Свойство-ориентированное программирование (СОП)

Рассмотрим сначала некоторые конкретные ситуации, возникающие при создании бизнес-приложений, и на их примере продемонстрируем преимущества предлагаемой технологии:

Изменение логики свойства

Так как очень часто при автоматизации предприятия используется итеративный подход, то есть модули внедряются постепенно, возникают ситуации, когда необходимо подменять одну действующую логику более общей или наоборот более конкретной. Например, на какой-то стадии цена на продукцию вводилась маркетологом вручную, а затем было принято решение по определенным группам ставить цену равную минимальной цене конкурентов минус 5 процентов. В существующих технологиях, так как большая часть логики заложена в запросах SQL (в противном случае критично падает производительность из-за невозможности использования оптимизатора), такая операция требует поиск и изменение запросов, задействующих эту логику (в том числе в UI), во всем приложении. Соответственно это операция очень трудоемкая и дорогостоящая. В СОП это можно сделать одним действием просто подменяя одну реализацию свойства другой (как это делается в ООП).

Пример:

Старая логика:

productPrice = addDProp(double,product);

Новая логика:

productPrice = addJProp(removePercent, addMGProp(priceCompetitor, 1), value(5)) ,где productPrice – цена продажи, priceCompetitor – цена конкурента.

Добавление ограничений

Обычно в системах кроме обычной логики ввода, целостность также может включать в себя некоторые ограничения, накладываемые на все данные в системе. Например, руководство во избежание ошибок не хочет, чтобы на предприятии остатки уходил в минус, или вводимые штрих-коды были не уникальны. Также бывают существенно более сложные случаи - если предприятие занимаемся поставками, производством и продажами так сказать “с колес” и во все моменты времени необходимо обеспечить наличие необходимых товаров и исполнителей. Или, пример из жизни - когда из-за особенностей законодательства заказчик хочет ограничить по всей сети общую сумму продаж по одному из своих юрлиц. Строго говоря, описанные задачи одни из самых сложных в программировании бизнес-приложений, потому как требует от разработчика следить за каждым, влияющим на ограничение, параметром в отдельности и проверять, не нарушает ли его изменение общую логику. На практике на поддержку одного такого ограничения (с учетом отладки, тестирования, устранения последствий) может уходить до месяца. В СОП же это делается одним действием, ответственность за выполнения всей логики берет на себя платформа. Более того автоматически платформа прозрачно для разработчика может поддержать эту логику при вводе, не отображая пользователю те данные, которые он не может ввести.

Пример:

addConstraint(addJProp("Кол-во резерва должно быть не меньше нуля", greater2, vzero, balanceStoreFreeQuantity));

Создание форм

При создании пользовательского интерфейса в существующих системах, как правило, используется “документно-отчетный” подход – в нем все элементы делятся на *формы ввода* (документы), в которые вводятся первичные данные, и *отчеты*, где пользователь может получать агрегированную информацию. Обусловлено это, прежде всего, особенностями современных технологий, так за первые отвечает сервер приложений (императивная часть - Java/ABAP/1C-язык), а за вторые - сервер БД (декларативная часть - SQL). Однако для пользователей такой подход не удобен, так как анализ любой информации имеет целью принятие решений, которые, как правило, сразу же необходимо внести назад в систему. Кроме того бывают очень сложные решения, для которой нужно видеть большое количество сводной информации в одном месте из разных модулей. Реализовывать такие требования очень непросто, потому как необходимо вручную реализовывать всю динамику работы формы: реагировать на выбор объектов \ изменение полей пользователем, организовывать просмотр ограниченного числа записей (чтобы не качать всю базу) рисовать саму форму, обеспечивать безопасность и т.д. В СОП же используется принципиально другой подход - любая форма изначально рассматривается как множество скомпонованных объектов и свойств, за организацию поведения которых отвечает сама платформа. При этом в дальнейшем предполагается, что компоновать форму может и должен сам пользователь.

