Прокофьева И.В., Шибанов С.В., Шашков Б.Д.

АНАЛИЗ СОВРЕМЕННЫХ ТЕХНОЛОГИЙ И СРЕДСТВ ИНТЕГРАЦИИ ДАННЫХ В КОРПОРАТИВНЫХ СИСТЕМАХ

Интеграция данных является одним из основных инструментов обеспечения функционирования корпоративных систем. Выбор технологии, методологического подхода и инструментальных средств, которые позволят успешно решать все интеграционные потребности сложной информационной системы, является непростой задачей. В работе проведен анализ современных средств интеграции данных, обоснован выбор технологии интеграции данных и описана реализация соответствующих программных средств в системе электронных платежей.

Интеграция данных является одним из основных инструментов обеспечения функционирования корпоративных систем. Выбор технологии, методологического подхода и инструментальных средств, которые позволят успешно решать все интеграционные потребности сложной информационной системы, является непростой задачей. В настоящее время существует множество технологий, моделей и средств интеграции данных. В работе проведен анализ современных средств интеграции данных. Также обоснован выбор технологии интеграции данных и описана реализация соответствующих программных средств в системе электронных платежей.

Обмен файлами – это наиболее распространенный способ взаимодействия приложений. Скорее всего, это связано с тем, что этот способ является наиболее интуитивно понятным и простым в реализации. Также сейчас существует набор стандартных форматов (например, CVS (Common Separated Values) – поля разделенные запятыми). Данный метод имеет ряд серьезных недостатков. В случае необходимости обмена сложными структурами, простые форматы становятся непригодными. Необходима разработка специализированных форматов файлов, что приводит к большой зависимости систем друг от друга. Также обмен с помощью файлов подразумевает наличие человека, который осуществляет выгрузку и загрузку файлов. Данный недостаток решаем путем организации задач, выполняемых по расписанию [1].

Часто можно столкнуться с задачей интеграции двух или нескольких баз данных, часть из которых являются базами данных унаследованных систем, другие – современные промышленные базы данных, например Oracle. В таком случае наиболее правильным решением будет использование шлюза, который позволит работать с унаследованной базой так, как будто это база Oracle. Данный вариант интеграции привлекателен своей простотой концептуального и архитектурного решений, а также высокой степенью надежности из-за четкого ограничения функциональности. Многие современные прикладные системы дают возможность работать с такими распределенными системами, как Oracle и Sybase, но они не могут взаимодействовать со старыми системами, такими, как TurboImage. Наилучшим выходом здесь является как раз использование шлюза БД. Совокупность продуктов, реализующих интеграцию на основе технологии шлюзов, можно разделить на следующие группы:

прозрачные шлюза, обеспечивающие доступ к унаследованным базам данных, на основе использования языка запросов SQL, т.е. эти шлюзы позволяют осуществить доступ к «чужим» данным, таким же образом, как и к своим;

процедурные шлюзы осуществляют обработку вызовов удаленных процедур в унаследованной программной системе;

менеджеры доступа обеспечивают поддержку доступа «чужих» СУБД посредствам языка запросов SQL;

службы репликаций позволяют выполнять репликации данных из «чужих» баз данных и обратно, используя прозрачные шлюзы.

Наибольший интерес из перечисленных компонентов представляют прозрачные шлюзы. Прозрачный шлюз – это некий «интеллектуальный программный канал», предоставляющий доступ к базам данных иных форматов, нежели основная. Шлюз обращается к унаследованной СУБД (например с помощью языка SQL, если это реляционная база данных), которая непосредственно и обеспечивает доступ к самой базе данных. Средства репликаций позволяют осуществить репликацию только измененных данных в целевую систему [2].

В настоящее время прочно заняла свои позиции технология Web-сервисов, которая представляется как удобное средство интеграции приложений, позволяющее легко реализовать межплатформенное взаимодействие. Но эта технология не может рассматриваться как общий подход к интеграции приложений по нескольким причинам: Web-сервисы непригодны для обработки больших объемов данных; в них отсутствуют средства поддержки транзакций; в момент взаимодействия интегрированные системы должны находиться в работоспособном состоянии. Непригодность для обработки больших объемов информации связано с тем, что все данные преобразуются в формат XML, что ведет к увеличению объема данных и нагрузки на систему в целом. Web-сервисы начинают «захлебываться», когда за одно взаимодействие нужно передать сотню килобайт (существует стандарт WS-Attachments, который позволяет обойтись без преобразования в XML формат, но ведущие поставщики программного обеспечения этот стандарт не поддерживают). В первоначальном варианте Web-сервисы не поддерживают механизм транзакций. Этот недостаток часто предлагают устранять с помощью выполнения компенсационных операций. Но никто не гарантирует успех компенсационной операции, помимо этого существуют системы не допускающие удаления данных, внесенных в базу. Существует стандарт, WS-Transaction, который должен решить данную проблему, но сейчас полноценные его реализации отсутствуют, те, что существуют сейчас, накладывают ограничения на выбор используемой платформы, подрывая тем самым основное преимущество Web-сервисов. Что же касается необходимости работоспособности взаимодействующих приложений, то можно сказать, что изначально удаленные вызовы были придуманы не для интеграции приложений, а для реализации распределенных систем, когда компоненты одной системы работают на разных машинах [1].

