Sapphire

Sapphire is a proposed open source project under the Technology project at Eclipse.

This proposal is in the Project Proposal Phase (as defined in the Eclipse Development Process) and is written to declare its intent and scope. We solicit additional participation and input from the Eclipse community. Please send all feedback to the Sapphire Forum.

Background

Little has changed in the way Java desktop UI is written since the original Java release. Technologies have changed (AWT, Swing, SWT, etc.), but fundamentals remain the same. The developer must choose which widgets to use, how to lay those widgets out, how to store the data being edited and how to synchronize the model with the UI. Even the best developers fall into traps of having UI components talk directly to other UI components rather than through the model. Inordinate amount of time is spent debugging layout and data-binding issues.

Sapphire aims to raise UI writing to a higher level of abstraction. The core premise is that the basic building block of UI should not be a widget (text box, label, button, etc.), but rather a property editor. Unlike a widget, a property editor analyzes metadata associated with a given property, renders the appropriate widgets to edit that property and wires up data binding. Data is synchronized, validation is passed from the model to the UI, content assistance is made available, etc.

This fundamentally changes the way developers interact with a UI framework. Instead of writing UI by telling the system how to do something, the developer tells the system what they intend to accomplish. When using Sapphire, the developer says "I want to edit LastName property of the person object". When using widget toolkits like SWT, the developer says "create label, create text box, lay them out like so, configure their settings, setup data binding and so on". By the time the developer is done, it is hard to see the original goal in the code that's produced. This results in UI that is inconsistent, brittle and difficult to maintain.

Scope

This project will be focused on identifying ways to significantly speed up UI development while simultaneously improving quality, maintainability and consistency.

Key deliverables:

- A declarative UI framework with a renderer for SWT and possibly other widget toolkits.
- A compact and easy to learn domain-specific modeling framework tailored to the needs of UI writers.
- Developer tools, documentation and samples.

Sapphire will be developed with extensibility in mind. Adopters will be able to extend the framework capabilities in a variety of ways, such as defining new UI parts, creating new property editors or even writing a complete new renderer for another widget toolkit.

While the modeling framework that's part of Sapphire could be used by itself for general modeling needs, it is not in scope of this project to promote such usage.

Initial Contribution

Oracle will contribute the initial codebase to the Sapphire project. This codebase has been used to deliver features in multiple releases of Oracle Enterprise Pack for Eclipse and is fairly mature, but it is still evolving rapidly to meet new requirements.

The initial contribution will include:

- The modeling framework, with support for value, element and list properties.
- A system for binding models to XML/DOM, including specific integration with the WTP XML editor. This system makes it easy to deliver form-based editors on top of XML files that have live bi-directional updates between the form view and the source view.
- The UI framework, with an XML-based UI definition parser and support for a variety of commonly-used parts:
    - Leaf Parts
        - property editor
        - label (for stand-alone use)
        - separator
        - spacer
        - action
    - Grouping Parts
        - composite
        - group
        - section
        - tab group
        - page book
    - Control Parts
        - with
        - if-else
- A UI renderer for SWT with a mature implementation of renderers for all UI parts, including the following property editor renderers:
    - string (default) - a text field with an optional browse button and other built-in facilities
    - boolean - a check box
    - enum - a radio buttons group or a combo box, as appropriate
    - integer - a combination of radio buttons and a text box to handle special values situation
    - integer - a scale with a text field for direct entry
    - list - a fully editable table
    - list - a slush bucket
    - path - a tree for selecting an Eclipse workspace path, similar to many Eclipse wizards

Initial Committers

- Konstantin Komissarchik (Lead), Oracle
- Ling Hao, Oracle

Interested Parties

- Nitin Dahyabhai, IBM
- Neil Hauge, Oracle

Relation to EMF

The focus of Sapphire is making UI development more productive. A core aspect of the architecture is tight coupling with a modeling framework. We did not chose to use EMF, as we feel that in many cases the needs of UI developers would be better served by a modeling framework designed to be easy to learn and tailored to specific scenarios that occur in UI development.

At the same time, we welcome participation of EMF experts interested in making it possible to use EMF models with Sapphire.

Relation to e4

Both Sapphire and e4 take a declarative approach to UI specification, so they may appear similar at first glance, but the two operate at different levels of abstraction. Sapphire deals with property editors and is tightly coupled to a model, while e4 talks about individual widgets and how to lay them out in detail. We believe that these two technologies complement each other and this project intends to support using Sapphire with e4.

We welcome participation of e4 experts interested in exploring the integration possibilities.

Relation to JFace Data Binding

The JFace data binding framework is a standard API to simplify synchronization of data between the UI and the model. It is possible to use the data binding framework as part of implementing a Sapphire property editor renderer, but the developer using Sapphire is not exposed to data binding concepts directly. This is possible due to a design decision that tightly couples the UI framework to a specific modeling framework.