УДК 519.2

ББК 22.17

Обработка данных и управление в сложных системах: Сборник статей / Под ред. Глуховой Е.В. Томск: Изд-во Том. ун-та, 2002. Вып. 4. 112 с.

ISBN 5-7511-1594-5

Сборник содержит статьи сотрудников и аспирантов факультета информатики, экономики и математики филиала Кемеровского государственного университета в г. Анжеро-Судженске и факультета прикладной математики и кибернетики Томского государственного университета, посвященные статистической обработке временных рядов, актуарной математике, а также вопросам управления в системах массового обслуживания и в измерительных системах.

Для студентов, аспирантов, научных работников, занимающихся вопросами временных рядов и управления в измерительных системах.

УДК 519.2

ББК 22.17

ISBN 5-7511-1594-5

ОБЪЕКТНЫЙ ПОДХОД К ПРОБЛЕМЕ ПРОЕКТИРОВАНИЯ ПОДСИСТЕМЫ НОРМАТИВНО-СПРАВОЧНОЙ ИНФОРМАЦИИ

К.Ю. ВОЙТИКОВ, О.А. ЗМЕЕВ, А.Н. МОИСЕЕВ

Введение

При разработке любой системы документооборота не обойтись без объектов, обеспечивающих функциональность подсистемы нормативно-справочной информации. В литературе встречается достаточно много определений сущностей, обеспечивающих функциональность этой подсистемы, например, справочник, перечисление и т. д. В теории баз данных соответствующие структуры получили название классификаторов, но в терминологии UML это название употребляется в ином контексте, поэтому в настоящей работе будет использоваться другой термин – «Справочник» (Referens). Под справочником в дальнейшем будет пониматься объект информационной системы, содержащий набор (список) значений условно-постоянных реквизитов для каких-либо других объектов приложения. Отдельный элемент такого списка в дальнейшем будет называться элементом справочника.

Существует и другое определение этого понятия. Справочник – это агрегированный тип данных, средство для работы со списками однородных элементов данных. В большинстве своем он является электронным аналогом каталога. Каждый элемент – это строка справочника, а сведения, заносимые в элемент, являются реквизитами справочника.

Классификация справочников

При реализации подсистемы нормативно-справочной информации средствами объектно-ориентированной парадигмы, справочники в зависимости от их практического применения можно разделить на четыре вида: простой, подчиненный, иерархический, иерархический подчиненный. Рассмотрим каждый из этих типов более подробно.

К простым справочникам можно отнести справочник, у которого обязательными являются только два реквизита: уникальный идентификатор элемента справочника и наименование элемента. Обычно справочник этого типа представлен пользователю в виде набора элементов оформленных на уровне интерфейса пользователя в виде обычной таблицы.

К подчиненным справочникам относятся справочники, которые в обычном проектировании образуют с элементом другого справочника отношение «главный – подчиненный», например, набор различных типов цен для некоторого товара. Для реализации представления подобных справочников достаточно трех обязательных атрибутов: уникальный идентификатор элемента справочника, идентификатор родителя, предоставляющий доступ к значению родительского элемента для выбранного элемента подчиненного справочника, и наименование элемента.

Иерархические справочники допускают упорядочивание своих элементов свободным, определяемым пользователем системы образом. Типичный пример такого справочника – это прайс-лист товаров, допускающий различную структуру номенклатурных групп для различных компаний. Для обеспечения функциональности такого справочника необходимо наличие четырех обязательных атрибутов: уникальный идентификатор элемента справочника, идентификатор родителя, предоставляющий доступ к значению роди-

тельской группы выбранного элемента справочника, наименование элемента и свойство, отвечающее за то, является ли элемент группой. Принципиальное отличие иерархических справочников от подчиненных в том, что для подчиненного уровень вложенности не более двух, а для иерархического ничем неограничен, более того, структура справочника в целом определяется пользователем системы во время ее эксплуатации.

Последний из перечисленных выше видов справочника, иерархический подчиненный, редко имеет практическое применение. Для его реализации от языка программирования требуется поддержка механизма множественного наследования, поэтому в рамках настоящей работы не рассматривается.

