УДК 515.142.33

СЕРВЕР ПРОБЛЕМНО-НЕЗАВИСИМОЙ КОМПОНЕНТЫ СИСТЕМЫ УПРАВЛЕНИЯ ДОКУМЕНТАМИ

М.В. СТЕРЖАНОВ, И.В. БАЙДАКОВ

Белорусский государственный университет информатики и радиоэлектроники
П. Бровки, 6, Минск, 220013, Беларусь

Поступила в редакцию 13 марта 2012

Рассматривается задача разработки сервера проблемно-независимой компоненты (СПНК) системы управления документами, который выступает в роли базового программного обеспечения (ПО) и позволяет наращивать функциональность системы. Основное назначение СПНК – обработка запросов от уровня представления. В работе описываются устройство, принцип работы и функциональные возможности СПНК. Выделяются основные компоненты СПНК, рассматривается протокол взаимодействия между ПО конечного пользователя и СПНК, описывается формат передаваемых сообщений.

Ключевые слова: система управления документами, сервер приложений, связующее ПО.

Введение

Архитектура программной системы определяет ее структуру, точнее – несколько структур, каждая из которых включает в себя элементы и взаимосвязи между ними [1]. Элементы могут быть вычислительными объектами, связанными потоком управления или бизнес-объектами, связанными семантическими ограничениями [2]. Системы управления документами (СУД), как правило, реализуются с использованием архитектуры «клиент-сервер». «Клиент-сервер» – архитектура распределенной вычислительной системы, в которой приложение делится на клиентский и серверный процессы [3]. Вариантом архитектуры «клиент-сервер» является введение сервера приложений в качестве ПО промежуточного слоя. В данном случае клиент выполняет только функции визуализации и ввода данных, а всю прикладную логику реализует сервер приложений. Обмен между клиентом и сервером в таких системах осуществляется на уровне команд вывода данных на экран и результатов пользовательского ввода. Чаще всего в модели сервера приложений компоненты прикладной логики и управления данными реализуются раздельно. Архитектуру «клиент-сервер» с сервером приложений часто называют «тонким» клиентом, в отличие от традиционного «толстого» клиента, реализуемого в архитектуре сервера базы данных (БД). «Тонкий» клиент является вариантом, который может быть использован, когда ресурсов, доступных на рабочих местах пользователей, недостаточно для исполнения логики приложения. Кроме того, эта технология позволяет сократить расходы на эксплуатацию клиентских компонентов системы за счет их сильного упрощения.

Система управления документами (СУД) − компьютерная система (или набор компьютерных программ), используемая для отслеживания и хранения электронных документов и/или образов (изображений и иных артефактов) бумажных документов.

В общем случае системы управления документами предоставляют хранение, версионирование, пометку метаданными и безопасность по отношению к документам, а также индексирование и развитые возможности поиска документа.

В СУД, как и в любых других информационных системах, можно выделить проблемно-независимую составляющую, которая является каркасом приложения и обеспечивает реализацию основных функциональных возможностей без привязки к конкретной предметной области. В настоящей работе предлагается модель организации СУД с выделением сервера проблемно-

независимой компоненты (СПНК) в отдельное специализированное приложение – одну из разновидностей ПО промежуточного слоя. СПНК предназначен для решения задачи обработки запросов от уровня представления (например, запросов на редактирование или просмотр данных).

Структура СПНК

Описываемый СПНК состоит из двух компонентов и имеет распределенную модель работы. Имеется клиентская часть, которая располагается на стороне веб-сервера, принимает HTTP-запросы от конечного пользователя и преобразует их во внутренние текстовые запросы к серверной части. Серверная часть состоит из различных модулей, каждый из которых обеспечивает выполнение определенного набора операций. Клиентская и серверная части СПНК взаимодействуют с помощью интерфейса TCP/IP и могут размещаться как на одной, так и на разных машинах (см. рис.). Клиентская часть узнает о местоположении серверной из собственного конфигурационного файла, в котором указываются адрес и порт, на которые необходимо отсылать сообщения. При выходе из строя машины, содержащей серверную логику, клиентская часть может быть легко перенастроена изменением конфигурационного файла. Также отметим, что несколько экземпляров СПНК (клиентская и серверная части) могут бесконфликтно функционировать на одной ЭВМ. Это удобно для ситуации, когда имеется несколько релизов (версий) СПНК.

