Explorer Framework

Tree List Declarative User Interface

Marcel Baumann

Version 1.0.1

Table of Contents

1 Introduction

Complex client server application often needs simple clients to visualize and manipulate application and administrative information. The structure of the data is almost always of hierarchical nature. The description of the entities and their attributes is documented in a UML diagram and accessible through a XMI interface.

The explorer framework was developed to easily create simple explorer based user interface displaying hierarchical data. The goals of the framework are

- Create complete but simple explorer based user interface with minimum costs.
- The user interface do not need to support sophisticated graphical components or be especially fast or flexible.
- The tools based on the explorer framework are modified during the life cycle of the project. Modifications of the underlying data model should be propagated to the client with minimal source code edition.

The explorer tries to implement sound ergonomic rules.

1. What does the software look like to a non-technical user who has never seen it before?
2. Is there any screen in the user interface that is a dead end, without giving guidance further into the system?
3. The requirement that end-users read documentation is a sign of UI design failure. Is the explorer UI design a failure?
4. For technical tasks that do require documentation, do they fail to mention critical defaults?
5. Does the project welcome and respond to usability feedback from non-expert users?
6. And, most importantly of all... do the framework allow the users the precious luxury of ignorance?

2 Concepts

The central concept of the explorer framework is the declarative nature how its model is described and connected to the user interface. Current toolboxes concentrate their effort to develop fancy wizards or declaratively describe the user interface using XML. But the connection of the user interface to the business model is still a manual and cumbersome activity. This task must be executed each time the model changes.

The approach of the explorer framework is quite different and borrow its concepts from the MDA movement. The domain model is described as a platform independent model PIM. Cartridges generate most of the code of the application. Each time the model is updated, the code generator is triggered and a new user interface is created. The process of developing a user interface is different when using wizard. Once you ran the wizard you start to extend the code it created. From now you can never again run the wizard to reflect the changes of the model.

The explorer framework borrows its terminology of standard client server framework.

One of this framework is described in “Business Object Graphs: A pragmatic approach to MDA”. The explorer has a left pane, which displays a hierarchical tree view of the business objects being edited. The right pane displays the list of children of the currently selected node. Custom views can be added in the right pane, such a graphical representation or tab based forms.

The following terminology is used in this article.

- *Business Object*: Business objects are entities describing the business model and viewed in the user interface.
- *Lightweight Objects:* Lightweight objects are a lightweight representation of business objects displayed in the navigator view or as a result of search queries. Only lightweight objects are displayed in the navigator pane of the explorer.
- *Field*: Fields are business object properties displayed in the user interface. Fields never contains business objects and can have multiple values.
- *Relations:* Relations are properties containing business objects and displayed as a hierarchical structure in the navigator view. The objects referenced through relationships must have a lightweight representation. Relations only contains business objects and can have multiple values.

2.1 Declarative Model

The user interface structure is defined declaratively. Each property is identified through its name, has a display name which is often derived from the property name, the type of the property and if it can be edited or not.

- **Business Objects**: Business objects are defined as nodes displayed in the navigator. Nodes in the navigator can be expanded or collapsed. The objects can also be visualized in the object view and as member of a table view.
- **Context Sensitive Menu**: if the business object is editable, the menu contains entries to edit the object, remove it, add a new business object for each of its relations. The structure is as follow.
  - View action is available for all business objects.
  - Edit action is available for all business objects which can be edited.
  - The view sub-menu provides access to all views defined for the node type. The default view is automatically selected when the user select a node in the navigator.
  - Remove action is available if the relation to which the object belongs is editable.
  - Insert action is available if the relation to which the object belongs is editable.
  - Add actions are defined for all editable relations of the node type.
  - Custom actions can be defined and are inserted at the end of the pop-up menu.
- **Simple Properties**: A property is only visible in a table view is it is set to visible.
  - *Primitive Types*: Primitive types are displayed as property values in the object view and in the table. They are not displayed in the navigator.

