ОБЪЕКТНО-ОРИЕНТИРОВАННЫЕ ТЕХНОЛОГИИ ПОСТРОЕНИЯ ИНТЕЛЛЕКТУАЛЬНЫХ РАСПРЕДЕЛЕННЫХ СИСТЕМ ОБУЧЕНИЯ САПР

Я.Е. Львович, Е.Н. Королев
Воронежский государственный технический университет

В настоящее время все более активно применяются объектно-ориентированные технологии как для анализа, концептуального проектирования, так и для построения сложных автоматизированных систем. Также актуальным является применение принципов объектно-ориентированного анализа для построения систем обучения автоматизированному проектированию, поскольку объектно-ориентированные методы характеризуются гибкой адаптацией к уровню знаний и взглядам специалиста, их применяющего, из-за отсутствия каких-либо формальных методов идентификации структуры предметной области, адекватных условиям решаемой задачи, и проблем с представлением результатов анализа в естественной форме.

Объектно-ориентированный подход к автоматизированному проектированию требует, чтобы любой метод моделирования, используемый в проектировании, имел объектно-ориентированный характер. Поэтому совершенно необходимо каким-то образом преодолеть существующую ограниченность методов объектно-ориентированного анализа (ООА) и адаптировать их к условиям проектирования. Существует несколько путей решения этой проблемы, но наиболее перспективным представляется подход, основанный на предположении, что описание объекта проектирования на каждом шаге проектирования есть описание разных состояний процесса его функционирования, а описание процесса обучения автоматизированному проектированию есть описание разных этапов обучения.

В рамках данного подхода предлагается применять следующие технологии при построении систем обучения САПР:

• Технологии создания распределенных информационных систем с использованием структурно-компонентного подхода на основе технологии Java2 Platform Enterprise Edition (J2EE).

• Аспектно-ориентированные технологии (АОП).

Сущность структурно-компонентного подхода к разработке автоматизированных систем заключается в ее декомпозиции на автоматизируемые функции: процесс проектирования или обучения проектированию разбивается на функциональные подсистемы, которые в свою очередь делятся на подфункции, подразделяемые на задачи, и так далее.

Процесс разбиения продолжается вплоть до конкретных процедур или методов, объединенных в классы. Классы изолированы друг от друга в силу инкапсуляции и могут быть независимо спроектированы и отлажены. В качестве двух базовых принципов методологии структурно-компонентного подхода к построению объектно-ориентированных систем используются следующие:

• принцип "разделяй и властвуй" - принцип решения сложных проблем путем их разбиения на множество меньших независимых задач, легких для понимания и решения, которые реализуются в виде объектов;

• принцип иерархического упорядочивания - принцип организации составных частей проблемы, представленных в виде объектов, в иерархические древовидные структуры с добавлением новых деталей на каждом уровне. Использование принципов наследования и полиморфизма позволяет легко наращивать сложность обучающих систем.

Предлагается реализация системы автоматизированного обучения системных пользователей САПР с использованием технологии построения распределенных компонентно-ориентированных систем с многоуровневой архитектурой J2EE. Данная технология позволяет реализацию активных Enterprise-компонент обучения в виде компонент Enterprise Java Beans (EJB), которые размещаются на сервере приложений. В этом случае управление компонентами берет на себя EJB-контейнер.

При размещении разработанных компонент обучения имеется возможность для каждого из них определить атрибут транзакции. Обычно контейнер начинает транзакцию непосредственно перед началом метода компонента обучения. Он подтверждает транзакцию перед выходом из метода. Каждый метод, соответствующий определенному этапу обучения, может быть связан только с одной транзакцией. Вложенные или многочисленные транзакции не разрешены внутри метода.

Вводимые атрибуты транзакции должны урегулировать отношения между методами, связанными с транзакциями. Они управляют областью видимости транзакции и могут иметь одно из следующих значений:

- Required;
- RequiresNew;
- Mandatory;
- NotSuported;
- Supports;
- Never.