При обзоре возможных справочников мы выделили только служебные реквизиты, то есть те, которые необходимы для реализации функциональности этих классов как справочников. Естественно, что эти классы, кроме служебных, должны иметь и другие атрибуты, которые отвечают за смысловое наполнение элемента справочника в рамках реализуемой в системе предметной области.

При реализации механизма использования справочников средствами объектного подхода, необходимо спроектировать соответствующую структуру классов, диаграмма наследования которой приведена на рис. 1.

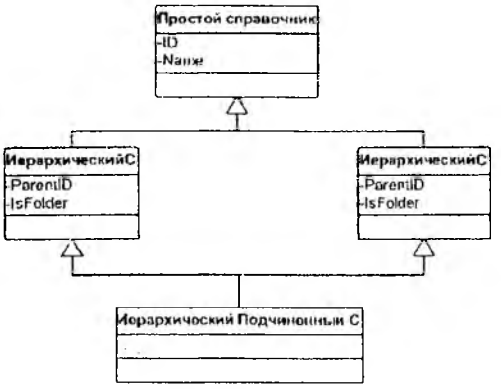

Рис. 1. Структурная диаграмма наследования

Рассмотрим обязанности справочников. Во-первых, он должен уметь отображать свое содержимое, то есть отображать свои элементы клиенту системы. Для этого у класса справочник должны быть определены соответствующие операции. Во-вторых, он должен обеспечить клиенту работу с уровнем хранения данных, как самого справочника, так и его элементов, проще говоря, обеспечить вставку, изменение и удаление данных. Также у него должны быть операции, отвечающие за поиск элемента в справочнике. Таких методов как минимум два: поиск по наименованию и поиск по уникальному идентификатору. Следующий метод – это выбор элемента справочника, который позволяет использовать те или иные значения реквизитов элемента справочника. Также в иерархических и подчиненных справочниках нужно определить операции, при помощи которых можно определить элемент владелец для выбранного элемента справочника, а также уровня элемента.

Применение шаблонов проектирования

Реализация шаблона Composite для справочников

Рассмотрим элемент справочника. Можно провести аналогию работы как со всем справочником, так и с отдельным его элементом. Если справочник работает с массивом своих элементов, то элемент работает с массивом своих реквизитов. Следовательно, у элемента будут операции, связанные со вставкой, изменением, удалением реквизитов. У него будет реализована операция поиска реквизита элемента, операции выбора реквизита элемента и т. д.

При работе со справочником его целесообразно рассматривать как цельный объект системы, не делая различий между понятиями «набор элементов справочника» и «непосредственный элемент». Из всего выше сказанного можно сделать вывод, что при проектировании справочников можно применить шаблон проектирования под названием Composite [1].

Рассмотрим аспекты применения данного шаблона в контексте проектирования системы справочников. Composite (компоновщик) – шаблон, структурирующий объекты. Этот шаблон компонует объекты в древовидные структуры для представления иерархий часть/целое и позволяет клиентам единообразно трактовать индивидуальные и составные объекты, что вполне удовлетворяет требованиям, которые мы предъявили к системе нормативно-справочной информации.

При проектировании справочников, используя данный шаблон, необходимо выделить классы примитивов, которые выступают в роли контейнеров примитивов. В данном случае классом контейнеров примитивов будет строка справочника – класс элемент. Исходя из концепции данного шаблона более мелкие объекты можно сгруппировать для формирования более крупных, применяя рекурсивную композицию таким образом, что клиенту не придется проводить различие между простыми и составными объектами. То есть клиент будет работать со справочником, не зная, чьи методы в данный момент используются, методы самого справочника или методы элемента справочника.

Ключом к шаблону Composite является абстрактный класс, который представляет одновременно и примитивы, и контейнеры. В нашем случае он носит название «*Справочник*». В нем объявлены операции, специфичные для каждого вида справочников, и общие для всех составных объектов.

