Compendium Developer Documentation

Michelle Bachler
Knowledge Media Institute, Open University, UK
devsupport@compendiuminstitute.org

This document is intended to supplement the Compendium JavaDocs. It is in no way a comprehensive documentation of the whole system, but merely tries to clarify some of the more complex areas of the application.

Overview of the 'Main' class and JFrame

The main class to start Compendium is com.compendium.ProjectCompendium. This class instantiates the application frame com.compendium.ui.ProjectCompendiumFrame. It also starts the log file for system output messages and the UIStartUp dialog which displays initial startup messages.

The application frame, ProjectCompendiumFrame, holds various application variables and several key manager classes, which it uses to draw and manage the user interface, and to communicate with MySQL databases. The diagram below shows the main classes involved.

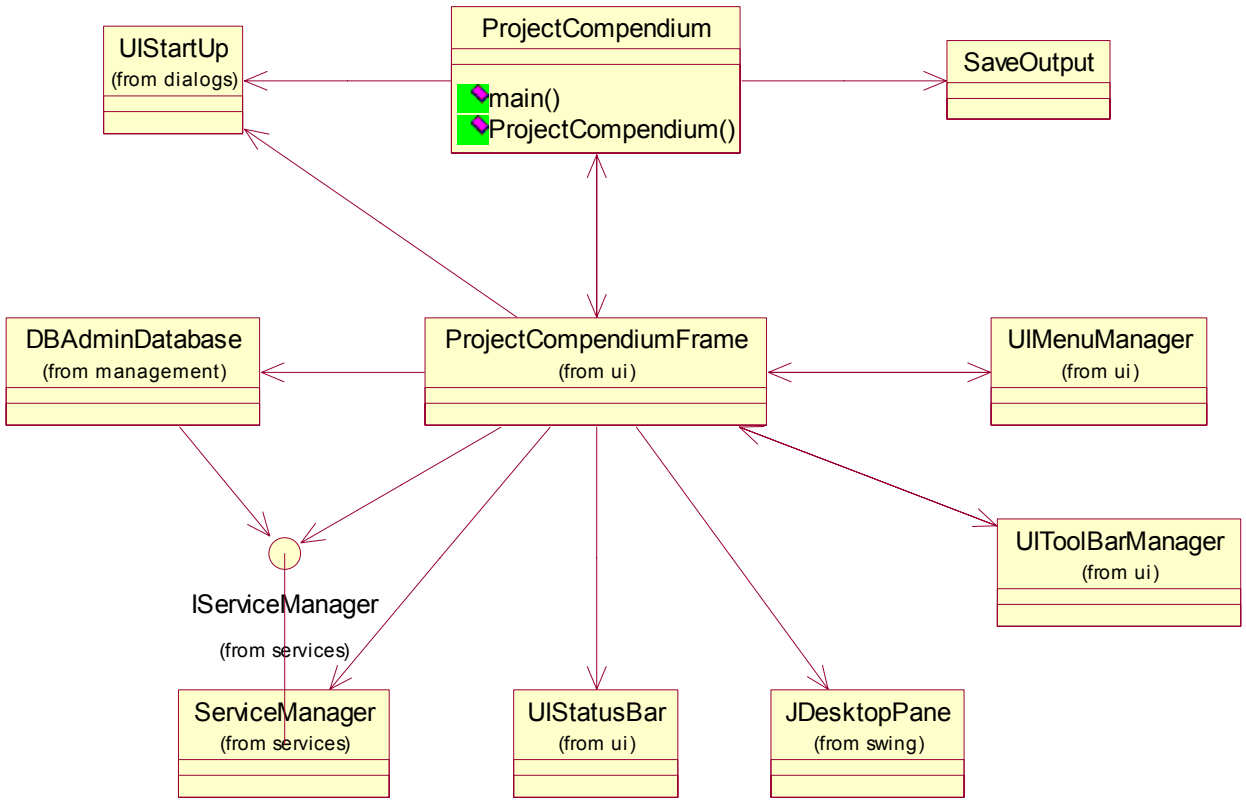


Diagram 1. Overview of the main frame.