Сейчас очень много говорят про архитектуру, ориентированную на сервисы (Service Oriented Architecture, SOA). Свое начало SOA берет от Web-сервисов и достаточно долго приставка Web не отделялась от сервисов. Для того чтобы использовать сервис, необходимо послать ему сообщение с контактной информацией, получив которое, как правило, он должен отправить подтверждение. Предполагается, что сервис выполняет какое-либо бизнес-действие (business concept), какую-то отдельную функцию. Завершив свою работу, сервис может вызвать другой сервис. В сервисной архитектуре нет жестких связей между модулями. Их заменяют, так называемой, слабой связностью компонентов (loose coupling). Используя такую связь можно на ходу собирать из сервисов такую конфигурацию, которая необходима в данный момент. Так же наряду с обычными данными при обращении к сервису вызывающий модуль, который знает, как его вызвать и какую работу он выполняет, передает метаданные. Используя совокупность данных и метаданных, сервис выполняет заказ на обслуживание. SOA – это не какие-то отдельные продукты, а технология построения информационных систем, которые представляют из себя сервисы, доступные для наружного использования. Это означает, что, по сути, для пользователя ничего не изменилось и не изменится, пока ведущие производители программного обеспечения не модифицируют свои продукты, которые сейчас являются замкнутыми системами, в соответствие с концепциями SOA [3].

Еще одной представительницей сервисной архитектуры – архитектура, управляемая событиями (Event Driven Architecture, EDA), способная адаптироваться к потокам работ в бизнес-процессах. Здесь про-

цессы могут выполняться как синхронно, так и асинхронно. Данная архитектура является перепрограммируемой, причем перепрограммирование выполняется при возникновении изменений в бизнес-процессах. Также система, реализованная на основе EDA, способна воспринимать события из внешней среды. Сообщения о событиях, поступающие в систему, распространяются по подписчикам (subscriber, автомат или человек). Сообщение состоит из двух частей: заголовок, в котором содержатся метаданные (идентификатор спецификации события, тип события, имя события, отметка времени и источник); тело, описывающее то, что произошло. Получив сообщение о возникновении события, подписчики тем или иным образом реагируют на него: это может быть дальнейшая пересылка сообщения, вызов какого-либо сервиса и т.д. EDA – это распределенная и очень слабосвязанная архитектура (extreme loose coupling). Источник сообщения знает только, что сообщение передано, но не отслеживает его дальнейшую судьбу. В связи с этим в EDA применяются асинхронные потоки данных и работ. Архитектуры EDA разделяют по способам обработки событий:

простая обработка событий. В этом случае выполняется прямая передача событий из входного потока в процессор обработки событий;

потоковая обработка событий. Простые события передаются сначала в генератор, где путем обработки формируются значимые события, отправляемые в процессор событий;

обработка сложных событий. В этом случае простые события неоднородны и выполняется фильтрация, а также различные статистические методы.

Если система построена на основе технологии EDA, то она должна включать следующие четыре компонента:

генератор событий (event generator) выполняет первичную обработку и выделение значимых событий;

канал событий (event channel) обеспечивает передачу событий из генератора в процессор;

процессор событий (event processor) оценивает события, вырабатывая необходимые решения (такие как запуск сервисов), осуществляет рассылку подписчикам и архивирование событий;

управление последующими событиями (downstream event-driven activity). В ситуации, когда входное событие требует выполнения комплекса действий, предоставляет соответствующие средства управления [3].

Интеграционные брокеры (или интеграционные сервера) представляют собой промежуточные серверы в интеграционной среде, шлюзы, осуществляющие обработку потоков данных и сообщений, а также распределение данных между приложениями, имеющими различные интерфейсы. В ядре брокера должны храниться бизнес-правила, на основе которых, а также полученных данных, выполняются вычислительные операции, анализ и принятие решений. Использование брокеров позволяет вынести правила интеграции из кода конкретных систем в отдельные бизнес-правила, что упрощает изменение этих правил, ведет к их открытости и применению ко всем взаимодействующим системам. Также на брокеры возлагаются функции адресации и преобразования данных. Т.е. мы получаем некоторый коммутатор, гипермейнфрейм [7].