Для реализации этого шаблона необходимо определим его участников и их обязанности:

- Referens – справочник:
  - объявляет интерфейс для объектов;
  - представляет необходимую операцию, общую для всех объектов;
  - объявляет интерфейс для доступа к потомкам и управления ими;
  - объявляет интерфейс для доступа к родителю в рекурсивной структуре и при необходимости реализует его;
- Item – элемент:
  - определяет поведение примитивных объектов в композиции;
  - представляет листовые узлы композиции;
- Composite – составной объект:
  - определяет поведение справочника в целом;
  - хранит компоненты-потомки, т.е. в нашем случае элементы справочника;

– реализует относящиеся к управлению потомками операции в интерфейсе класса Referens;

- Client – клиент:

– манипулирует объектами композиции через интерфейс Referens.

В результате определения участников получилась несколько видоизмененная композиция. Структура, описанная выше в несколько упрощенной форме, отражается на структурной диаграмме классов, приведенной на рис. 2.

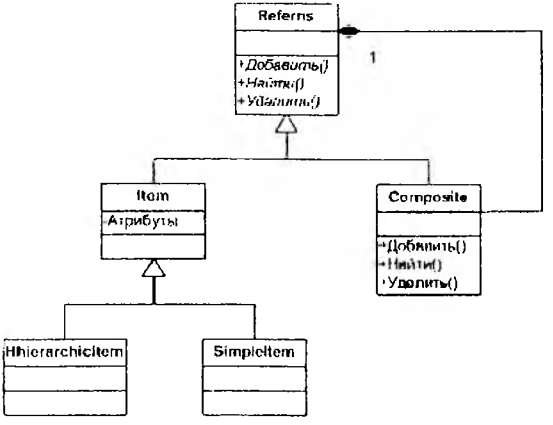

Рис. 2. Диаграмма наследования шаблона Composite

Дадим оценку результатам, полученным в результате применения шаблона проектирования:

- определены иерархии классов, состоящие из примитивных и составных объектов. Из примитивных объектов можно составлять более сложные, которые, в свою очередь, участвуют в более сложных композициях и так далее. Любой клиент, ожидающий примитивного объекта, может работать и с составным;
- упрощена архитектура клиента. Клиенты могут единообразно работать с индивидуальными и объектами и с составными структурами. Обычно клиенту неизвестно, взаимодействует ли он с элементарным или составным объектом. Это упрощает код клиента, поскольку нет необходимости писать функции, ветвящиеся в зависимости от того, с объектом какого класса они работают;
- облегчено добавление новых видов справочников. Новые подклассы классов Composite или Item будут автоматически работать с уже существующими структурами и клиентским кодом. Изменять клиента при добавлении новых компонентов не нужно;
- способствует созданию общего дизайна.

Практическая реализация шаблона

При практической реализации шаблона компоновщик для системы справочников нужно рассматривать несколько вопросов.

1. Явные ссылки на родителей. Хранение в справочнике ссылки на своего родителя упрощает обход структуры и управление ею. Наличие такой ссылки облегчает передвижение вверх по структуре и удаление справочника.

Мы предлагаем определить ссылку на родителя в классе Referens. Классы Item и Composite в этом случае наследуют саму ссылку и операции с ней. При этом поддерживается следующий инвариант: если некоторый объект в составной структуре ссылается на другой составной объект как на своего родителя, то для последнего первый является потомком. Простейший способ гарантировать соблюдение этого условия – изменять родителя компонента только тогда, когда он добавляется или удаляется из составного объекта. Если это удалось реализовать в операциях родителя, то реализация будет унаследована всеми подклассами и, значит, инвариант будет поддерживаться автоматически.

2. Максимизация интерфейса класса Referens. Одна из целей паттерна компоновщик – избавить клиентов от необходимости знать, работают ли они с листовым или составным объектом. Для достижения этой цели класс Referens должен сделать как можно больше операций общими для классов Composite и Item. Обычно класс Referens предоставляет для этих операций реализации по умолчанию, а подклассы Composite и Item замещают их.

Но это может вступить в конфликт с принципом проектирования иерархии классов, согласно которому класс должен определять только логичные для всех его подклассов операции. Класс Referens поддерживает много операций, не имеющих смысла для класса Item. Для этого нужно перенести в класс Referens операции, которые, на первый взгляд, имеют смысл только для составных объектов. Например, интерфейс для доступа к потомкам является фундаментальной частью класса Composite, но вовсе не обязательно класса Item. Однако, если рассматривать потомков класса Item как Referens, у которых *никогда* не бывает потомков, то мы можем определить в классе Referens операцию доступа к потомкам как никогда не возвращающую потомков. Тогда подклассы Item могут использовать эту реализацию по умолчанию, а в подклассах Composite она будет переопределена, чтобы возвращать потомков.