The Core: The Datamodel Layer

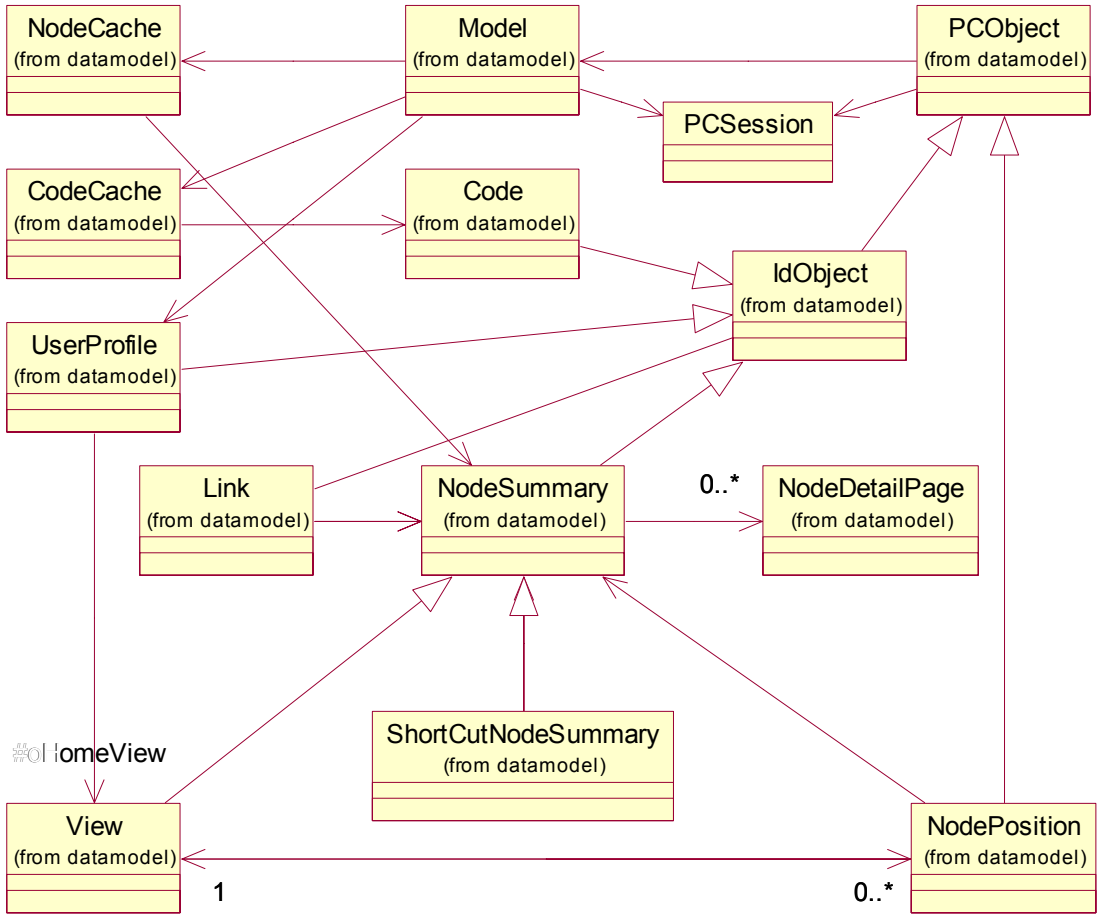
The datamodel layer contains classes to store the raw data recovered from the database, which in turned is used by the user interface classes to access the data to display to the user. The base class for the major datamodel classes is PCObject. All classes which have a unique identifier as part of their stored data, such as NodeSummary, Code and Link are subclasses of IdObject (itself a subclass of PCObject).

A NodeSumary instance holds all the basic information for a node record, including a collection of instances of NodeDetailPage, for its pages of detail text. There are two subclasses of NodeSummary, ShortcutNodeSummary, and View. ShortcutNodeSummary is a small refinement on a standard NodeSummary, in that it holds extra information to say which parent node it is a shortcut to.

The View class, on the other hand is a much more complex refinement. A view is itself a node (appears as a node icon in another map or view), so has all the basic data associated with a node record, but it also holds the data required to realize a view at the user-interface level. For a list type view, the View instance will hold a collection of NodePosition instances. For a map view, it will hold collections of NodePosition and Link instances.

A NodePosition instance associates a NodeSummary and a View instance and holds the position information for that node in that view.

The diagram below attempts to illustrate these relationships. You will also see other classes such as the Model, PCSession and UserProfile, which will be explained further in the next section.



The Core: The Model and Services

At one stage in its past, there was an attempt to make Compendium a client-server system. Now the client and server code has been merged into one free-standing application. However, due to remaining legacy code structures, there is still a class called 'Model', which controls access to the services layer used to access the database, and holds a reference to the current UserProfile and the current PCSession objects. This stems from when the Model class managed the client side, of the client-server implementation. Though much of the old client-server code has been removed, the Model and services layer have been retained at present.

The UserProfile holds all the information about the currently logged in user. The PCSession object holds the association between the user and the open database, and it contains a unique session id to identify this association, (again, stemming from when this was the client side of the code) .

Currently all datamodel classes (NodeSummary, View, Link etc) have two states, uninitialized and initialized. When datamodel objects are first created, they are returned in an uninitialized state. In this state, they hold information which can be accessed through 'getter' methods, but any setter methods which need to write to the database cannot do so until the object is initialized. Each object needs to have its 'initialize' method called at some point after creation. This method takes a reference to the current Model instance and the PCSession instance held in the Model, and both are required by the datamodel classes before they are fully functional and can write and read to the database.

The datamodel classess require these two references because it is the Model which holds all the instances of the service layer classes needed to access the database. These service layer class methods also need to be passed the instance of PCSession. For this reason, methods in the datamodel layer that read or write to the database using a service instance can throw a ModelSessionException, if they have not been initialized.