  - *Business Object Types*: Business object types are never displayed as nodes in the navigator. Their properties can be manipulated in the generic dialog of the data object containing them.
- **Indexed Properties**: Business object values of a indexed property are only visible if it the property is set to visible.
  - *Primitive Types*: Primitive types are displayed as properties in the object view.
  - *Business Object Types*: Business object types are displayed as nodes in the navigator. Nodes in the navigator can be expanded or collapsed. Their properties can be manipulated in the object view and as member in the table view. Business objects can added, inserted, and removed from the indexed property.

2.1.1 Context Menu

The context menu provides a set of standard actions. These actions can be enabled or disabled through configuration.

- *Save:* The save action saves a data object graph. The action is only possible if the object is modified.
- *Revert:* The revert operation reverts the local changes on a data object graph. The operation is only available if the object is not new. The operation can only be enabled if the client supports two levels caching.
- *Reload:* The reload operation reloads the data object graph from the server. The operation is only available if the object is not new.

2.1.2 Naming Conventions

The declarative model uses naming conventions to access properties. The displayed beans must fulfill the naming rules.

- Simple Properties are mapped to user interface fields.
  - The getter method follows the bean conventions.
  - The setter method follows the bean conventions.
- Indexed Properties are mapped to user interface relations. The property name should be plural.
  - Add + singular property name adds a new item at the end of the collection.
  - Remove + singular property name removes the given item from the collection.
  - *Get + singular property name returns the item with the given index.*
  - *Set + singular property name sets the given item at the given index.*
  - Get + plural property name returns the collection of all items part of the relation.

Remove all items is realized using the get all items and remove item methods.

The lines in italics identify methods currently not used in the framework. The bean methods are called through reflection. The bean programmer is free to add any

business logic inside these methods.

2.1.3 Generic Dialogs

The declarative model is used to generate a generic dialog. The dialog displays all simple and indexed properties of the data object instance. Their values can be edited. The generic dialog retrieves declarative information from the user interface factory owning the declarative type of the edited explorer node. The design assumes that all explorer node types are handled through the same manager[1].

The following mappings are defined.

- **Simple Properties**: A property is only visible in a table view is it is set to visible.
  - *Primitive Types*: Primitive types are displayed in a text field. Special support is provided for types such as date, boolean, decimal numbers, and natural numbers.
  - *Business Object Types*: Business object types are displayed as a special area. Buttons are provided to edit a value and to delete a value. If no instance exist the edit will create a new data object. The edit action will display the generic dialog of the data object type.
- **Indexed Properties**: Business object values of a indexed property are only visible if it the property is set to visible.
  - *Primitive Types*: Primitive types are displayed as lists of values. Support is provided to add, modify and delete a value in the list. Special support is provided for editing types such as date, boolean, decimal numbers, and natural numbers[2].
  - *Business Object Types*: Business object types are displayed as nodes in the navigator. They are never displayed in a generic dialog.

2.1.4 Standard Right Views

The declarative model is used to display properties in the right pane. The list of attributes to be displayed and their order can be configured.

A node type can have right views.

2.1.5 Internationalization

The explorer framework support localization of all texts used in dialogs or views. The Internationalization concept uses a simple but powerful concept.

- Each text is defined as a static string in a class
- The text translations are stored in a resource bundle following the Java conventions
- Helpers methods are provided to retrieve a resource bundle and update the static fields using reflection

1 This assumption is anyway already build in the explorer framework.
2 This feature works only with generics. The generic dialog needs to know which type of objects it must instantiate.

This approach allows developers to use the refactoring tools provided in modern integrated development environments.

2.2 Runtime Engine

2.2.1 Data Access Rules

The explorer framework provides a set of powerful yet simple rules how application data is retrieved and stored.