3. Объявление операций для управления потомками. Хотя в классе Referens реализованы операции Добавить и Удалить для добавления и удаления потомков, но для шаблона компоновщик важно, в каких классах эти операции объявлены. Надо ли объявлять их в классе Referens и тем самым делать доступными в Item, или их следует объявить и определить только в классе Composite и его подклассах?

Решая этот вопрос, мы должны выбирать между безопасностью и прозрачностью:

– если определить интерфейс для управления потомками в корне иерархии классов, то мы добиваемся прозрачности, так как все компоненты удается трактовать единообразно. Однако расплачиваться приходится безопасностью, поскольку клиент может попытаться выполнить бессмысленное действие, например, добавить или удалить объект из листового узла;

– если управление потомками сделать частью класса Composite, то безопасность удастся обеспечить, ведь любая попытка добавить или удалить объекты из листьев в статически типизированном языке будет перехвачена на этапе компиляции. Но прозрачность мы утрачиваем, ибо у листовых и составных объектов оказываются разные интерфейсы.

В шаблоне компоновщик мы придаем особое значение прозрачности, а не безопасности. Если для вас важнее безопасность, будьте готовы к тому, что иногда вы можете потерять информацию о типе и придется преобразовывать компонент к типу составного объекта.

4. Кэширование для повышения производительности. Если приходится часто обходить композицию или производить в ней поиск, то класс Composite может кэшировать информа-

цию об обходе и поиске. Кэшировать разрешается либо полученные результаты, либо только информацию, достаточную для ускорения обхода или поиска.

Любое изменение компонента должно делать кэши всех его родителей недействительными. Наиболее эффективен такой подход в случае, когда компонентам известно об их родителях. Поэтому, если вы решите воспользоваться кэшированием, необходимо определить интерфейс, позволяющий уведомить составные объекты о недействительности их кэшей.

5. Удаление компонентов. В языках, где нет сборщика мусора, лучше всего поручить классу Composite удалять своих потомков в момент уничтожения. Исключением из этого правила является случаи, когда листовые объекты постоянны и, следовательно, могут разделяться.

6. Структура данных, более всего подходящая для хранения.

Составные объекты могут хранить своих потомков в самых разных структурах данных, включая связанные списки, деревья, массивы и хэш-таблицы. Выбор структуры данных определяется, как всегда, эффективностью. Собственно говоря, вовсе не обязательно пользоваться какой-либо из универсальных структур. Иногда в составных объектах каждый потомок представляется отдельной переменной. Правда, для этого каждый подкласс Composite должен реализовывать свой собственный интерфейс управления памятью.

Реализация шаблона Proxy для справочников

Для лучшей реализации справочника стоит использовать этот шаблон совместно с другим шаблоном – Proxy [1]. Proxy (заместитель) – шаблон, структурирующий объекты. Является суррогатом другого объекта и контролирует доступ к нему. Разумно управлять доступом к объекту, поскольку тогда можно отложить расходы на создание и инициализацию до момента, когда справочник действительно понадобиться. Так, для объектов справочника затраты на создание нескольких таких объектов могут оказаться весьма значительными. Но система должна работать быстро, поэтому нужно избегать создания всех тяжелых объектов на стадии открытия, да и при работе с системой многие вряд ли понадобятся. Поэтому разумно создавать такие объекты по требованию. Для этого нужно ответить на ряд вопросов: что поместить вместо объекта справочник? Как скрыть то, что объект создается по требованию? И как сделать так, что бы оптимизация не отражалась в коде, относящемуся к реализации методов справочника?

Решение заключается в том, чтобы использовать другой объект – объект-заместитель справочника. Он временно представляется вместо реального справочника. Заместитель ведет себя точно так же, как и сам объект. Шаблон заместитель применяется еще и в тех случаях, когда необходимо сослаться на объект более изощренно, чем это возможно, если использовать простой указатель. Для справочника с помощью этого шаблона можно реализовать следующее.