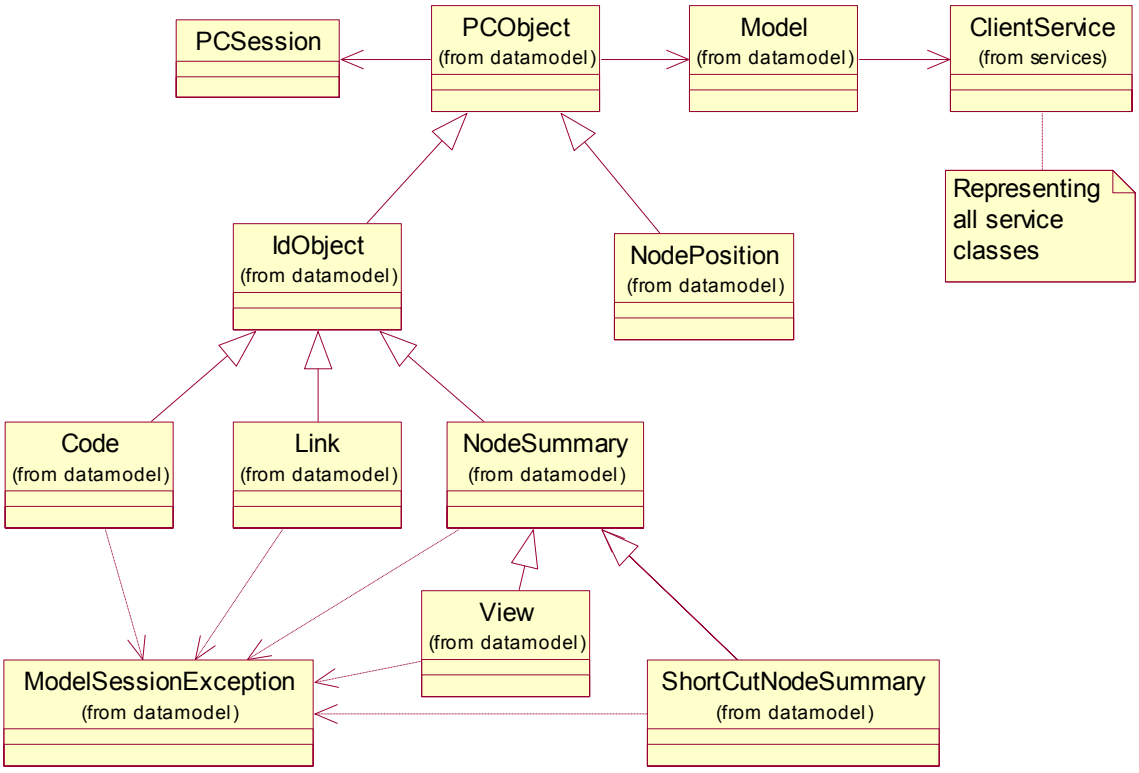


Diagram 3. The Datamodel classes and the Model class.

The Service Manager

As you will see later in the section on 'Database Projects', there is a ServiceManager class which is responsible for validating a user logging in, obtaining their UserProfile instance, then creating a PCSession instance for that user in that database and finally creating and returning the associated Model instance. It is the SerivceManager class which creates and manages instances of the service classes and assigns them to the Model object when it creates it. This code was originally part of the server side of the client-server version of Compendium.

The service layer classes are all subclasses of ClientService, which is itself a subclass of the Service class. When the ServiceManager creates the service instances (which it later sets in a Model instance), it passes each service a reference to the com.compendium.core.db.management.DBDatabaseManager class. The services can then use the DBDatabaseManager class to obtain DBConnection objects to pass to the db layer classes which actually access the database.

Diagram 4, below, illustrate some of the relationships of the ServiceManager class.

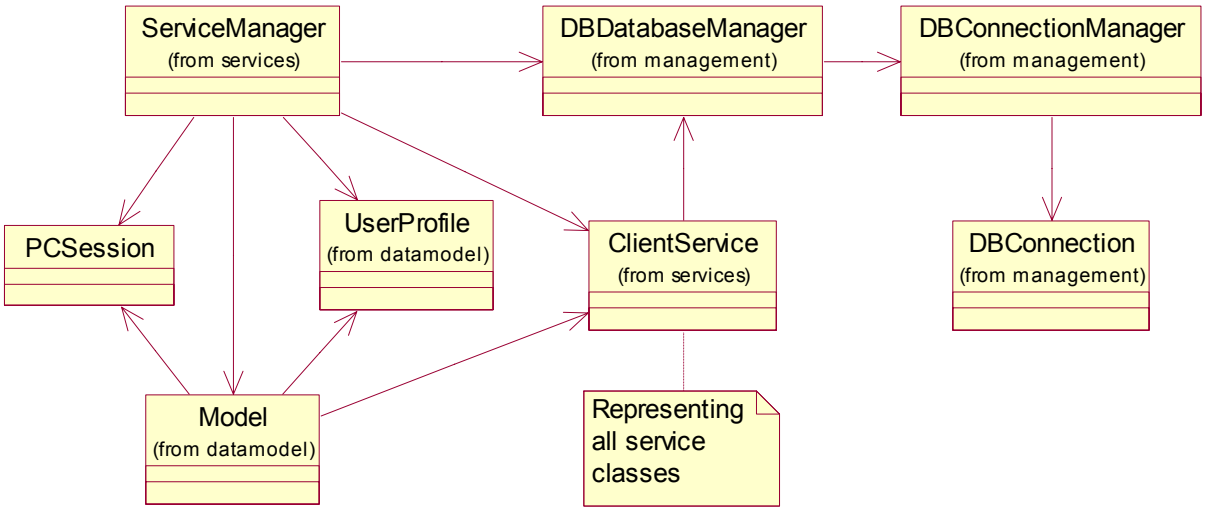


Diagram 4. The ServiceManager class.

