# ОПЫТ ПРИМЕНЕНИЯ СЕРВИС-ОРИЕНТИРОВАННЫХ АРХИТЕКТУР В РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

*А. С. Даниленко, М. А. Марунина, В. И. Гунин*

ФГУП «РФЯЦ-ВНИИЭФ», г. Саров Нижегородской обл.

Существует несколько подходов к организации архитектуры программного обеспечения: модульное программирование, распределенные вычисления, программное обеспечение с сервис-ориентированной архитектурой **SOA**.

Создавая комплексы программ, разработчики программного обеспечения часто сталкиваются с проблемой наращивания или сокращения функций программы, повторного использования отработанных решений для различных заказчиков.

В случае разработки программного обеспечения как целостной неделимой структуры приходилось бы переписывать программное обеспечение для разных заказчиков.

**SOA** позволяет создавать комплексы программ функционирования с заданным количеством функций из ранее созданных сервисов, обеспечивающих работу той или иной задачи.

SOA представляет собой гибкий набор принципов проектирования, применяемых на этапах масштабирования и интеграции систем. Система, развернутая на базе SOA, предоставляет свободно интегрируемый комплекс сервисов, которые могут быть использованы во многих областях деятельности.

SOA определяет, как интегрировать приложения, разработанные для разных платформ в web-пространстве. Вместо определения интерфейса прикладного программирования API SOA определяет интерфейс в терминах протокола и функциональности приложения.

Сервис-ориентация не учитывает связей сервисов с операционными системами и прочими технологиями, лежащими в основе приложений. SOA разделяет функциональность в различные модули или сервисы, которые можно сделать доступными через сеть, что позволит пользователям комбинировать и использовать их для создания приложений. Данные сервисы и связанные с ними потребители общаются друг с другом, передавая информацию в четко определенном общем формате или координируя активность двух и более сервисов [1–3].

Например, несколько отделов в компании разработки программного обеспечения могут разрабатывать и разворачивать SOA-сервисы на различных языках программирования, при этом их клиенты получат преимущество легкого, понятного и хорошо определенного интерфейса доступа к данным. Для взаимодействия с SOA-сервисами принято использовать xml-формат обмена данными, но это не является обязательным требованием.

SOA можно представить себе как своего рода непрерывную совокупность самостоятельных сервисов, в противовес распределенным вычислениям или модульному программированию. Это означает, что один и тот же сервис может использоваться в нескольких комплексах, которые будут обращаться к нему через постоянный адрес, и точно так же один сервис может объединять информацию, полученную от нескольких сторонних сервисов.

Таким образом, сервисы образуют единую непрерывную сеть, на которую опираются реализации SOA. Сервисы представляют собой не объединенные, слабо связанные единицы функциональности без встроенных вызовов друг к другу. Каждый сервис выполняет одно действие, например, онлайн заполнение приложения для счета или онлайн просмотр банковского баланса, или заказ авиабилетов по сети. Вместо того, чтобы содержать встроенные вызовы друг к другу в исходном коде, сервисы используют определенные протоколы, описывающие, как сервисы передают и анализируют сообщения, применяя метаданные.

Разработчики SOA объединяют отдельные SOA-объекты, используя оркестрацию, т.е. исполнение специальным контейнером логики связывания нескольких сервисов [2]. В процессе оркестрации разработчик объединяет программную функциональность (сервисы) в неиерархической компоновке, применяя инструментарий, содержащий полный перечень доступных сервисов, их характеристики и средства для создания приложения, использующего эти источники.

Метаданные сервисов должны быть достаточно детализированы, чтобы описывать не только характеристики сервисов, но также данные, управляющие ими. Программисты расширили использование расширяемого языка разметки XML в SOA, чтобы упорядочивать данные в рамках описанного контейнера. Язык описания web-сервисов (WSDL) обычно описывает сами сервисы, в то время как протокол SOAP описывает протоколы связи. SOA зависит от данных и сервисов, описанных при помощи метаданных, которые должны отвечать следующим критериями:

1. Метаданные должны включаться в форме, которую программные системы смогут использовать для динамического конфигурирования, находя и

объединяя описанные сервисы, в то же время поддерживая их логичность и целостность. Например, метаданные могут быть использованы другими приложениями в качестве каталога, предоставляя возможность самоопределения сервисов без необходимости модификации функционального контракта сервиса.

