Java Architecture Overview

version 0.1
7/10/02

Version	Date	Person	Comments
0.1	7/10/02	Greg Eakman	Created

1. INTRODUCTION

Model Based Software Engineering (MBSE), as presented by Pathfinder Solutions, is a rigorous, complete, unambiguous, high level executable model of a particular problem domain. As such, this model can be translated into other executable implementations in the form of source code.

This document is an overview of the Java architecture and describes the build and runtime environments for the generated code.

> The overall architecture is based on the C++ architecture. For more information, see the C++ Architecture Users Guide.

2. Referenced Documents

Springboard User's Guide

C++ Architecture User's Guide

EJB Architecture User's Guide

"Model Based Software Engineering – an Overview of Rigorous and Effective Software Development Using UML™" 1998, Pathfinder Solutions, Inc.

3. System Requirements

The Java code generated from the models through the templates can be compiled using a Java Development Kit (JDK), of version 1.3 or greater, though it has also been built and tested using JDK 1.2. The code is targeted toward a J2SE (Java 2 Standard Edition) environment. The code should run easily with a few simple modifications, under the J2ME, the Micro Edition, although this has not been tested, and not officially supported yet.

The templates can also generate Enterprise JavaBeans (EJB, also known as J2EE for Enterprise Edition) code, by assigning certain properties to the model elements. See the EJB Architecture User's Guide for more information.

The code will run on any Java compliant virtual machine.

4. Building the System

This section describes how the code is generated and built.

4.1 Directory Hierarchy

Java organizes classes by packages, and, in the development environment, these package names are reflected in the directory structure. The generated Java files will be placed in a directory structure based on the Domain Chart, with the system name (Domain Chart name) as the top level class, and each of the domains as a subdirectory and subpackage of the system.

A Package property on the domain chart will be available to specify the upper parts of the package hierarchy. Most packages reflect the DNS domain name of the company producing the Java software – e.g., com.pathfindersol.java.mechanisms. The corresponding high level directories will be created in the document set.

4.2 Code Generation

Code generation is performed by the Springboard translation engine using the Java templates in the <PathfinderInstall>\design\java\templates directory. The top level template to use on the command line should be sys_top.

The following command line is commonly put into a .bat file.

```
springboard –x <xml file> -d <PathfinderInstall>\design\java\templates  sys_top <systemName>
```

4.3 Java Files

Generally, each UML class and Domain will result in one Java file that contains the implementation of the class. In some cases, more than one file and Java class is required to support the design.

Java filenames must reflect the top level class they contain.

The Java source code conforms to de-facto standard format and naming conventions, and can be tailored to any coding standards required by the client.

4.4 Mechanisms

A set of mechanisms, in the form of source code, is be provided by Pathfinder Solutions. These mechanisms provide the classes required to execute the MBSE semantics within the Java language.

4.5 SystemApp.java

This file is provided as the default starting point of the generated application. The SystemApp class contains the main() method and the calls to initialize the generated code and the main event loop. This file can be copied and modified to suit the individual application requirements, including realized code, initialization, etc., if necessary.

4.6 Compilation

The generated Java code has been compiled and tested with Sun's JDK command line compiler. Since this is a reference implementation, any Java IDE supporting a compatible revision of the Java language, should be able to compile the code.

A makefile is generated for the package that uses the Sun JDK command line compiler. This requires a make utility not provided with the Pathfinder Solutions package. A compatible make utility can be obtained from a number of sources, including GNU or the NMAKE utility provided with Microsoft Visual C++ products. If a standard IDE is chosen as the Java development environment, it may be possible to generate the project file for the IDE.

Note that the environment variable CLASSPATH must be set to include the Pathfinder mechanisms in <PathfinderInstall>\design\java\mechanisms and the SystemApp.java file in <PathfinderInstall>\design\java\system. The CLASSPATH should also include the top level directory of the package for the generated code, as well as any packages containing realized code.

No utilities are provided for packaging (jar'ing) the compiler .class files into a .jar file. Standard JDK tools exist for this purpose.

5. Realized Domains

Realized domains are domains that are not developed using UML models. They may be hand-written or come from existing or off-the-shelf components. Realized domains are wrapped in an interface to enforce the domain boundary and integrate the analyzed and realized domains.

Realized domain have a generated interface class, <DomainPrefix>_IF.java. The developer should create a class that implements this interface, instantiate it, and register it with the domain. This is accomplished with the generated method registerRealizedClass(). For example:

```
SW.registerRealizedClass(new SW_Impl());
```

6. Modeling Conventions and Restrictions

Springboard expects your OOA models follow the conventions described in "Pathfinder Solutions OOA/UML Modeling Guide".

Ideally, there are no conventions and restrictions placed on the analysis by the implementation or design technology. Practically, there is a small amount of feedback, especially for a design that is under development and intended to be used before it is complete. The following are conventions and constraints on the model for use with the Java design.

- Java supports return values from methods, but not output parameters. If targeting a Java architecture, then the models can reflect this constraint. The Java translation rules support the use of output parameters through “carrier” classes that carry the output parameter value from caller to callee.
- Use of output parameters from services in complex expressions. Service calls with output parameters cannot be used within a compound expression, like an IF or WHILE statement.
- Service handles are not yet implemented for the Java architecture.