Roughly speaking there is one com.compendium.core.datamodel.services class and one corresponding com.compenduim.core.db class for each table in a Compendium database. The db layer classes hold the SQL statements and the methods associated to read and write data to and from a specific database.

Diagram 5 below, illustrates the services and db layer class relationships.

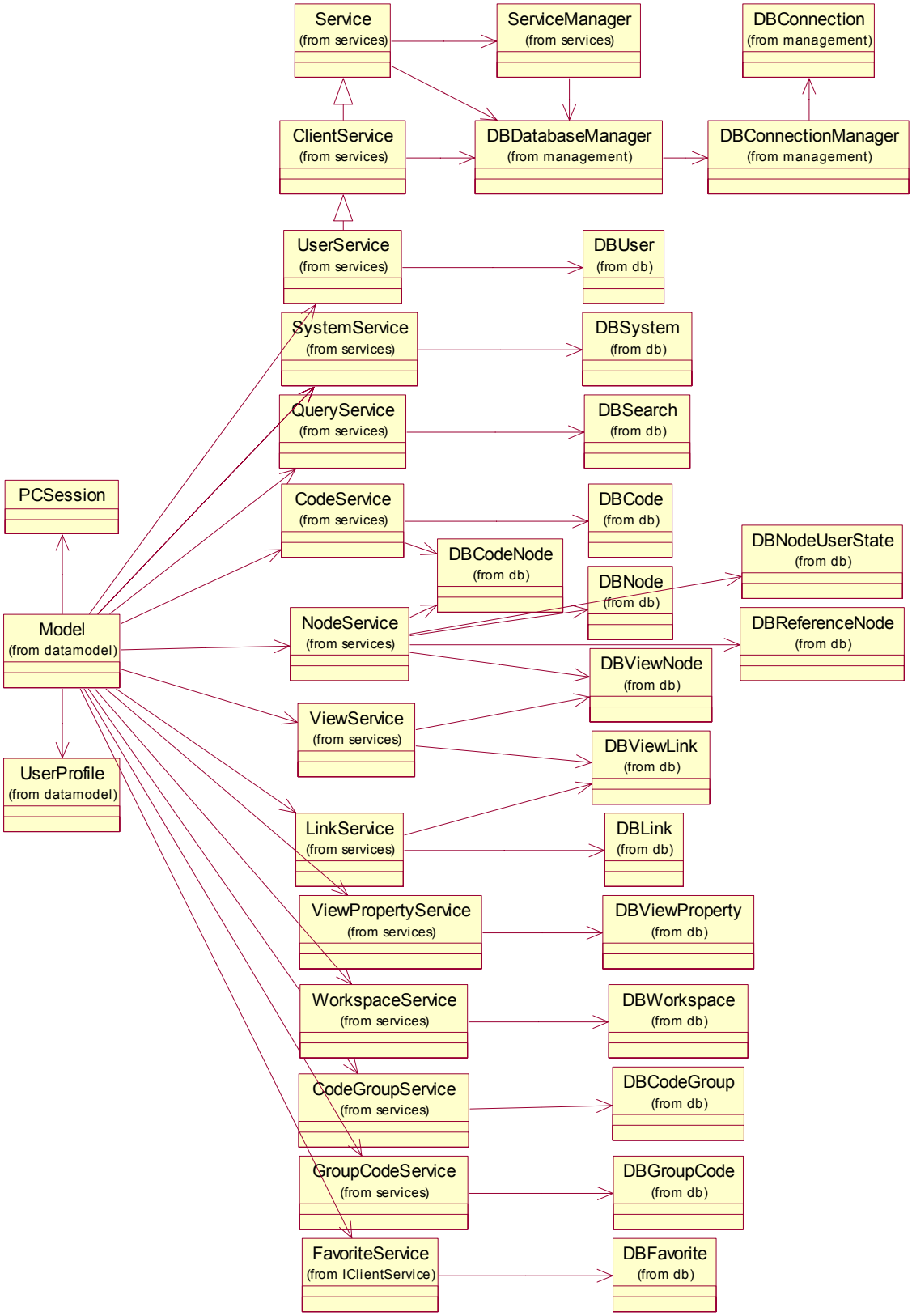


Diagram 5. The Model and Services

Initializing database connectivity

When the Compendium application is first started, the ProjectCompendiumFrame instance creates a new ServiceManager instance and then, using this, a com.compendium.core.datamodel.management.DBAdminDatabase instance. The DBAadminDatabase class accesses the 'compendium' administration database, which list all databases created. From this class, a list of the existing database projects is loaded. These two classes are used by the application for database maintenance tasks and database project access.

