Архитектура модуля программного интерфейса ЕВФРАТ-Документооборот

Б. Л. Романов, Д. Я. Слободецкий

В работе рассматривается задача разработки программного модуля, описывающего классы объектов предметной области информационной системы, и реализующего программный интерфейс (API) для взаимодействия с ними. Основное назначение такого модуля — отделение алгоритмов, реализующих основные операции с объектами от пользовательского интерфейса и, тем самым, упрощение процедуры разработки пользовательского интерфейса, а также повышение надежности системы. Описываются проблемы, возникающие при разработке пользовательского интерфейса системы, а также пути их решения с помощью соответствующей организации интерфейсов модуля.

Введение

Рассмотрим задачу разработки пользовательских приложений информационной системы. Система состоит из нескольких приложений, каждое из которых решает свою задачу. Модель данных информационной системы состоит из набора взаимосвязанных типов объектов предметной области, каждое приложение обеспечивается некоторую функциональность по работе с этими объектами. Свойства любого объекта системы и отношения с другими объектами не зависят от того, в каком приложении они используются.

Представление объектов в информационной системе может быть довольно сложным. Объект может описываться файлом на диске, набором записей в базе данных, некоторой структурой в памяти и т. п. Методы управления объектами могут быть оформлены в виде библиотек функций и классов, веб-сервисов, хранимых процедур базы данных и т. д. Объекты

разных типов могут сильно отличаться друг от друга, как способами представления данных, так и способами управления.

Таким образом, когда информационная система состоит более чем из одного приложения, целесообразно выделить в архитектуре системы программный модуль, реализующий работу с программными объектами, соответствующими объектам предметной области, скрывая тонкости их реализации. Этот модуль обеспечивает контроль целостности данных, как каждого объекта, так и их совокупности, управляет взаимоотношениями между объектами, выполняет кэширование информации, необходимой для быстрого доступа к данным объектов, представляет информацию об объектах таким образом, чтобы ее было удобно использовать при разработке конечных приложений, в частности, при разработке пользовательского интерфейса. Фактически, такой модуль реализует программный интерфейс — API — для взаимодействия с объектами информационной системы.

При проектировании интерфейса модуля API следует уделить пристальное внимание задачам, которые возникают при разработке пользовательского интерфейса системы. Реализация программного интерфейса модуля API с учетом этих задач позволит сильно упростить и сократить время разработки пользовательских приложений.

1. Круг проблем

При создании пользовательских приложений для СЭД ЕВФРАТ-Документооборот была разработана библиотека классов DFAPI, реализующая программный интерфейс для взаимодействия с информационными сущностями системы документооборота (документы, бизнес-процессы, поручения, согласования и т. п.). Несмотря на различное назначение и кажущуюся непохожесть разных типов объектов, многие задачи, которые возникали при разработке пользовательского интерфейса для работы с объектами системы, были общими и не зависели от их типа:

- поиск и отображение списков объектов;
- одновременное использование данных объекта в разных частях программы;
- своевременное обновление данных объекта при их изменении.

Использование единых, не зависящих от типа объекта методов решения этих проблем позволило существенно сократить время разработки пользовательских приложений, а также повысить их надежность за счет повторного использования уже отлаженного кода.

2. Поиск и представление списков объектов

Объекты разных типов хранятся в разных частях базы данных системы, для управления ими используются различные службы. Для представления свойств объектов используются разные способы. Запросы на выборку объектов разных типов могут иметь разные представления. Представление набора объектов тоже может зависеть от типа. Для того чтобы упростить и уменьшить объем кода пользовательского приложения, решающего эти задачи были предприняты следующие меры:

- Выбран единый язык для представления запроса на выборку объектов. Все запросы на выборку объектов, независимо от их типа, представляются в виде строки на этом языке. Это дало возможность, во-первых, использовать общий пользовательский интерфейс для формирования сложных пользовательских запросов на выборку объектов, а во-вторых, легко использовать типовые запросы в разных частях программы, что было бы затруднительно в случае, когда запрос представляется в виде иерархии классов сложной структуры.
- Был разработан обобщенный, не зависящий от типа, интерфейс для обращения к списку объектов, устроенный таким образом, чтобы его можно было использовать в качестве источника данных стандартных элементов управления.

На различных этапах развития информационных технологий предпочтение давалось разным способам представления и хранения данных. Самым четким переломом способа работы с объектами стало появление объектно-ориентированного подхода разработки приложений. Было разработано большое количество программ, которые есть смысл использовать в дальнейших разработках. Различный способ хранения данных повлек за собой различные способы представления запроса на поиск. В базе данных НИКА, которая является популярным средством хранения данных на территории России, запрос на поиск представляется в виде XML-файла. Для большинства реляционных баз данных, разработан единый язык запросов SQL. Каждое из приложений имеет свой вид представления объектов, что влечет за собой различные способы работы с ними.