- The lightweight representation in the navigator is the sole responsibility of the application developer. The tree can be retrieved from the database, through a search dialog or any other method.
- The default table view in the right pane uses only properties of the lightweight business objects. Therefore the navigation through the left pane tree is very efficient.
- When the user selects a view representation from an explorer pop-up menu, the explorer framework checks if the business object graph of this node is available. If not it walks up the lightweight tree until it finds a node being root. The graph of business object graph under this root is loaded from the underlying layer. During this process the navigator representation is synchronized with the loaded graph. Finally the requested view is displayed.
- Business objects are always loaded if necessary before a view, an edit, an insert or a save operation. A remove operation does not request a load. A load operation is always performed on a root object.
- Save operations are tricky. The explorer framework must handles the case where the object to save is a new one and the collection containing it must therefore also be saved. The above rule guarantees that the owning object is already loaded. The explorer must walk up the navigator until it reaches a modified object not being new or the higher object is not persistent. This object is the best solution for the save operation. Because this behavior is not transparent to the user, the save operation generates an error message on new objects.
- Explorer nodes containing root business objects have a save function. When selected the complete graph of business objects is stored in the underlying layer. The optimistic timestamps are automatically updated to synchronize the client view with the underlying layer.

Naturally the application user is free to add his own mechanisms such as an explicit or implicit save action in the view.

2.2.2 UI Node Type

The user interface node type provides semantic actions through the context pop up menu defined for each node type. Context menus are available on the lightweight tree representation in the navigator pane.

- When a view must be displayed the explorer checks if the business object is already available in the application. If not it loads the associated graph of business objects and connects all the objects with their explorer nodes. The graph is selected

from the first lightweight node being a root.

2.2.3 Lightweight Object Handler

The handler provides services to synchronize lightweight representation with their business objects. Lightweight representations are often displayed in the navigator pane of the explorer. The default provided handler has the following assumptions.

- Properties have the same name in the business object and its lightweight representation.
- Only business objects have lightweight representations.
- A graph of lightweight objects is always complete. No partial graphs of lightweight objects exist. This assumption is reasonable due to the existence of lightweight objects.

The handler provides the following services.

- Synchronize a lightweight object with its business object. Each lightweight object has a reference to its business object. This reference can be null.
- Create the lightweight representation of a business object graph.

Lightweight trees can also be retrieved through the persistence layer if the business objects are persistent.

3 Architecture

The explorer package provides all mechanisms to map hierarchical data structures to various graphical representations. Hierarchical data structures are any kind of directed graph structures. The main goal is that no modification to the underlying model classes are necessary. So a clean interface between the business model and the visualization model is realized. Additionally legacy systems can be integrated without expensive modification activities. Another goal is that the package can be used to create remote user interfaces using the same remote model stored on a server application. The communication mechanisms between the framework and the graphical clients are local method invocation, CORBA or RMI.

- The node class is wrapper class, which models the concept of a node in a hierarchical structure. The associated object in the model is referenced in the node. The node contains the name and the associated icon. It implements lazy evaluation for its children. The children nodes are only instantiated as needed. This mechanism supports tree of thousands of nodes as long as one node as no more than about thousand children.
- The node type defines the visible attributes of the underlying node class in the graphical representation. If an attribute is modifiable is also described. The node type provides the method used to retrieve the children of the business object and the method to inquiry the name displayed in the tree. The Java reflection features are used to provide the meta information.
- Care was taken to allow sophisticated graphical representations. The framework provides hooks to select the icon associated with a node and its current status.

- Each node type object can define a context menu for all nodes of its type. The context menu is accessed through the usual mechanism of the platform on which the application is executed.
- The node type factory provides a factory pattern to create new nodes and their associated node type for all underlying nodes part of the model. The factory creates a new node for a given underlying object and creates objects for all its children.

3.1 Context Menus

A context menu selector is configured for each user interface node type.

4 Cookbook

This chapter shows the reader how to create an explorer based interface displaying a hierarchical data. The major steps are the following.

1. Create a subclass of the explorer.
2. Define all node types with their properties, indexed properties and visible attributes of the children. If desired a custom icon selector can be registered per node type.
3. For each node type create the view class used to display and modify the attributes of business objects. The type of the business objects are the ones defined in the node type.
4. Run the new explorer application.
5. Optionally define additional context sensitive actions in the pop-up menu or global actions in the menu bar of the explorer. Support classes are provided to simplify creation of new actions. Standard actions such as new, open, save and save as are already supported. Only the application specific operations need to be implemented.

5 History

The actual version of the explorer framework is derived from a prototype developed to visualize networks of Internet enabled devices. The framework was refined to better support declarative programming styles and aligned to use the more modern terminology of the MDA movement.