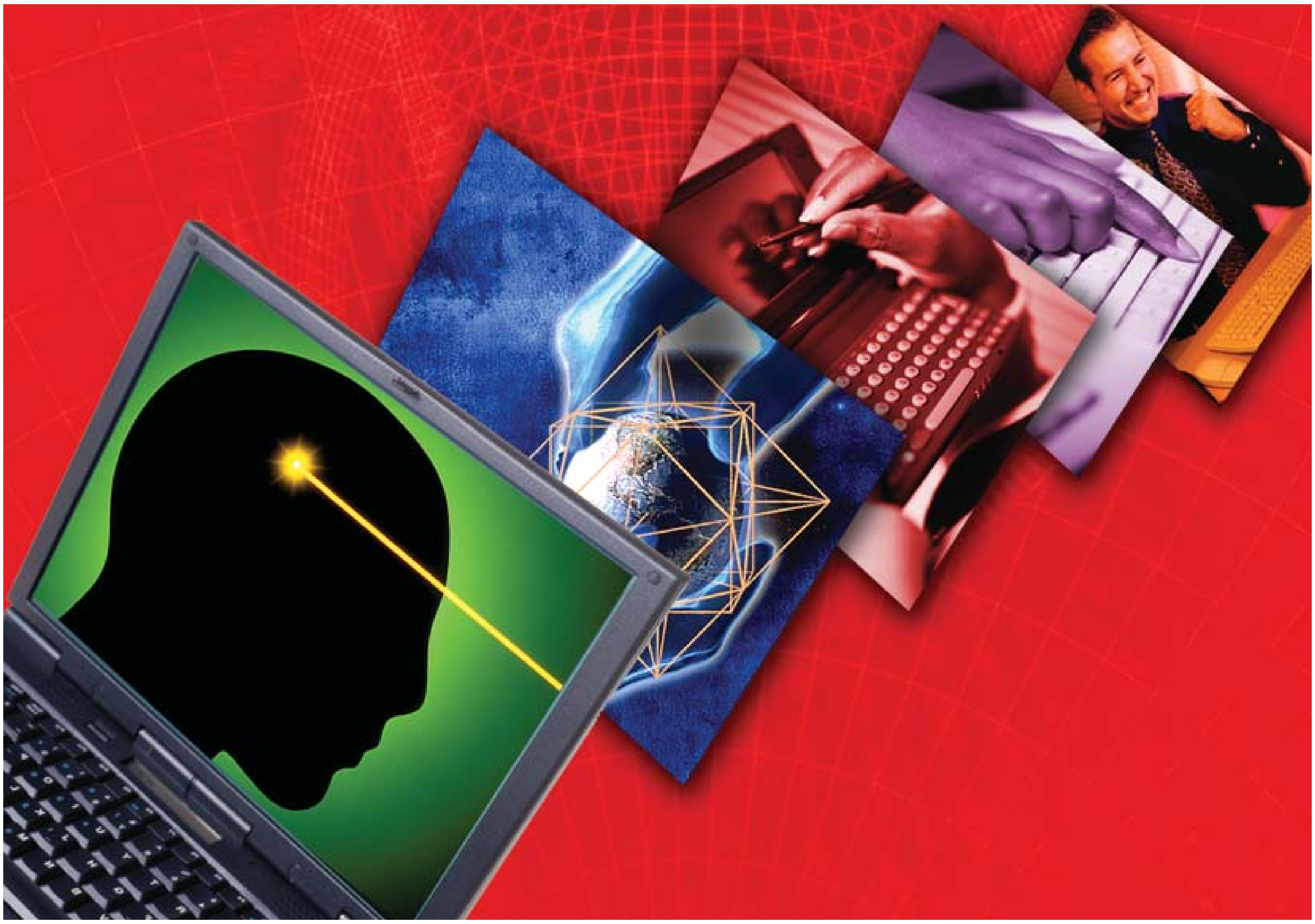


Capturing Developer Intent in Eclipse via Application Factories

By Ravi Kumar

One of the most persistent problems faced by all developers is how to transfer knowledge between team members. For example, how does an architect indicate intent in code for subsequent developers to build on? How do members who specialize in implementing the Web interface of an application know where to fix a problem in the persistence layer? And where, if anywhere, are the code snippets and reusable components that the member assumes they have? Application Factories will enable senior developers to communicate intent, capture instructions and recommendations, and point to resources—all in a single, simple tool.