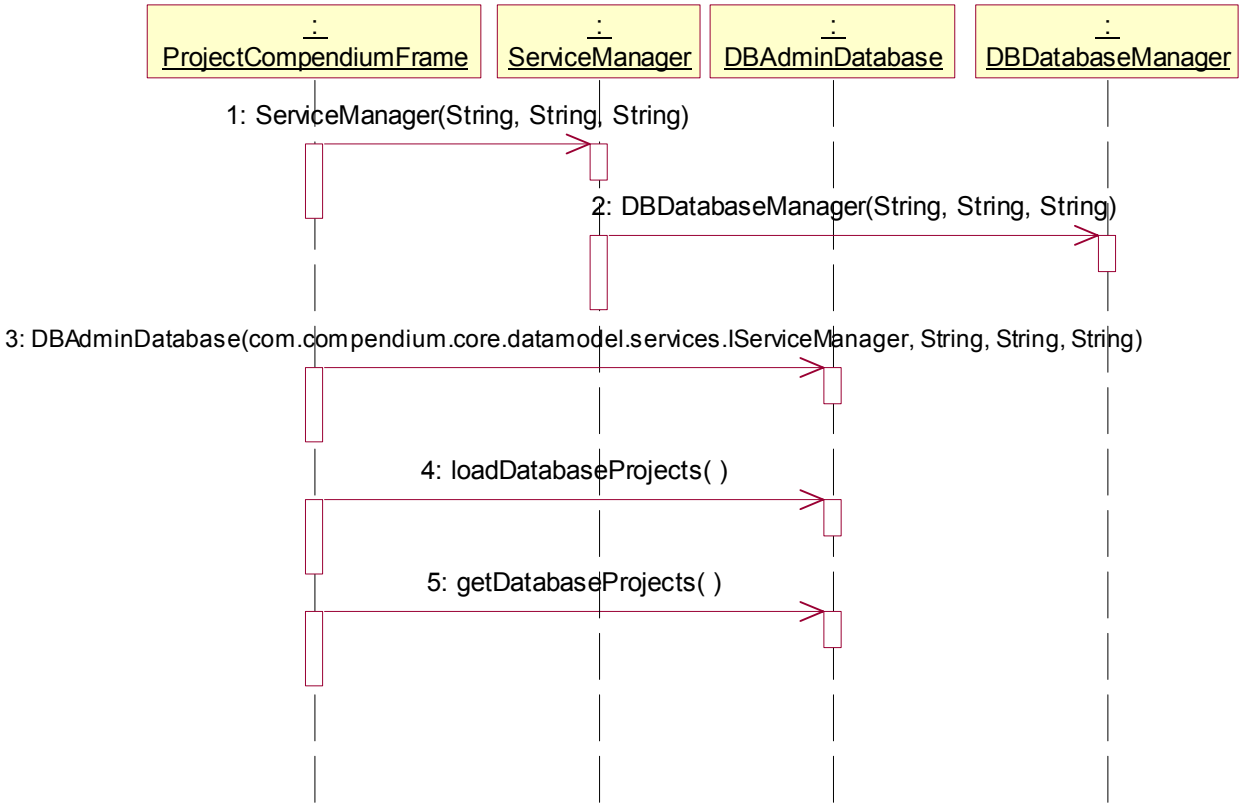


Diagram 6. Initialization of database connectivity.

Creating a new Compendium Database Project

When the user requests to create a new database, ProjectCompendiumFrame will create a new instance of UINewDatabaseDialog. This dialog has fields for the user to enter the details for the new user account to create for the new database, and to enter the name of the new database. It is also this dialog class that communicates with the com.comendium.db.management layer to create the new database itself. It does this by creating a new instance of the DBNewDatabase class and then calling the createNewDatabase method, passing it the name for the new database, as entered by the user.

This name is then converted to a system database name by removing any illegal characters and by adding a date stamp (in milliseconds) to the end of it. DBNewDatabase uses DBEmtyDatabase to create the new blank database in MySQL, and it then loads the default data require, (com.compendium.core.db.management.DefaultData.sql), which includes the Administration user's record, System table records and the default node codes (tags).

The diagram below details this sequence of events:

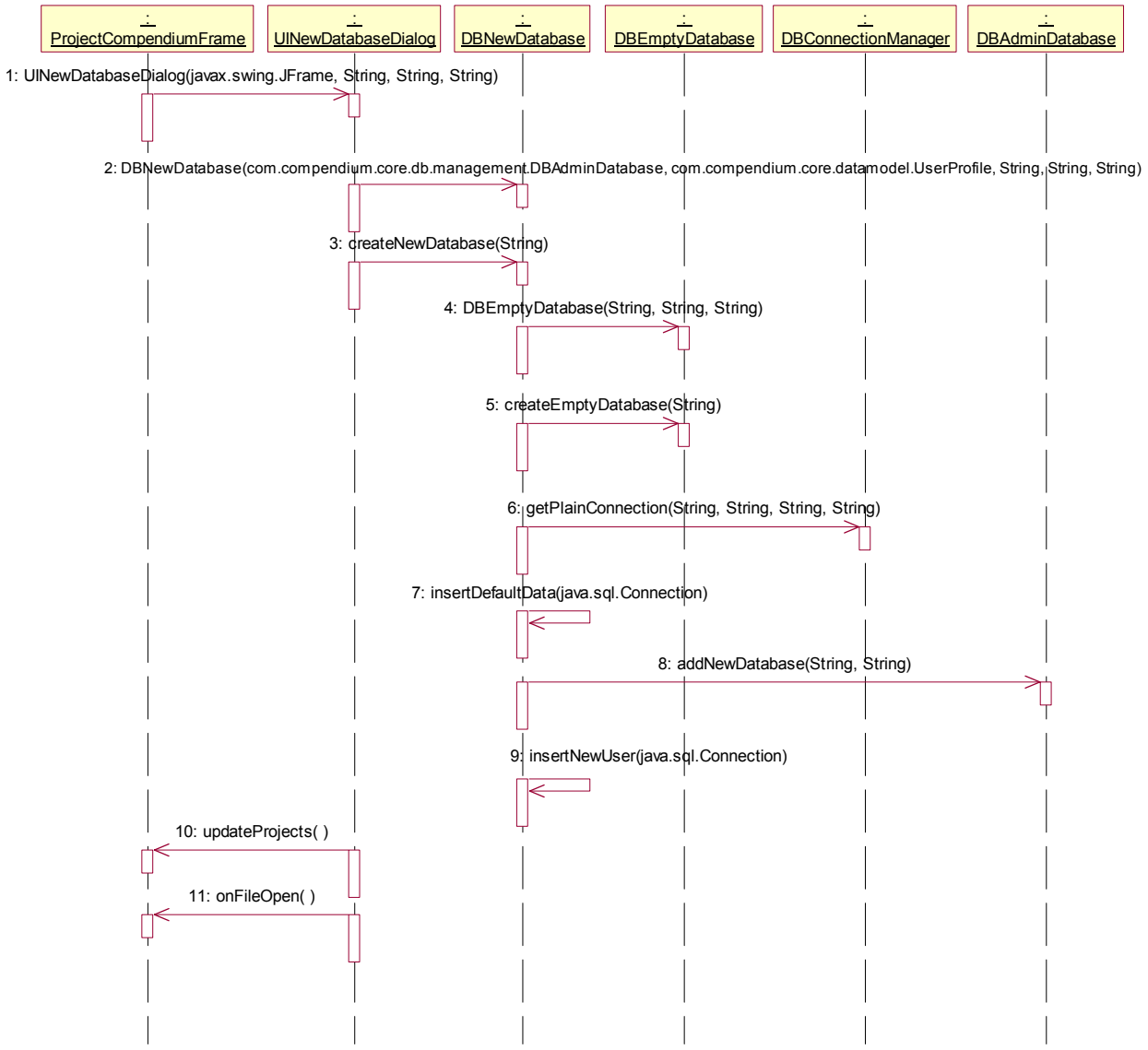


Diagram 7. Create a new database project.

Opening a Database Project

Due to the legacy structure of the 'core' Compendium code (which used to be part client code, part server code), this process is somewhat convoluted.

If the user selects to open a database project a UILoginDialog instance is created by ProjectCompendiumFrame and the user can select a database and enter their username and password. ProjectCompendiumFrame then requests the database name, username and password from the dialog and uses these to request the ServicesManager instance to validate and register the new user for the given database.

The diagram below details the sequence of events involved. For a more detailed breakdown of how the ServiceManager validates the user details, see diagram below 'User Validation Sequence Diagram'. After validation the ServiceMananger returns a new instance of the Model object which acts as the datamodel layer manager, holding references to all the service instances for accessing the various database tables for the database opened, and caches for node and codes loaded.