Следующий шаг в этом направлении совершила компания Sonic, которая предложила создание некоего метакомпьютера для всего предприятия, информационную магистраль предприятия (Enterprise Service Bus - ESB), но гораздо более дешевого и компактного, чем интеграционные брокеры. ESB стала новым подходом к организации основы средств интеграции. С использованием этой шины можно реализовать корпоративную магистраль, имеющую распределенную архитектуру, ориентированную на службы. ESB позволяет создавать контейнеры для размещения служб. Службы легко собираются и согласуются. Реальная физическая сеть может подвергаться изменениям, т.к. вся конструкция, размещенная в ней, является виртуальной. В процессе работы одна или несколько служб находятся в специальном контейнере (service container). С помощью контейнеров службы продвигаются в соответствие с маршрутом сообщений. К поступившему на вход ESB сообщению добавляется информация о маршруте, который обеспечивает контентно-управляемое продвижение по распределенному процессу. По этому процессу сообщение проходит через несколько служб и достигает конечной точки, для указания которых могут использоваться не физический, а логические имена. Сервисная шина предприятия обеспечивает поддержку дополнительных методов взаимодействия между хабом и приложениями, прежде всего поддержку протокола SOAP. ESB может быть использована для реализации идей EDA. Шина ESB выполняет функции промежуточного слоя между различными процессами, что делает возможным инициацию запуска сервисов, включенных в систему ее средствами, клиентами и событиями. Сервисная шина предприятия может рассматриваться как некий архитектурный шаблон, который реализуется разнообразными программными продуктами такими, как Sonic ESB и IBM WebSphere ESB [5].

Существует два основных подхода к организации взаимодействия интегрируемых систем. Первый – непосредственное взаимодействие каждой из системы по принципу «каждая с каждой», второй – взаимодействие через общий узел [1].

Примером первого подхода к интеграции данных является ETL (Extract-Transform-Load). Как видно уже из названия, эта метод позволяет извлекать данные из различных источников, преобразовывать в соответствие с целевым форматом полученные данные, загружать переработанные данные в целевое хранилище данных. Остановимся на каждом из компонентов ETL и рассмотрим их более подробно. Инициализация процесса ETL начинается с работы программы, производящей извлечение данных из исходного хранилище данных. В качестве таких программы можно использовать собственные, готовые специализированные средства ETL или сочетать оба этих варианта. В реальности чаще всего используется последний вариант, где применяются программные средства, написанные сторонними разработчиками и программы, реализующие специфический функционал для данного окружения. По окончанию извлечения данных, процедуры преобразования данных выполняют их подготовку к загрузке в целевом хранилище данных. Здесь могут выполняться такие операции, как объединение, перевод значений, создание полей и фильтрация. Наконец после завершения всех преобразований выполняется загрузка данных в целевую базу данных. Загрузка данных может выполняться периодически или постоянно. В первом случае данная операция выполняется с определенной периодичностью (ежемесячно, ежеквартально, ежедневно). Постоянная загрузка осуществляет оперативное предоставление данных в целевую базу данных. Чаще всего средства ETL поддерживают оба способа загрузки, а наиболее современные позволяют загружать только изменившиеся данные. Также средства загрузки разделяют по следующим методам тиражирования: push-тиражирование – выполняется «проталкивание» данных из исходного в целевое приложение; pull-тиражирование – «вытаскиваются» данные целевым приложением по мере необходимости [6]. Метод ETL целесообразно применять, если накоплено огромное количество информации за время существования организации, а также существует множество хранилищ данных. Помимо этого метод интеграции ETL также применяется при необходимости проверки данных, их корректировки, для интеграции разнообразных справочных данных [7].