Пример

Отображает все свойства, которые существуют по товарам в разрезе форматов:

```
private class FormatArticleNavigatorForm extends NavigatorForm {

        protected FormatArticleNavigatorForm(NavigatorElement parent, int ID) {

                super(parent, ID, "Остатки по форматам");

                ObjectNavigator objFormat = addSingleGroupObjectImplement(format,
        "Формат", properties, allGroup, true);

                 ObjectNavigator objArt = addSingleGroupObjectImplement(article, "Товар",
        properties, allGroup, true);

                addPropertyView(objFormat, objArt, properties, allGroup, true);

        }

  }
```

Изменение физической модели

В классической теории база данных может находиться в различных, так называемых, “нормальных формах”. Чем более она нормализована, тем проще ее поддерживать и тем меньше она занимает места. С другой стороны в менее нормализованном виде, быстрее выполняются

операции JOIN и соответственно общее чтение из базы. В существующих технологиях логическая модель никак не формализована, поэтому структуру БД надо определять сразу до создания самой системы. При этом оценивать будущую общую производительность приходится только исходя из предположений и собственного опыта. Если в результате решение оказывается ошибочным, это приводит к катастрофическим последствиям – надо переписывать приложение, останавливать сервер, переносить вручную данные из старой модели в новую и т.д. В СОП логическая модель отделена от физической. То есть сначала задается бизнес-логика и первоначальное отображение ее на физическую модель, после чего уже в процессе эксплуатации системы администратор (или впоследствии сама система) на основе накопленной статистики (как соотношения количеств записи и чтений) определяет, какая именно физическая модель будет оптимальной.

Пример

Сделать, чтобы все свойства для пары (товара, заказ) лежали в отдельной таблице:

```
tableFactory.include("articleorder",article,order);
```

Сделать, чтобы остатки постоянно хранились в базе, а не пересчитывались каждый раз:

```
addPersistent(balanceStoreFreeQuantity);
```

Наследование бизнес-логики

Как и в программировании, в бизнес-логике часто возникают проблемы с наследованием. В качестве примера можно привести необходимость добавить определенный вид продажи – скажем в розницу по безналичному расчету, который должен поддерживать дополнительный функционал расчетов, и при этом иметь функционал обычный продажи. В большинстве существующих технологий наследование поддерживается очень ограничено – только на уровне сервера приложений и только одиночное и только для одного объекта, (а в некоторых как 1С вообще не поддерживается), что делает его использования практически невозможным. Поэтому конкретные приложения получаются запутанными и с большим количеством “дублирующего” кода. В СОП же наследование максимально обобщено и поддерживается на самом базовом уровне работе с данными, поэтому решения получаются максимально структурированными и расширяемыми.

Также благодаря возможности визуальной настройки СОП позволяет решать задачи классификации прикладных объектов (например товаров, услуг и т.д.) на качественно ином уровне. Так пользователь может создавать прикладные классы, например, “алкоголь” для которого определять, что для него может\должна задаваться крепость, а затем наследовать от него “вино” и “пиво”, для первого из которых надо задать вкус (сухое, сладкое) и цвет (красное, белое) а для второго тип (светлое, темное). При этом эти “пользовательские “ свойства как и любые другие можно дальше использовать в бизнес-логике - группировать , включать в ограничения, отображать на формах и т.д.

Общий пример

Подытоживая все вышесказанное, в приложении представлен “настроечный код” (который впоследствии будет создаваться мышкой, и сохраняться в XML) приложения, которое автоматизирует все торгово-логистические операции – управления закупками, продажами, распределением, ценообразованием, акциями и т.д., в далеко не самой простой постановке с

поддержкой статусов (распределением ответственностей), глобально-партионным учетом и т.д. Если условно оценить общее количество строк в нем, то получим что - настройка классов задается в 70 строках, свойств - в 150 строках, и форм – в 300 строках. Причем аналогичный код на современных технологиях даже с гораздо меньшими нефункциональными требованиями, будет содержать не менее чем в 20 раз больше строк.