The fact of the matter is that as applications become more complex and involve increasing numbers of layers and frameworks, developers are obliged to become specialists with detailed, but tightly circumscribed knowledge of a small cadre of technologies that inhibits expertise or even exposure to technologies in other parts of the same application. The result is that knowledge becomes increasingly silo'ed and knowledge transfer is difficult and, at times, nearly impossible. Even within the same subteam, members can find it difficult to leave useful bread crumb trails for developers who will come behind them; likewise, they have difficulty mining the knowledge of existing team members unless they work geographically close enough and can ask in person.

As projects become more complex, this problem emerges as a gating factor to timely product delivery, correct functionality, responsiveness to changing requirements, and crucially the ability to promptly and properly resolve defects. Despite its universality, this problem is rarely discussed as such and, until now, has not been addressed directly by tools.

CodeGear, an independent developer tools organization formed from Borland's Developer Tools Group, will soon ship a new IDE technology called Application Factories, which enable senior developers to communicate intent, capture instructions and recommendations, and point to resources—all in a single, simple tool. The first shipping implementation of Application Factories will be in our CodeGear Eclipse-based IDEs. This article provides a preview of how this new technology works and how it can be used by teams to leverage their in-house expertise.

What Are Application Factories?

Application Factories are specialized set of tools, scripts and templates for building and reusing applications. They contain a description of the application, configuration information, application development logic, and breadcrumbs to use and build out each of the components. They are devised by architects or senior developers for either repetitive programs or one-off applications, in which the designer can specify all the information a more junior developer will need to write the final product.

For that developer, the experience is not the typical one of looking at abstract UML diagrams or CRC cards and figuring out the code that is needs to be written. Rather, it is more a navigation experience through the application factory. This navigation consists of following the steps in the template, reading the logic and explanation of constraints, following the breadcrumbs to various resources, and then writing the code.

An Application Factory could contain scripts with indications such as: "Reuse the graphical interface module from the derivatives application at /code/derivapp/mainUI.java, and have it tie to the bond-trading DB instead of the options system." Or it could include other scripts, templates to be filled in by the developer. Used this way, Application Factories transform development into a series of actions that convert purpose-designed artifacts and templates into code sequences.

These artifacts can also be inserted into the code by developers as they work to indicate the reasons for implementation decisions, with embedded suggestions on how to modify the code in the event future changes are necessary. By thoughtfully including these artifacts at the decision time, developers provide support for future maintainers and give them appropriate signposts by which they can navigate the code successfully.

Application Factories can contain other numerous useful artifacts, such as architecture diagrams, code completion metadata, as well as various technologies for extending the application.

How Application Factories Work

Figure 1 shows how this process appears inside Eclipse. In the central pane, the cursor is located on a line that needs to be modified. In the right pane, the corresponding script is highlighted (in this case, a JavaScript file). In the bottom central pane is the script with the templated names of variables highlighted for the developer to attend to. In the top central pane, is the resource associated with the highlighted line, which in this case is part of an auto-generated stub.