- Удаленный заместитель. Предоставляет локального представителя вместо объекта, находящегося в другом адресном пространстве. При использовании трехуровневой архитектуры клиент-сервер, при необходимости создается локальный представитель экземпляра объекта справочник.
- Виртуальный заместитель. Создавать объекты справочник по требованию.
- Защищающий заместитель, предназначенный для контроля доступа к исходному объекту справочник. Это требуется, когда для различных объектов справочник определены различные права доступа.

• «Умная ссылка» – это замена обычного указателя. Она позволяет выполнить дополнительные действия при доступе к объекту справочник. Например:
– подсчет ссылок на реальный объект справочник для того, чтобы занимаемую им память можно было автоматически освободить, когда не останется ни одной ссылки;
– загрузка экземпляра объекта справочник при первом обращении к нему;
– проверку и установку блокировок на реальный объект при обращении к нему, чтобы никакой другой объект не смог в это время изменить его.

На рис. 3 показана диаграмма классов для структуры с заместителями.

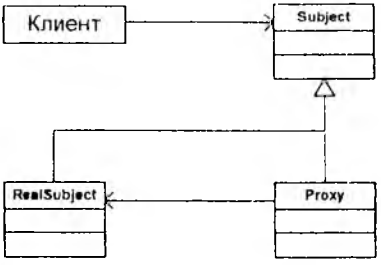

Рис. 3. Диаграмма классов для структуры с заместителями

Proxy – заместитель:

– хранит ссылку, которая позволяет заместителю обратиться к реальному субъекту. Объект класса Proxy может обращаться к объекту класса Subject, если интерфейсы классов Real Subject и Subject одинаковы;

– предоставляет интерфейс, идентичный интерфейсу Subject, так что заместитель всегда может быть подставлен вместо реального субъекта;

– контролирует доступ к реальному субъекту и может отвечать за его создание и удаление;

– прочие обязанности зависят от вида заместителя:
- удаленный заместитель отвечает за кодирование запроса и его аргументов и отправление закодированного запроса реальному субъекту в другом адресном пространстве;
- виртуальный заместитель может кэшировать дополнительную информацию о реальном субъекте, чтобы отложить его создание;
- защищающий заместитель проверяет, имеет ли вызывающий объект необходимые для выполнения запроса права.

Subject - субъект:

– определяет общий для Real Subject и Proxy интерфейс, поэтому класс Proxy можно использовать везде, где ожидается RealSubject.

Real Subject – реальный субъект:

– определяет реальный объект, представленный заместителем.

Стоит отметить, что при разработке информационной системы документооборота с использованием трехзвенной архитектуры клиент-сервер реализация шаблона Proxy для справочника находит место, как на сервере приложений, так и на стороне клиента. На сервере приложений можно реализовать загрузку в память объекта при первом обращении, блокировки, права доступа к объектам системы, подсчет числа ссылок на объект, для того чтобы выгружать объект из памяти. На стороне клиента тоже можно реализовать загрузку

в память объекта при первом обращении, а также реализовать еще одну оптимизацию. Это связано с копированием измененного объекта: объект копируется только в том случае, если он действительно был изменен.

Выводы

В данной работе рассмотрен вопрос об объектно-ориентированном проектировании справочников. Нами предлагается некий шаблон проектирования справочников. На наш взгляд, при проектировании справочника наилучшим выходом является совместное использование шаблонов Composite и Proxy, компоновщик и заместитель. С помощью заместителя осуществляется косвенный доступ к объекту справочник и в его реализации хранится ссылка на другой объект справочник. Так же Proxy предоставляет клиенту интерфейс, совпадающий с интерфейсом заменяемого объекта справочник. Компоновщик занимается реализацией справочника на физическом уровне, то есть осуществлением основных методов, обязанностей, а заместитель – некой оптимизацией работы объекта справочник как на уровне клиента, так и на сервере приложений. Сочетание двух этих шаблонов дает оптимальный результат при проектировании справочников.

ЛИТЕРАТУРА

1. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования СПб.: Питер, 2001. 368 с.