2. Метаданные должны поступать в форме, которую разработчики системы могут понимать и управлять с приемлемыми затратами сил и средств.

SOA предоставляет разработчикам возможность объединять довольно большие части функциональности, чтобы формировать для того или иного заказчика приложения, почти целиком построенные из существующих программных сервисов. Чем больше эти части, тем меньше требуется точек взаимодействия, чтобы обеспечить любой требуемый набор функциональности; в то же время слишком большие части могут оказаться неприменимы для дальнейшего использования. Каждый интерфейс влечет за собой некоторое количество дополнительной обработки, поэтому требуется тщательное рассмотрение выбора степени структурирования функциональности. Предполагается, что в перспективе маргинальная стоимость создания N-ного приложения станет невысокой, так как все требуемое программное обеспечение (ПО) уже существовало в составе других приложений. В идеале для производства нового приложения потребуется только оркестрация.

Основные правила для разработки, поддержки и использования SOA определяются следующими ведущими принципами:

1. Повторное использование, степень структурирования, модульность, компонентное представление и способность к взаимодействию.

2. Соответствие стандартам (общим и характерным для отрасли).

3. Идентификация и категоризация, подготовка к работе и передача заказчику и мониторинг сервисов.

Конкретная тематика работы влияет на внутреннее поведение системы, стиль ее проектирования, а также на архитектурные принципы проектирования и определения служб:

1. *Инкапсуляция сервисов* — множество сервисов объединены для использования под SOA. Часто такие сервисы не были изначально предназначены для SOA.

2. *Слабые связи между сервисами* — сервисы поддерживают отношения, минимизирующие зависимости, и требуют только осведомленности друг о друге.

3. *Контракты сервисов* — сервисы придерживаются соглашений о коммуникациях, определенных в одном или нескольких документах, описывающих сервисы.

4. *Абстракция сервисов* — сервисы прячут логику работы от внешней среды, за исключением описаний в контрактах сервисов.

5. *Повторное использование сервисов* — логика разделена между сервисами для дальнейшего использования.

6. *Компонуемость сервисов* — коллекции сервисов могут быть скоординированны и собраны в составные сервисы.

7. *Автономия сервисов* — сервисы контролируют инкапсулируемую в них логику.

8. *Оптимизация сервисов* — при всех прочих равных, сервисы более высокого качества обычно предпочтительнее низкокачественных.

9. *Открытость сервисов* — сервисы спроектированы быть внешне наглядными, чтобы их можно было найти и получить к ним доступ через доступные механизмы поиска.

10. *Релевантность сервисов* — функциональность, представленная на уровне модульности, представляется существенной для пользователя.

Рассмотрим частный случай применения сервис-ориентированного программирования, а именно модуль договоров, являющийся частью комплекса программ функционирования «АРМ Метролога».

Для отделения модуля договоров от проекта «АРМ Метролога» необходимо выполнение следующих условий:

1. Доступ к базе данных без участия программных компонентов «АРМ Метролога».

2. Автономное функционирование модуля договоров.

3. Прозрачный для пользователя доступ к модулю договоров из программы «АРМ Метролога»:

– автоматическая регистрация,

– возможность встраивания в проект.

4. Возможность дальнейшего расширения функциональности модуля.

5. Возможность использования отдельных служб или всего модуля для других проектов подразделения.

При проектировании модуля договоров было решено использовать сильное функциональное дробление сервисов, что обусловлено спецификой поставленной задачи. В частности, модуль договоров должен обеспечивать учет многих видов документов, каждый из которых имеет особую структуру и регистрируется в отдельной таблице базы данных. Соответственно каждый сервис реализует отдельное действие над отдельным документом.

Отдельные сервисы представляют собой реализацию таких функций, как выборка некоторой информации из базы данных по данным условиям, изменение или удаление отдельной записи и т. д.