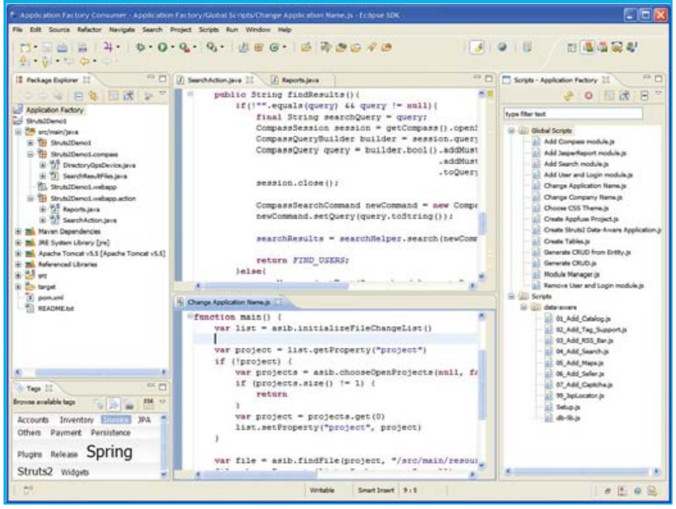
This example captures quite well a principal application of Application Factories: a body of code is auto-generated, let's say, from a UML tool. The architect goes through the generated code inserting scripts and tags for the developers; the developers then use it as a basic template for future instances of the specific application, or one-off variants. A site can easily

“The Application Factories’ approach to creating templates easily and to capturing the intent and knowledge of developers who implement an application provides a new model for designing software. It creates the possibility of rapidly deployable applications that consist of configuration data, templates, and some code.”

Ravi Kumar

Principal Architect, Java Tools Group, CodeGear

Figure 1. An Application Factory in Eclipse in which a template has been implemented as a JavaScript file (right pane and bottom middle pane).



```
public String findResults(){
    if(!"".equals(query) && query != null){
        final String searchQuery = query;
        CompassSession session = getCompass().open
        CompassQueryBuilder builder = session.query
        CompassQuery query = builder.bool().addMust
                                           .addMust
                                           .toQuery
        session.close();

        CompassSearchCommand newCommand = new Comp
        newCommand.setQuery(query.toString());

        searchResults = searchHelper.search(newComm

        return FIND_USERS;
    }else{
```

```
function main() {
    var list = asib.initializeFileChangeList()

    var project = list.getProperty("project")
    if (!project) {
        var projects = asib.chooseOpenProjects(null, f
        if (projects.size() != 1) {
            return
        }
        var project = projects.get(0)
        list.setProperty("project", project)
    }

    var file = asib.findFile(project, "/src/main/resou
```

maintain a collection of these templates for retrieval as needed, without fear that if the designer of the forms or the programmer who most works on the applications were to be absent that the work would have to wait. The artifacts and breadcrumbs make it possible for most any developer to step into the specialist’s shoes and move the task forward.

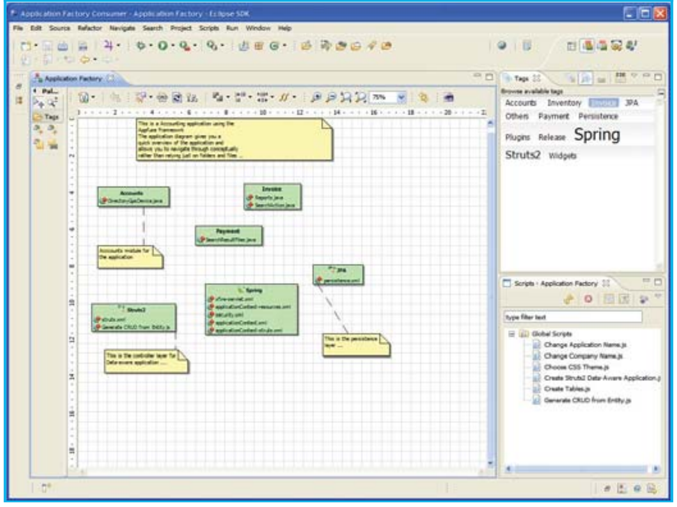
Application Factories make it possible to tag and track application evolution, run scripts to generate useful code snippets and capture changed artifacts as part of the script Archeology. In a typical scenario, a developer or and architect opens an application factory and then:

- Starts to navigate conceptually through the application using application factory meta-data
- Runs a script for generating template code or trails
- Switches to a resolver phase for the changes made by the script. This phase includes:
  - Reviewing each change and the reason why the change was made
  - Making any in-place edits to the changes performed by the script
  - Tagging the changes as needed
  - Scroll to the script line which produced the change and if need be customizing or modifying it
  - Committing the changes to a changeset
- Repeat as needed.