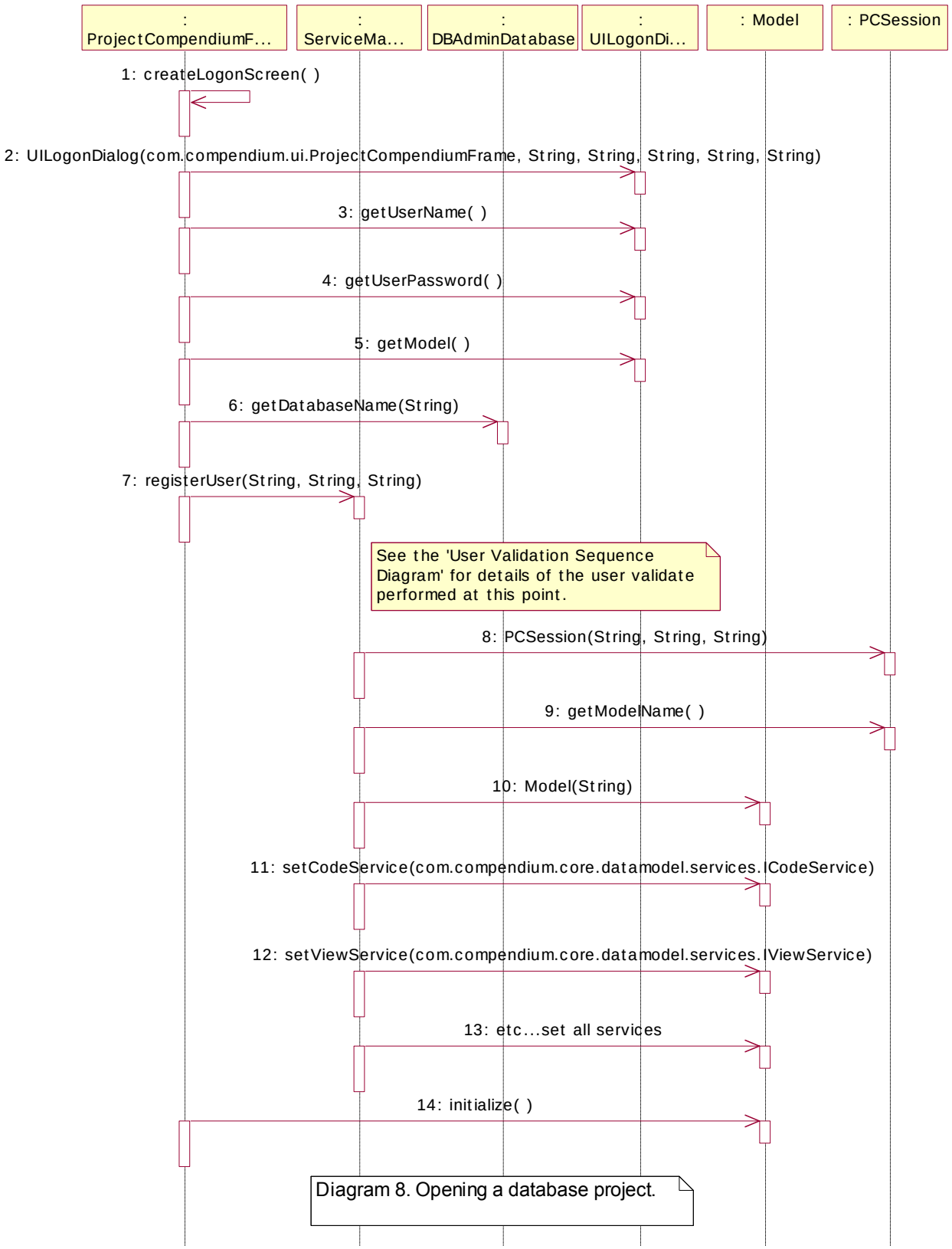


Diagram 8. Opening a database project.

User Validation Sequence Diagram

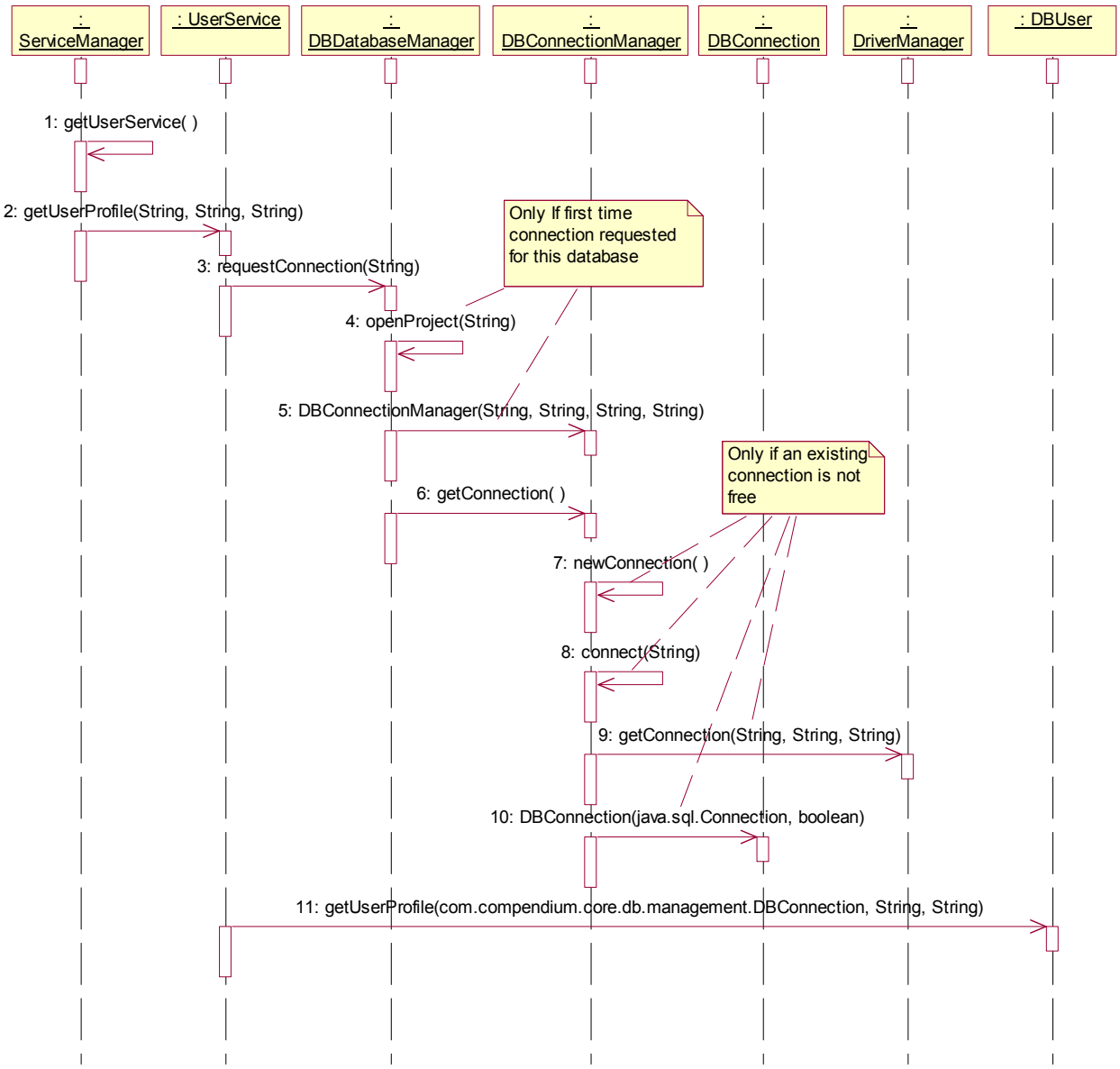


Diagram 9. User validation.