Далее сервисы, относящиеся к работе с определенными данными, объединяются при помощи оркестрации в составные сервисы, снабженные удобным рабочим интерфейсом, выполненном в стиле всего проекта. Эти составные сервисы предоставляются для работы пользователям.

Рассмотрим особенности реализации приложения с использованием web-сервисов на примере одного из простейших составных сервисов конечного приложения – сервиса работы с поставками (см. рис.).

1. *Сервис добавления документа в базу данных* является одним из общих сервисов модуля и применяется

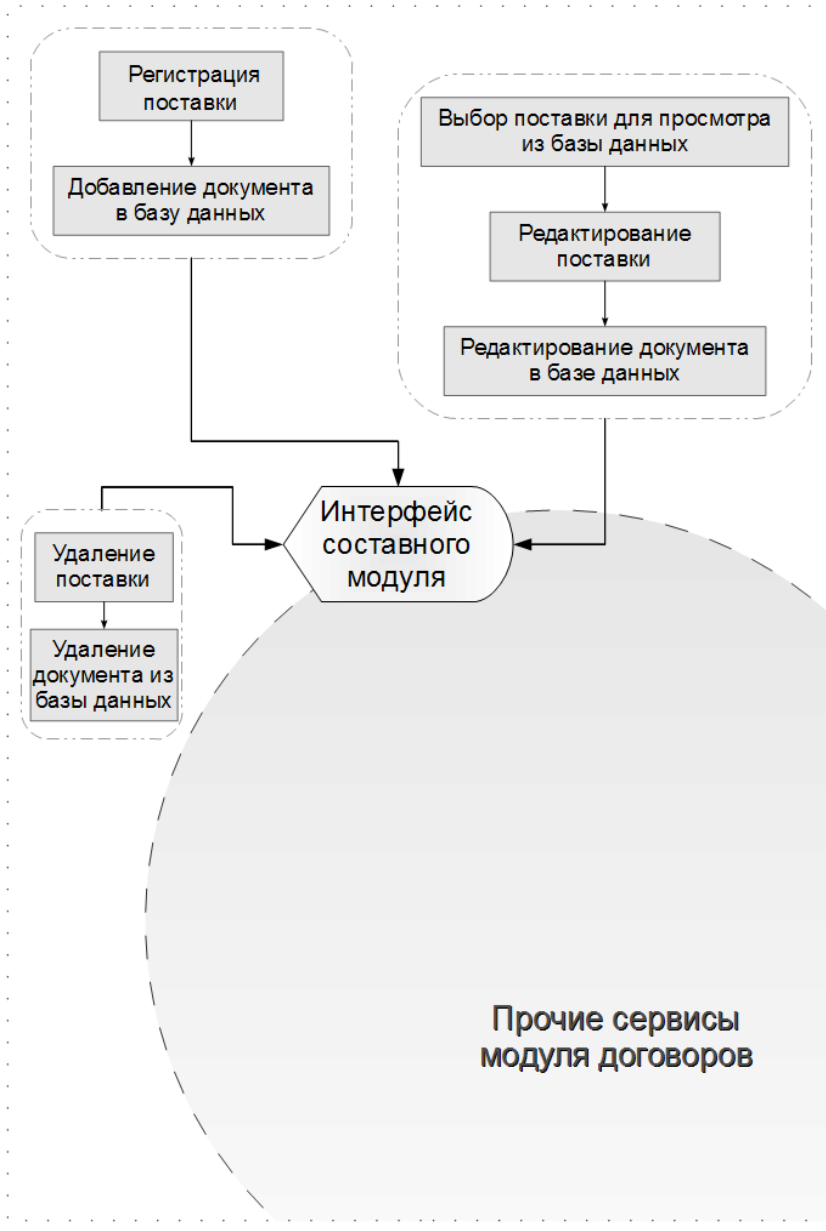


Рис. Модуль договоров

в нескольких составных сервисах. Он принимает данные о платежных документах, регистрирует документ в соответствующей таблице и возвращает вызывавшему сервису id зарегистрированной записи в базе данных.

2. *Сервис регистрации поставки* принимает данные, введенные пользователем, обрабатывает и проверяет их на правильность ввода и регистрирует в базе данных, используя сервис добавления документа.