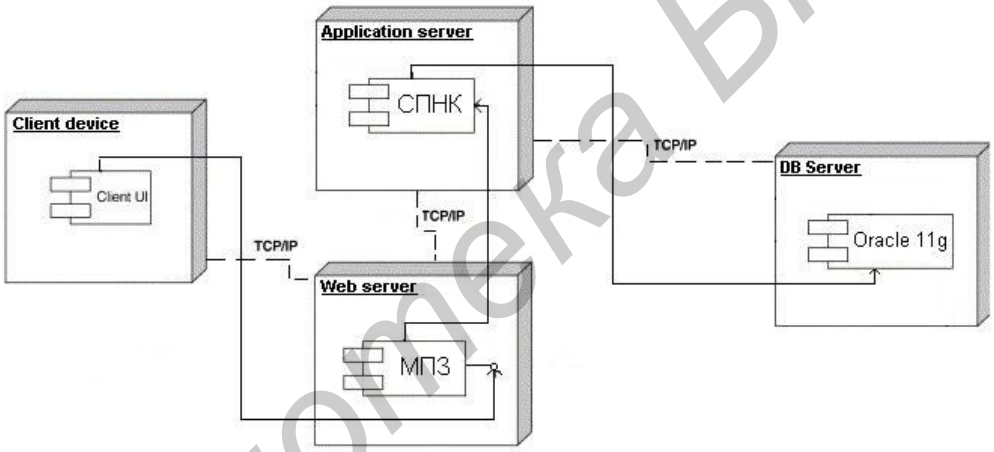

Диаграмма развертывания СПНК

После выполнения запрашиваемой операции серверная часть отсылает ответ клиентской части, которая возвращает его конечному пользователю по HTTP. В качестве клиентской части выступает модуль переадресации запросов (МПЗ), который устанавливается в виде модуля в веб-сервер (в данной работе IBM_HTTP_Server/6.1 Apache/2.0.47). МПЗ реализован на языке программирования C++. Известно, что Apache обрабатывает поступивший запрос примерно в 11 этапов [4]. Каждому этапу соответствует определенный обработчик (модуль) Apache. Событие получения каждого HTTP-запроса перехватывается на стороне веб-сервера, и управление передается встроенному МПЗ посредством функции обратного вызова.

Строка запроса анализируется. Если запрашивается статический ресурс (например, Javascript модуль или изображение), то никаких дополнительных действий не происходит, и дальнейшая обработка осуществляется стандартными средствами веб-сервера. Если запрашиваемый URI содержит зарезервированное ключевое слово, это означает, что запрос должен быть обработан соответствующим модулем серверной части СПНК.

Клиентская часть готовит текстовое сообщение, содержащее информацию о полученном запросе, а также идентификатор модуля серверной части, который будет обрабатывать сообщение. Если в процессе обработки запроса возникнет ошибка, МПЗ сохранит ее описание в журнале регистрации, а также вернет пользователю XML-сообщение с характеристикой ошибки.

Структура серверных компонентов

Перейдем к описанию функциональности серверных компонентов. Серверная часть состоит из основного исполняемого файла - менеджера сервисов (Service Manager, SM) и набора динамически подключаемых модулей. SM содержит интерфейс для обращения к базе данных, который позволяет вызывать хранимые процедуры и передавать на выполнение SQL код. SM хранит очередь сообщений, которые поступают от клиентской части. Очередь сообщений просматривается в отдельном потоке, сообщения обрабатываются с учетом порядка размещения и приоритета.

Все модули имеют унифицированный интерфейс, посредством которого внешние объекты могут подключиться к модулю. Тем самым гарантируется, что модули со схожими спецификациями являются взаимозаменяемыми и допускают возможность их независимой модернизации. Также модули можно объединить друг с другом, формируя более крупные компоненты. Благодаря применению четких интерфейсов становится возможным реализовать на практике преимущества повторного использования компонентов – повышение производительности труда при разработке, простоту применения, единообразие структуры приложения. В качестве примера приведем две основные операции: возможность произвести инициализацию внутренних переменных из конфигурационного файла, а также возможность принять текстовый запрос для обработки. Модули способны выполнить два типа запросов: локальные – для выполнения которых достаточно функциональности самого модуля (например, математические преобразования), и расширенные – требующие обращения к ресурсам менеджера сервисов (например, обращение к базе данных). Это позволяет локализовать интерфейсы основных операций непосредственно в SM, дав возможность модулям обращаться к ним по мере надобности.