Для решения проблемы различного представления запросов, необходимо выбрать некоторый единый формат представления поискового запроса. Формат запроса должен в себе нести информацию о типе искомых объектов, для дальнейшего преобразования в нужный вид. В качестве единого стандарта представления запросов может выступать XP-строка. XP-строка имеет вид:

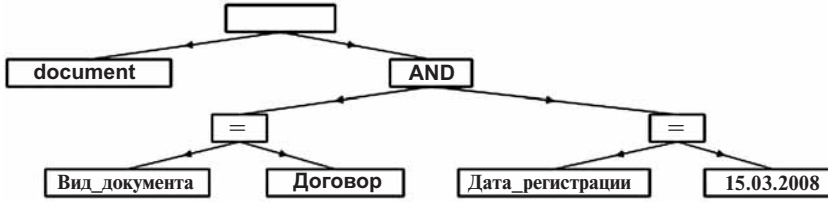

Рис. 1. Пример дерева разбора

```
<тип_объекта>
  [<имя_атрибута><операция_сравнения><значение_атрибута>
                  <операция_объединения_атрибутов>...
  ],
```

где `<тип_объекта>` несет в себе информацию о типе объекта, по которому производится поиск.

Процесс преобразования запроса из вида XP-строки в необходимое представление состоит из двух этапов:

1. С помощью общего парсера происходит разбор XP-строки в дерево функциональных единиц. Например, если имеется такая строка запроса:

```
document[(Вид_документа=Договор)AND
         (Дата_регистрации=15.03.2008)]
```

В результате разбора мы получим дерево вида рис. 1.

2. В зависимости от типа искомых объектов строится генератор необходимого вида запроса, который работает с деревом разбора XP-строки. Суть генератора состоит в правильной обработке каждого элемента узла. На выходе генератора получается необходимый объект представления запроса.

Поскольку часто поиск производится для получения информации о конкретных атрибутах объекта, то нет необходимости в выдаче в качестве результата множество найденных объектов, тем более что каждый объект описывается различными классами. В качестве элемента результата поиска используется ссылка на объект, которая содержит в себе идентификатор найденного объекта, а также значения запрошенных атрибутов.

3. Кэширование данных и использование ссылок

Один и тот же объект может быть получен разными способами — например, как результат поиска или по ссылке из другого объекта. Для поддержания целостности данных, система должна понимать, что имеет дело с одним объектом. В частности, все изменения, вносимые в одно

представление объекта, должны быть отражены и в другом его представлении. Для решения этой задачи были предприняты следующие меры:

- Была введена глобальная идентификация всех объектов, независимо от их типа. При этом использовалось то, что все объекты одного типа имели уникальный идентификатор внутри типа. Общий идентификатор объекта включал в себя идентификатор типа и идентификатор объекта внутри типа.
- Было введено понятие «ссылка на объект»—сущность, в которой хранится глобальный идентификатор объекта, а также некоторые свойства исходного объекта, доступные только для чтения. При этом в качестве списков объектов, полученных в результате выборки, использовались списки ссылок, которые содержали только те свойства объектов, которые были необходимы для отображения списка.
- При необходимости получить данные самого объект из ссылки, использовалась процедура загрузки объекта, соответствующая его типу. При этом полученный объект помещался в кэш. В дальнейшем, при обращении к этому объекту по его идентификатору — возвращается уже загруженный объект. Объект удаляется из кэша автоматически после уничтожения всех ссылок на него (рис. 2).

Использование ссылок на объекты в сочетании с кэшированием позволило использовать один и тот же объект из разных частей программы. Причем в программе нигде, кроме кэша, не хранятся данные объекта, используются только ссылки. Это уменьшает объем использованной памяти, за счет отсутствия дублирования, а также позволяет контролировать целостность данных объектов API.

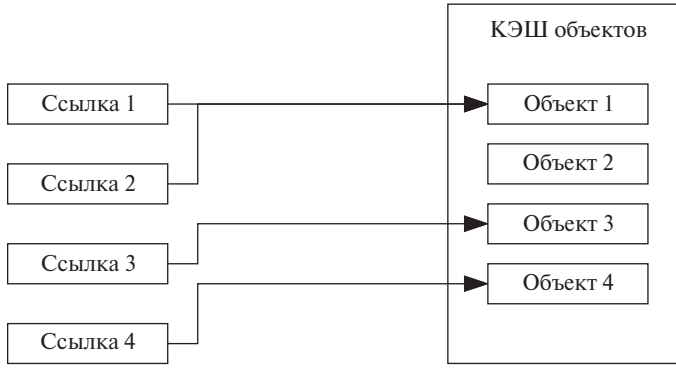

Рис. 2. Взаимодействие ссылок и объектов

С небольшими доработками эта схема позволяет решить еще одну важную задачу. Бывает ситуация, когда необходимо принудительно обновить данные объекта, сохраненного в кэше. Например, если объект изменился на сервере. Для этого достаточно удалить закэшированные данные — тогда при следующем обращении обновленные данные объекта загрузятся автоматически. Проблема в том, что не всегда известно, изменились ли данные на сервере, а непрерывное слежение за их состоянием может отрицательно сказаться на производительности системы. Однако, есть ситуации, в которых с большой вероятностью, данные объекта отличаются от закэшированных и объект следует обновить. При этом может оказаться, что обновлять надо не один объект, а группу взаимосвязанных объектов. Например, в системе ЕВФРАТ-Документооборот есть понятие процесса Workflow, состоящего из деятельностей по этому процессу. При изменении состояния одной деятельности может измениться как состояние всего процесса, так и других его деятельностей, если они связаны в последовательность. Также при принудительном завершении процесса изменяются состояния всех его деятельностей — они также завершаются. Для упрощения автоматического обновления свойств взаимосвязанных объектов в кэше хранится информация о связях между объектами и при принудительном обновлении объекта происходит автоматическое обновление всех связанных с ним.