3. *Сервис выбора поставки для просмотра* выбирает из базы данных сведения об одной из поставок по id и выводит их пользователю для просмотра и редактирования.

4. *Сервис редактирования документа в базе данных* является одним из общих сервисов модуля и применяется в нескольких составных сервисах. Он позволяет редактировать данные о соответствующем платежном документе по id записи.

5. *Сервис редактирования поставки* позволяет пользователю вносить изменения в данные выбранной с помощью сервиса выбора поставки. При редактировании также применяется сервис редактирования документа.

6. *Сервис удаления документа из базы данных* является одним из общих сервисов модуля и применяется в нескольких составных сервисах. Он удаляет из базы данных запись о платежном документе по id записи.

*Сервис удаления поставки* позволяет удалить из базы данных сведения о поставке и связанных с ней платежных документах. Использует сервис удаления документа из базы данных.

Для удобства работы с данными о поставках вышеперечисленные сервисы сведены в единый интерфейс, соответствующий интерфейсу всего проекта. При этом сервисы скрыты от пользователя, но при необходимости доступны разработчикам.

Пользователь видит перед собой только web-страницу, содержащую необходимую ему информацию и элементы управления, он может просматривать и редактировать информацию по своему усмотрению.

Комплекс программ функционирования создан в виде традиционного web-приложения, тогда как модуль договоров создан на основе сервис-ориентированного программирования, однако для пользователя разница не видна. Это происходит потому, что интерфейс модуля обеспечивает только представление в удобном виде полученных при помощи сервисов данных, соответственно это представление может быть любым.

Основным преимуществом подобного решения является то, что пользовательский интерфейс отделен от внутренней логики программы, поэтому при любых изменениях, например, структуры базы данных, нужно будет переписать только сервисы, не затрагивая интерфейс ПО.

Кроме того, однотипные сервисы (такие как сервисы добавления, редактирования и удаления документов) создаются один раз и затем применяются во всем модуле, что сокращает время разработки. К подобным сервисам можно обращаться как из других сервисов, так и непосредственно из пользовательского интерфейса, если возникнет такая необходимость.

В настоящее время общепринятым стандартом для обмена данными с web-сервисами является использование сообщений, основанных на XML. XML позволяет передавать информацию в упорядоченном виде, создавая сложные структуры данных, поддающиеся автоматическому разбору. Гибкость этого языка обеспечивает возможность его применения в любых программах.

В модуле договоров для обмена данными также применяется XML. Сервисы определяют формат получаемых и передаваемых данных, эта информация доступна разработчикам в описании сервисов, и они могут оперировать получаемой от сервисов и передаваемой сервисам информацией, чтобы представить ее в наиболее удобном для пользователя виде.

Использование XML для обмена сообщениями позволяет представить каждый сервис в форме «черного ящика», т.е. как объект, реагирующий определенным образом на определенное внешнее воздействие. Не нужно знать ничего ни о языке, на котором написан сервис, ни о платформе, на которой он располагается. Это позволяет при необходимости легко заменять сервисы или использовать их в других проектах, поскольку важно лишь сохранить формат сообщений.

Таким образом, использование технологии SOA позволило разработать программный модуль, удовлетворяющий всем выдвинутым условиям:

а) реализовать доступ к базе данных в качестве отдельной службы;

б) создать модуль договоров как автономную составную службу;

в) включить в комплекс ПО «АРМ Метролога» компонент, обеспечивающий доступ к модулю договоров и преобразующий ответные сообщения в единообразный для «АРМ Метролога» внешний вид;

г) функциональность модуля при необходимости может быть расширена с помощью добавления новых служб;

д) отдельные службы модуля или модуль целиком могут быть встроены в любую систему, взаимодействующую на базе интернет-протокола.

## Литература

1. Архитектура SOA как она есть. Л. Маквитти. – Сети и системы связи (www.ccc.ru).

2. Реализация бизнес-процессов в SOA. Д. Фейгин. – Открытые системы (www.osp.ru).

3. Моделирование SOA. – СМ-Консалт (cmcons.com).