Модули подключаются SM посредством чтения конфигурационного файла, в котором для описания модулей отводится отдельная секция. Для каждого модуля указывается имя объектного файла, имя инициализационного файла, имя журнала регистрации и уровень детализации событий. Приведем пример описания секции подключения модулей.

```
[Modules]
ModuleNames=v1_web_app,WebMenuModule,RequestProcessor
ModuleLibs=libWebRequests.so,libWebMenuModule.so,libRequestProcessor.so
ModuleInis=WebRequests.ini,WebMenuModule.ini,RequestProcessor_Web.ini
```

Очевидно, что модули могут создаваться и модифицироваться без изменения SM. Для того, чтобы подключить новый модуль, необходимо всего лишь изменить файл конфигурации SM и перезапустить SM.

Рассмотрим работу СПНК на конкретном примере. Работа пользователя в системе начинается с прохождения процедуры аутентификации. Так как клиент является «тонким», то в качестве клиентского ПО выступает только веб-браузер. Перед тем как предложить пользователю ввести имя учетной записи и пароль, клиентская HTML-страница отсылает СПНК запрос инициализации соединения. Основная задача данного запроса заключается в создании сессии и получении открытого ключа шифрования сервера. Получение и диспетчеризация запросов на стороне СПНК осуществляется менеджером веб-сессий (МВС). Задачей МВС является проверка того, что доступ к функциям СПНК предоставляется легальным пользователям. Запросы от нежелательных пользователей отфильтровываются. МВС хранит коллекцию активных веб-сессий. При поступлении нового запроса проверяется, имеет ли он активный идентификатор сессии. Если идентификатор сессии имеется, то проверяются cookie. Если сессия устарела, пользователь перенаправляется на страницу повторной аутентификации. Коллекция активных сессий просматривается через определенный интервал времени, и сессии, последнее время обращения к которым превысило заданный пороговый промежуток времени, считаются устаревшими и удаляются.

При получении запроса на подготовку установления соединения, МВС создает новую сессию и присваивает ей уникальный идентификатор. В качестве параметров cookie на компьютер пользователя передаются открытый ключ шифрования и идентификатор сессии. Затем клиентская страница предлагает пользователю ввести имя учетной записи и пароль. Перед отправкой пароль предварительно шифруется на стороне клиента при помощи алгоритма RSA [5] с использованием открытого ключа, полученного на предыдущем шаге. СПНК получает за-

шифрованный пароль, расшифровывает его своим открытым ключом и производит проверку легальности пользователя сравнением пароля со значением из базы данных.

Большинство пользователей информационных средств и систем используют компьютеры для доступа к ряду сервисов, к которым машина пользователя присоединяется через сеть. Поэтому стандартной является схема однократного входа SSO (Single Sign on). Описываемый СПНК поддерживает схему Web SSO, основанную на использовании cookie, при реализации процедуры входа выполняются следующие шаги:

1. Пользователь передает на Web-сервер (например SiteMinder) имя пользователя и пароль.

2. Агент Web-сервера извлекает мандат пользователя из сервера мандатов (например LDAP).

3. Агент Web-сервера сохраняет зашифрованный мандат в качестве cookie на компьютере пользователя.

4. Когда пользователь обращается к СПНК, осуществляется чтение мандата пользователя из его cookie.