Атрибут Required означает, что если обучаемый работает внутри транзакции и вызывает метод другого компонента, то этот метод выполняется в первоначальной транзакции обучаемого. Если обучаемый не работает в транзакции, контейнер запускает новую транзакцию перед выполнением метода.

Атрибут RequiresNew означает, что если обучаемый работает внутри транзакции и вызывает метод другого компонента, то контейнер выполняет следующие действия:

1. Приостанавливает первоначальную транзакцию.
2. Начинает новую транзакцию.
3. Посылает запрос в метод.
4. Продолжает первоначальную транзакцию обучаемого после завершения метода.

Если обучаемый не работает в транзакции, контейнер запускает новую транзакцию перед выполнением метода. Использовать атрибут RequiresNew необходимо, чтобы метод всегда выполнялся в новой транзакции.

Атрибут Mandatory означает, что если обучаемый работает внутри транзакции и вызывает метод другого компонента, то метод выполняется внутри первоначальной транзакции. Если обучаемый не работает в транзакции, то контейнер генерирует исключительную ситуацию.

Использовать атрибут Mandatory необходимо в тех случаях, когда метод вызываемого компонента должен использовать первоначальную транзакцию.

Атрибут NotSupported означает, что если обучаемый работает внутри транзакции и вызывает метод другого компонента, то контейнер приостанавливает первоначальную транзакцию перед вызовом метода. После завершения метода контейнер продолжает первоначальную транзакцию. Если обучаемый не работает в транзакции, то контейнер запускает новую транзакцию перед выполнением метода. Использовать атрибут NotSupported необходимо для методов, не требующих транзакций. Поскольку транзакции увеличивают накладные расходы, этот атрибут может улучшить производительность.

Атрибут Supported означает, что если обучаемый работает внутри транзакции и вызывает метод другого компонента, то метод выполняется внутри первоначальной транзакции. Если обучаемый не работает в транзакции, то контейнер не запускает новую транзакцию перед выполнением метода. Поскольку поведение метода в отношении транзакций может меняться, то использовать атрибут Supports необходимо с осторожностью.

Атрибут Never означает, что если обучаемый работает внутри транзакции и вызывает метод другого компонента, то контейнер генерирует исключительную ситуацию. Если обучаемый не работает в транзакции, то контейнер не запускает новую транзакцию перед выполнением метода.

Атрибуты транзакции можно указать для компонента обучения в целом или для его отдельных методов. Если указывается один атрибут для метода, а другой для компонента, атрибут для метода имеет преимущество.

Существует два способа произвести откат управляемой контейнером транзакции. Первый - при возникновении системной исключительной ситуации контейнер автоматически произведет откат транзакции. Второй - при невыполнении плана обучения, то есть при ошибке прохождения контрольного теста или при возникновении тайм-аута.

Когда контейнер производит откат транзакции, он всегда отменяет изменения статистических данных хода обучения, выполненного внутри транзакции, оставляя лишь информацию о произошедшем откате.

Реализовать программный комплекс для обучения системных пользователей САПР с возможностью использования указанных типов транзакций можно, используя технологию построения распределенных компонентно-ориентированных систем с многоуровневой архитектурой J2EE (Java2 Platform Enterprise Edition). При размещении компонент на сервере приложений допускается возможность заключать методы компонент в транзакции с указанными атрибутами. В этом случае управление транзакциями берет на себя EJB-контейнер.

При размещении компонента обучения, представленного в виде Enterprise Java Beans, в соответствующем конфигурационном xml-файле необходимо указать, какой из методов компонента связан с транзакциями, при помощи установки атрибутов транзакции. При размещении компонент обучения на большинстве J2EE-совместимых серверах, конфигурационный файл называется ejb-jar.xml. Указать тип транзакции можно для каждого метода. Например, указать тип транзакции Required для метода employment, относящегося к компоненту (EJB) LevelBeanTraining, можно следующим образом:

```
<container-transaction>
<method>
 <ejb-name> LevelBeanTraining</ejb-name>
 <method-intf>Remote</method-intf>
 <method-name> employment
</method-name>
        <method-params>
        <method-param>int</method-param>
         </method-params>
</method>
<trans-attribute>Required</trans-attribute>
</container-transaction>
```

Другой важной функцией EJB-контейнера является функция оптимизации использование памяти сервера, а также обеспечение устойчивости к сбоям. Эта функция осуществляется путем активизации и деактивизации EJB-компонент.