Figure 2 shows a simple Eclipse dialog that enables a developer to manage the tags for different parts of an application.

Application Factories also come with focus navigation features that enable developers to go back and forth easily between the Factory tags and the application, so that navigation is truly seamless. This feature avoids developers having to search through reams of code to find the items they need to change—which on a new project is a key productivity aid.

Figure 2. The tags for a given application can be grouped logically and intelligently managed.


Application Factory Consumer - Application Factory - Eclipse SDK
File Edit Source Refactor Navigate Search Project Scripts Run Window Help
Application Factory
Tags
Browse available tags
Accounts Inventory JPA
Others Payment Persistence
Plugins Release Spring
Struts2 Widgets
Accounts
Invoice
Payment
JPA
persistence.xml
Spring
Struts2
Accounts module for the application
This is the controller layer for Data-aware application ...
This is the persistence layer ...
Scripts - Application Factory
type filter text
Global Scripts
Change Application Name.js
Change Company Name.js
Choose CSS Theme.js
Create Struts2 Data-Aware Application.js
Create Tables.js
Generate CRUD from Entity.js

Using Application Factories with Frameworks

The benefits of this approach are clear for shops that rely on frameworks, such as Struts or Spring, or on enterprise software such as Tomcat. The senior designers can create Application Factory artifacts as they code, and so preserve their thinking for the use of other developers. As mentioned previously, this approach enables tutoring and mentoring as well.

At CodeGear, we are working currently on developing skeleton applications, called Modules, which dovetail with the major Java frameworks for Web applications and persistence. Existing pre-packaged applications derive from work with AppFuse (www.appfuse.org), which is a tool that specializes in creating framework-based application skeletons using Struts2, SpringMVC, JSF, Hibernate, JPA, Spring, and others. Our upcoming version of JBuilder will include Modules based on other frameworks as well as significantly expanded Application Factory functionality.

Application Factories provide architects with many tools that enable them to build their own Application Modules that are as rich as needs dictate. Basic design of the Module can be done through tools. These tools are particularly useful for applications that have clearly defined and bounded functionality. Scripts for the module can be written by hand or automatically generated. Application Factories enable architects to mine existing applications for features they want, so that reuse is maximized. In addition, various portions of the Modules can be made read-only or read/write depending on whether the architect wishes to allow developers to change and update the module design.

A New Model for Software Design

This approach to creating templates easily and to capturing the intent and knowledge of developers who implement an application provides a new model for designing software. It creates the possibility of rapidly deployable applications that consist of configuration data, templates, and some code. Starting from this base, a developer can customize and build the application and have it ready for roll-out in a matter of hours—thereby greatly reducing the cost and overhead associated with the complexity of today's Java applications.

For more information on Application Factories watch for upcoming announcements at **www.codegear.com** and read about CodeGear's other Eclipse-based JBuilder, JGear and 3rdRail IDEs. And join in discussions and share best practices with the more than 7 million developers doing Java, Eclipse, Rails, Windows, PHP and database development at CodeGear's Developer Network - **http://dn.codegear.com/**

Ravi Kumar

Ravi Kumar is principal architect in the Java tools group at CodeGear and is responsible for the vision and the architecture of Eclipse-based JBuilder product line. He specializes in SOA and Web Services tooling. He represents CodeGear on the Java Community Process and is on the Expert Group for JSR 208, Java Business Integration.