Views

A view, in Compendium, is either a map or a list type view and appears inside a subclass of JInternal Frame (UIViewFrame) and is placed in the JDesktop instance held by ProjectCompendiumFrame (see diagram 1). The first view loaded, is loaded by the application when a database project is opened. This view is the home page of the user who has logged in, and is always a map view. Other views, either maps or list, are then initially created in the home map and then, of course, within these views, new views can be created etc..

The diagram below attempts to illustrate how the user-interface and datamodel objects used to construct a view relate to each other, (note: only major relationships are shown here).

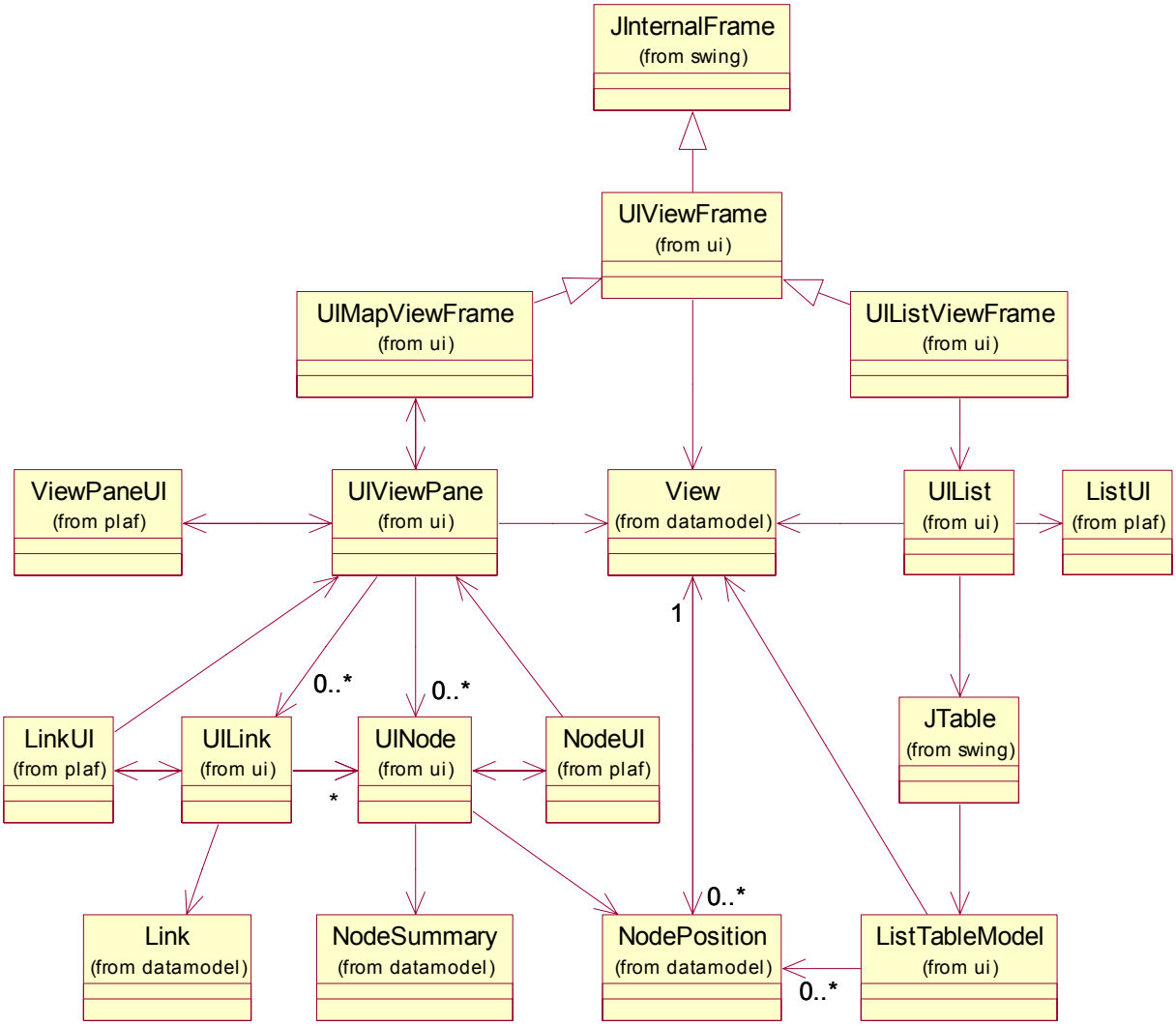


Diagram 10. Views.