Примером второго подхода к интеграции данных является EII (Enterprise Information Integration) или федерализация, позволяющая осуществлять интеграцию в режиме реального времени в момент поступления запроса от пользователя. Метод интеграции EII позволяет представлять множество источников данных в качестве единого виртуального хранилища. Реально же данные могут располагаться в базах данных, хранилищах, созданных с помощью метода ETL, файлах, приложениях. В то время, когда пользователь формирует запрос к виртуальному хранилищу, с помощью инструментов федерализации, осуществляется извле-

чение данных из источника, их интеграция в соответствие с правилами, определенными заранее и отправка результата инициатору запроса. Преобразования данных выполняется, как правило, в оперативной памяти сервера в момент их извлечения из источника. Этот метод позволяет реализовать принцип автономности подразделений компании и осуществлять единый контроль путем разграничения прав доступа. Плюсом EII является возможность расширения количества источников данных, изменения правил запросов в режиме реального времени. Данный метод интеграции может применять при оперативном составлении запросов и формировании отчетности. Так же EII очень удобен, если из-за соблюдения мер безопасности данные запрещено перемещать и копировать из операционной базы. Методология федерализации может использоваться на первом этапе, например для двух объединяющихся компаний, т.к. обеспечивает идеальную интеграцию. EII целесообразно применять, когда затраты на внедрение ETL превышают получаемые от нее выгоды. EII ориентирована на обработку относительно «легких» запросов, которые необходимо обработать немедленно, в то время, как ETL предназначена для обработки «тяжелых» агрегированных данных [7].

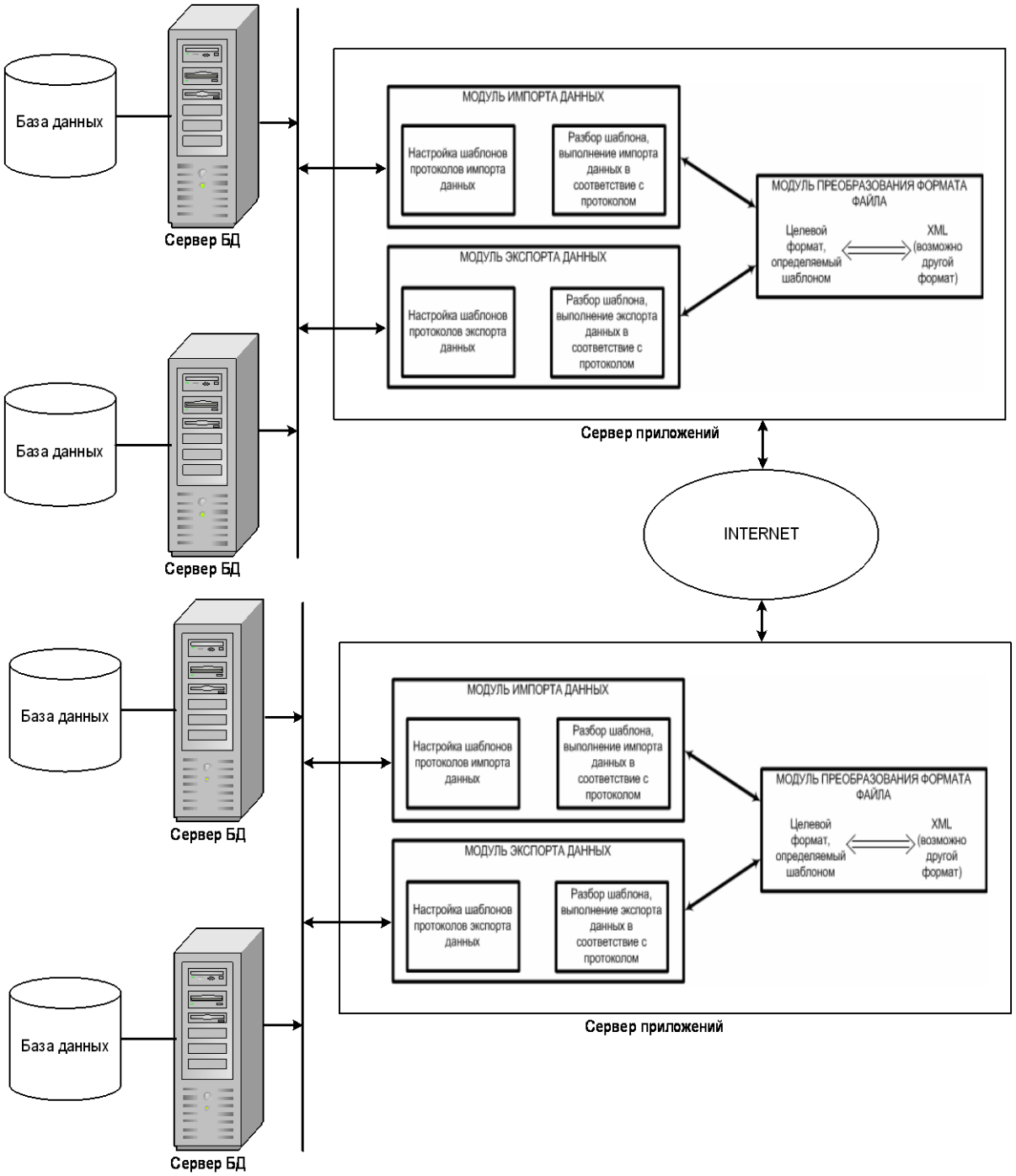

Рисунок 1 – Архитектура информационной системы «Электронный платеж»