Выводы

Важно понимать, что в СОП речь идет не о решении каких-то частных случаев (как это обычно делается), это следствие системного, можно даже сказать научного, подхода к проблеме и создания принципиально новой, абсолютно декларативной парадигмы программирования, в основе которой лежит функциональное, а не автоматное программирование. Проводя аналогию, можно сказать, что все остальные платформы \ технологии пытаются лечить симптомы, мы же лечим болезнь.

Также в качестве предпосылок такого положения вещей, стоит отметить принципиальную ошибку при создании ООП – привязать атрибуты к одному объекту. Она была обусловлена реализацией первых языков поддерживающих ООП (в частности самого популярного C++) и именно она во многом привела к остановке развития программирования как такового. Впрочем, при создании SQL также было допущено немало ошибок, из-за чего он так и остался на уровне реляционной алгебры. Так что очень тяжело выделить одну единственную причину, почему технологии в программировании остановились на уровне 80-х годов.

Если говорить о СОП, то оно фактически обобщает SQL до чисто функционального языка (парадигмы), добавляя в него поддержку наследования, и устраняя все его недостатки. Кроме того включает в себя формализм для построения пользовательских интерфейсов на основе созданной парадигмы.

Если смотреть в чуть более далекую перспективу СОП позволяет решать многие смежные задачи опять-таки на качественно новом уровне.

Интеграционные сервисы

Очень интересно применение СОП в задачах B2B (business-to-business) интеграции. В качестве примера можно привести необходимость размещать сделанные заказы у поставщика или наоборот обеспечивать в программе его актуальный прайс. В таком случае если у поставщика тоже стоит система, основанная на СОП, для организации такого взаимодействия, достаточно всего лишь указать что такое-то свойство в вашей системе, совпадает с таким-то свойством в системе поставщика. Система сама возьмет на себя функции по отслеживанию изменений, отсылки их во внешнюю базу, проверки ограничений, обновления постоянно хранимых свойств и т.д.

Модель описания бизнес-логики

Предлагаемую модель программирования, в том числе, можно рассматривать как модель описания бизнес-логики. Она конечно более конкретная, чем обычные диаграммы бизнес-процессов, и требует более четкой проработки со стороны бизнес-аналитика. Но именно это и является ее плюсом, потому как не предполагает неоднозначных трактовок и является четким ТЗ к созданию бизнес-приложения. А если учесть что такое ТЗ может выполняться разрабатываемой

платформой, то в такой схеме исключается программист как ненужное промежуточное звено между предприятием и информационной системой.

Web-сервисы

В свете всех вышесказанных (и особенно последнего) преимуществ можно говорить о создании user-friendly Web-сервиса бизнес-приложений. В нем предприятие создает свой аккаунт, его бизнес-аналитики на основе типовой настройки донастраивают ее под свою бизнес-логику, после чего обычные пользователи могут начинать работу с сервисом, как если бы это было обычное классическое приложение, разработанное прикладными программистами специально для них. Таким образом, можно также реализовывать принцип - бизнес-приложение в аренду или лизинг. Такой подход также может заинтересовать в проекте крупные “сервис-ориентированные” интернет-компании, такие как Google, Yahoo, Yandex и т.п.

Технологические особенности

Платформа обеспечивает полную кросс-платформенность относительно используемых внешних программ – операционных систем (Windows, Linux) и СУБД (Postgre, Oracle, MS SQL и т.д.), разрабатывается с использование бесплатных открытых технологий и как следствие не требует никаких дополнительных затрат на ПО. Также использует минимальное (теоретически возможное) общение между 3 звеньями, что позволяет использовать в качестве клиента, в том числе и мобильные устройства с мобильными каналами связи, а также легко и дешево наращивать кластера серверов при большом росте числа пользователей.

Таким образом, можно говорить, что с технологической точки зрения платформа (и СОП в том числе) является практически совершенной, как следствие не может быть подвержена моральному устареванию (как это было с программами 10-летней давности).