Технология АОП ставит своей целью разработать механизм реализации сквозной функциональности в объектно-ориентированных обучающих системах. АОП не заменяет собой объектно-ориентированное проектирование (ООП), данная техника всего лишь достраивает концепции ООП. Под АОП предоставляется новый механизм композиции, отличный от имеющихся в ООП. Это новая технология, позволяющая компоновать сложные системы из взаимно пересекающихся блоков (crosscutting concerns). АОП вводит в действие аспекты (aspects). Они инкапсулируют особенности поведения взаимно влияющих друг на друга этапов обучения, представленных в виде классов, в составе повторно используемых обучающих модулей.

Предлагается использовать технологию АОП как дополнение принципов объектно-ориентированного проектирования, обогащая его другим типом модульности, который позволяет локализовать код реализации crosscutting-логики в одном модуле. Такие модули, содержащие реализацию определенного раздела обучения, будем обозначать термином *аспекты обучения*, от аспектно-ориентированного программирования. За счет аспектно-ориентированного кода работа с crosscutting-отношениями упрощается. Аспекты обучения в системе могут изменяться, вставляться, удаляться на этапе компиляции и, более того, повторно использоваться.

Реализация объектно-ориентированной системы обучения хорошо вписывается в технологию аспектно-ориентированного проектирования. Если в качестве JoinPoint рассматривать цепочку обучения на определенном этапе обучения, то before advice может содержать предварительный анализ знаний, необходимых для изучения текущего этапа, а after advice может содержать итоговый тест контроля качества обучения. В общем, аспекты добавляют дополнительную функциональность в точки обучения пользователя (pointcut) через advice. Важно то, что pointcut могут собирать не только точки выполнения, то есть этапы обучения, но и контекст, в которых эти точки находятся.

В качестве примера рассмотрим фрагмент кода, предназначенный для обучения определенного этапа проектирования и формирование log-файла, содержащего отчет по обучению, реализованный на AspectJ. Для каждого обучаемого предлагается включать реализацию log-вызовов в начало и в конец каждого метода. Кроме того, требовалось заносить в log-файл значения параметров каждого метода. Следование этим соглашениям требовало существенных усилий со стороны разработчика. При использовании AspectJ большое количество вызовов может быть заменено одним аспектом, который автоматически регистрирует и параметры, и возвращаемые значения наряду с входами и выходами метода. Модель обучения, отвечающая этим требованиям, можно представить в виде следующего аспекта:

```
public aspect AutoLog{
pointcut publicMethods() : execution(public * sapr.lession.*(..));
pointcut testCalls() : execution(* TestLession.*(..));
pointcut allCalls() : publicMethods() && logObjectCalls();
before() : allCalls(){
Logger.entry(thisJoinPoint.getSignature().toString());
}
after() : allCalls(){
Logger.statistic(thisJoinPoint.getSignature().toString());
}
}
```