Проблема интеграции разнородных систем в рамках корпоративной системы электронных платежей изначально решалась нами с помощью простого обмена файлами. Основной причиной выбора именно этой технологии интеграции систем стала универсальность: практически любые решения на разнородных платформах могут быть взаимосвязаны с помощью файлового обмена. Данный способ достаточно прост в реализации, но взаимодействие с различными корпоративными системами вело к необходимости поддержки широкого ряда различных протоколов обмена, причем, даже если различные корпоративные системы поддерживали единый протокол, могли требоваться различные форматы самих файлов (формат файла, текстовый, xml или др. мог различаться). Все это вело к необходимости реализовать для каждой новой корпоративной системы, с которой требовалось осуществить интеграцию, программное обеспечение, позволяющее формировать файлы, соответствующие тому или иному протоколу взаимодействия. Требовалась постоянная доработка системы обмена данными при необходимости интегрироваться с новой информационной системой, что вело к высоким накладным расходом и увеличению сроков интеграции. В связи с вышесказанным было предложено следующее

решение: разработана система соответствующей методологии ETL, осуществляющая работу с файлами, реализующими разнообразные протоколы обмена и в различных форматах.

Как и любая система, реализованная в соответствие с методологией ETL, наша система состоит из следующих модулей: модули экспорта, преобразования и импорта (рисунок 1).

Одним из звеньев описываемой системы является модуль преобразования формата файла. Его реализация выполняется с помощью стандартных средств, имеющихся во многих средах разработки программного обеспечения. Совокупность программных средств, реализующих преобразование данных в различные форматы, объединены в dll-библиотеку, к которой, в случае необходимости, выполняется обращение из программ разбора шаблонов протоколов [8].

Реализация модулей импорта и экспорта данных базируется на использовании метаданных. Метаданные просматриваются, создаются и редактируются пользователем с помощью программы настройки шаблонов протоколов, в которой можно выполнить определение метаданных, описывающих протоколы обмена данными. Описание протокола выполняется с помощью конструкций, задающих правила извлечения данных из БД, структуру файла протокола и правила проверки импортируемых данных [8].

Программные средства разбора шаблонов протоколов реализуют машинный анализ метаданных. Для определенного протокола здесь в соответствие с настройками, заданными шаблоном, выполняется генерация текстового файла, содержащего программный код, запуск которого приводит к созданию файла протокола. Полученная программа выполняет запрос параметров, определенных в шаблоне, осуществляет выбор данных, с помощью полученного запроса, создает файл протокола, структура которого также соответствует настройке, обращается, в случае необходимости, к модулю преобразования формата файла и отправляет итоговый файл по адресам из списка контактов [8].

Предлагаемая схема обмена данными в разнородной распределенной системе достаточно универсальна, но не лишена недостатков, например ограниченность набора целевых форматов файлов в модуле преобразования форматов. Так же, если модуль экспорта выполняет свою работу в соответствием с расписанием автоматических задач, имеющихся в системе, то модуль импорта запускается вручную. Данные проблемы необходимо решать, описанную систему можно развивать. А предложенный подход следует рассматривать как один из вариантов решения задач интеграции.

ЛИТЕРАТУРА

1. Алексей Добровольский, Интеграция приложений: методы взаимодействия, топология, инструменты. Открытые системы 2006, №9;
2. Глеб Лодыженский, Шлюзы, как средства интеграции баз данных. Открытые системы 1999, №2;
3. Леонид Черняк, EDA как очередная инкарнация SOA. Открытые системы 2006, №9;
4. Администрирование компьютерных сетей Интеграция приложений на основе WebSphere MQ. Интеграционный брокер и требования к его функциональности. admin-seti.ru/about/clause/298/679686/
5. Леонид Черняк, Общая шина предприятия. Открытые системы 2003, №4;
6. Скотт Стейначер, ETL – ключ к готовности и корректности данных. computerworld 2001, №4
7. Дмитрий Ломаки, Вместе — надежнее. Директор ИС 2007, №10;
8. Прокофьева И.В., Шибанов С.В., Шевченко О.А., Модель метаданных и программные средства обмена данными между разнородными информационными системами в системе электронных платежей. Научно-технические ведомости СПбГПУ 2008, №1