Данная схема кэширования объектов была реализована в модуле `API` системы ЕВФРАТ-Документооборот на платформе .NET. Данные объектов системы разных типов были представлены в виде разных классов, соответствующих типам объектов, причем все они унаследованы от класса `DFObjectInstance`, описывающего базовые свойства объекта. Для представления ссылок на объекты используется класс `ObjectRef`. Этот класс имеет метод `GetObject`, который возвращает указатель на соответствующий ссылке `DFObjectInstance`. Для получения указателя на данные объекта класс ссылки использует метод `OpenObject` класса `DFConnection`, представляющего соединение с сервером системы. В этом методе осуществляется поиск объекта в кэше по идентификатору и, в случае если требуемый объект не найден, выполняется процедура получения данных объекта с сервера и помещение его в кэш. В случае частого обращения к закэшированному объекту из ссылки процедура поиска в кэше может отрицательно сказаться на производительности системы. Для исключения этой проблемы класс ссылки после успешного получения указателя на `DFObjectInstance` запоминает его во внутренней переменной

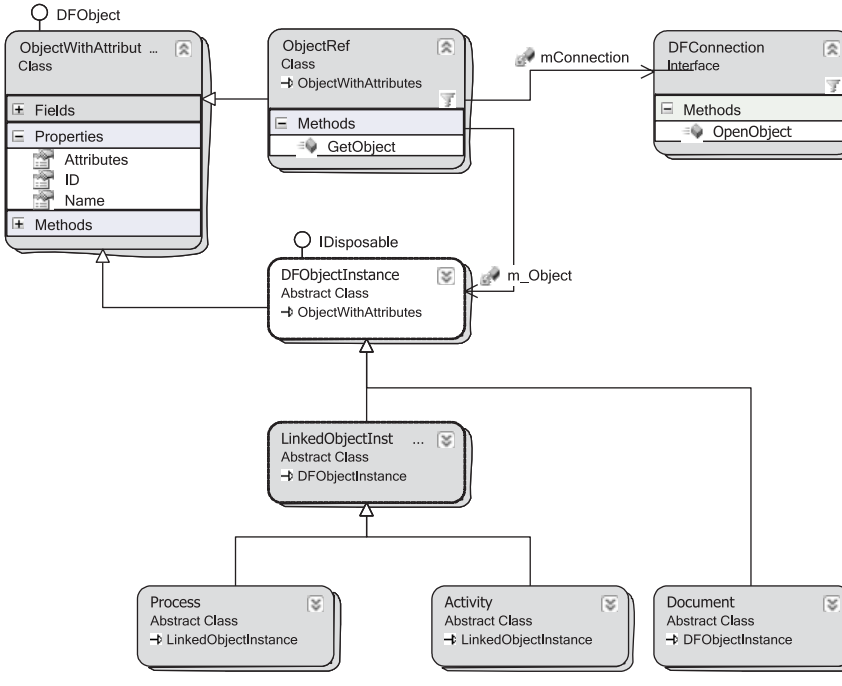

Рис. 3. Диаграмма классов **API** ЕВФРАТ-Документооборот

`m_Object`, и при последующих обращениях к объекту через эту ссылку поиск не производится. Такое запоминание указателя может привести к рассогласованию данных после выполнения процедуры обновления данных объектов (см. выше). Для исключения этой проблемы все экземпляры класса ссылки после успешного получения соответствующего экземпляра класса `DFObjectInstance` регистрируются в нем. Это дает возможность при удалении экземпляра класса *DFObjectInstance* из кэша уведомить об этом все заинтересованные экземпляры класса `ObjectRef` и обнулить значение их переменной `m_Object`.

В результате применения описанной модели взаимодействия с объектами разных типов при разработке модуля `API` и приложений системы ЕВФРАТ-Документооборот удалось быстро реализовать основные элементы пользовательского интерфейса. Простота и надежность модели позволила реализовать в модуле `API` полный контроль целостности данных объектов, что, в свою очередь, позволило избежать многих ошибок и повысило надежность системы в целом.

Литература

1. *Басс Л.*, *Клементс П.*, *Кацман Р.* Архитектура программного обеспечения на практике. 2-е изд. СПб.: Питер, 2006.
2. *Богданов А. С.*, *Емельянов Н. Е.*, *Ерохин В. И.*, *Романов Б. Л.* Реализация запросной системы на основе языка XPath для СУБД НИКА // Организационное управление и искусственный интеллект: Труды ИСА РАН. М.: URSS, 2003. С. 132–148.