Проанализируем этот пример и посмотрим, какие действия осуществляет аспект. Объявление аспекта подобно объявлению класса и, так же как класс, определяет тип Java. Кроме того, аспект содержит конструкции *pointcut* и *advice*. Прежде всего рассмотрим, что представляет собой *join point* (точка соединения). Точки соединения в реализации обучающей системы - это однозначно определенные контрольные точки при выполнении программы обучения. Под точками соединения реализации системы обучения в AspectJ подразумеваются: вызовы методов, точки обращения к членам класса и исполнение блоков обработчиков исключений и т.д. Точки соединения могут, в свою очередь, содержать другие точки соединения. Например, результат вызова метода может передаваться каким-то другим методам. A pointcut является языковой конструкцией, которая отбирает множество точек соединения на основании определенного критерия, например по принадлежности к одному разделу или одной тематики. В приведенном выше примере первый pointcut под именем publicMethods выбирает исполнения всех public методов в пакете sapr.lession. То есть этот pointcut включает все разделы по тематике САПР. Подобно int, который является базовым типом Java, Execution является базовым pointcut. Он выбирает исполнения методов, соответствующих сигнатуре, заданной в скобках. Для сигнатур допустимо включение символов шаблонов: в приведенном примере оба pointcut-а содержат несколько таких символов. Второй pointcut с именем testCalls выбирает все исполнения методов в классе TestLession. Третий pointcut allCalls, объединяет два предыдущих, используя && , что означает выбор всех public методов из пакета sapr.lession и из класса TestLession. Теперь, после того как в аспекте определены точки, нужно использовать конструкцию advice, чтобы выполнить текущую регистрацию. Advice - это фрагмент кода, выполняющийся до, после или в составе точки соединения. Advice определяется для pointcut, что представляет собой нечто наподобие указания "выполнить этот код после каждого вызова метода, который надо регистрировать".

Таким образом, мы добьемся, что обучаемый, прежде чем приступить к изучению определенных раз-

делов, реализованных в классах пакета sapr.lession, или начать тестирование методами класса TestLession, должен пройти авторизацию, реализованную в методе entry класса Logger. После соответствующего обучения или тестирования метод statistic класса Logger запишет результаты данного этапа обучения.

АОП удобно использовать для контроля и управления ходом обучения. Очевидно, что аутентификация обучаемого перед обучением определенного раздела нужна не для всех разделов программы обучения; следовательно, нам необходимо сделать следующее - перед вызовом разделов обучения, для которых аутентификация необходима, мы будем проверять пользователя, и в случае если он не аутентифицирован, то перенаправляем запрос на аутентификацию.

Итак, АОП при правильном использовании может следующее:

- уменьшить объем кода системы обучения (следовательно, снизить вероятность программных ошибок);
- улучшить дизайн системы с точки зрения реализации сквозной функциональности, улучшить модульность;
- упростить код системы, благодаря локализации кода, не относящегося к основной функциональности;
- упростить тестирование (можно тестировать различные аспекты обучения отдельно, а только потом - вплетенные в систему). Улучшить управляемость кода; как следствие - простота эволюции и сопровождения;
- увеличить количество повторно используемых модулей благодаря слабой связности подсистем.

В данной статье предложены достоинства применения технологии J2EE и технологии аспектно-ориентированного проектирования для реализации распределенной системы дистанционного обучения автоматизированному проектированию и наглядно продемонстрировано, как можно применять некоторые правила на практике.

Реализовать программный комплекс для обучения системных пользователей САПР можно, используя технологию построения распределенных компонентно-ориентированных систем с многоуровневой архитектурой J2EE (Java2 Platform Enterprise Edition) в интеграции с технологией АОП. Данные технологии позволяют реализацию компонент обучения в виде компонент Enterprise Java Beans (EJB), которые выполняются под управлением EJB-контейнера. EJB-контейнер загружает модель жизненного цикла обучения автоматизированному проектированию и управляет Enterprise-компонентами обучения согласно загруженной модели, и в соответствии с требованиями аспектов.

Литература

1. *Буч, Г.* Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, А. Джекобсон. - М. : ДМК Пресс, 2000. - 432 с.

2. *Stein, D., Hanenberg, S., Unland, R.* Designing Aspect-Oriented Crosscutting in UML. In proc. of Workshop on Aspect-Oriented Modeling with UML at AOSD, 2002.

3. *Aldawud, O., Elrad, T., Bader, A.* A UML Profile for Aspect-Oriented Modeling. In proc. of Aspect-Oriented Programming Workshop at OOPSLA, 2001.