Рассмотрим принцип вызова хранимых процедур сервера БД. Протокол взаимодействия клиентов с СПНК основан на пересылке XML-сообщений, описывающих запрашиваемые операции. Клиентский процесс (в этой роли может выступать HTML-страница или приложение, запущенное конечным пользователем) отсылает СПНК по протоколу HTTPS сообщение следующего формата:

```
<DB_REQUEST>
        <TYPE>DB_REQUEST_TYPE</TYPE>
        <SUBTYPE>DB_REQUEST_SUBTYPE</SUBTYPE>
        <INPUT_PARS>
        <PAR><NAME>PAR_NAME1</NAME><VAL>VALUE1</VAL></PAR>
        ...
        <PAR><NAME>PAR_NAMEN</NAME><VAL>VALUEN</VAL></PAR>
        </INPUT_PARS>
</DB_REQUEST>
```

Тип запроса определяет компонент системы, к которому относится запрос. Комбинация типа и подтипа запроса уникально определяют хранимую процедуру, которую необходимо вызвать на сервере БД. Отметим, что тип и подтип запроса описываются в терминах бизнес-логики. Поэтому если злоумышленник перехватит и расшифрует сообщение, то он не сможет определить настоящее имя хранимой процедуры.

Для определения полного имени процедуры СПНК конфигурируется файлом описаний запросов, содержащим записи вида:

```
<DBREQUEST>
        <TYPE>DB_REQUEST_TYPE</TYPE>
        <SUBTYPE>DB_REQUEST_SUBTYPE</SUBTYPE>
        <ST_PROC>SCHEMA.PACKAGE.SP_NAME</ST_PROC>
        <INPUT_PARS>
                <PAR NAME="PAR_NAME1" DATATYPE="S"/>
                ...
                <PAR NAME="PAR_NAMEN" DATATYPE="I"/>
        </INPUT_PARS>
        <RESULT_PARS>
                <PAR NAME="p_out_cursor" DATATYPE="C" />
        </RESULT_PARS>
</DBREQUEST>
```

Как видно из приведенного выше листинга, формат полного описания запроса хранит конечное имя процедуры, а также спецификацию типов входных и выходных параметров. Заметим, что СПНК может обработать только те запросы, для которых имеется полное описание. Т.е. даже получив доступ к СПНК, злоумышленник не сможет вызвать процедуру БД, которая не входит в набор разрешенных для вызова через СПНК процедур.

Запросы могут обрабатываться в двух режимах – командном и диалоговом. Командный режим используется для запуска различных служебных операций. В диалоговом режиме помимо статуса выполнения запроса клиенту возвращается XML-документ, содержащий ответ. Приведем пример формата ответа:

```
<RESULT><RESULT_PARS/><RESULT>
<FIELDNAMES COLS="4">
<NAME name="ParName1" displayName="Parameter #1" type="STRING" precision="0" display_width="300">ParName1</NAME>
...
<NAME name="ParName4" displayName="Parameter #4" type="STRING" precision="0" display_width="200" >ParName4</NAME>
</FIELDNAMES>
<RESULTSET ROWS="1">
<ROW>
<PAR><NAME>ParName1</NAME><VAL>Value1</VAL></PAR>
...
<PAR><NAME>ParName4</NAME><VAL>Value4</VAL></PAR>
</ROW>
</RESULTSET>
</RESULT></RESULT>
```

Ответ представляет из себя таблицу N×M, где *N* – значение атрибута ROWS тега RESULTSET; *M* – значение атрибута COLS тега FIELDNAMES. В блоке FIELDNAMES хранятся атрибуты визуализации всех столбцов таблицы (имя для отображения, ширина, тип данных и др.).

После обработки каждого запроса система сохраняет в журнале регистрации данные производительности БД (время выполнения, предыдущее время выполнения, минимальное, максимальное, среднее время выполнения). Анализ этих данных позволит выявить процедуры, производительность которых подлежит оптимизации.

Для регистрации событий, которые могут представлять угрозу для безопасности СПНК, используется процедура аудита. Если фиксировать в журнале аудита все события, объем регистрационной информации будет расти слишком быстро, что затруднит ее эффективный анализ. Поэтому процедура аудита может конфигурироваться на мониторинг определенных событий или параметров запросов.

Помимо процедуры аудита имеется возможность выводить информацию о ходе работы приложения в журнал регистрации. Каждый модуль имеет свой лог-файл. Также для каждого модуля имеется возможность задать степень детализации выводимых сообщений.

При разработке СПНК с целью повышения производительности труда в качестве сторонних библиотек были использованы: xerces для операций разбора и обработки XML документов при помощи интерактивного парсера SAX, sqlapi для выполнения операций работы с БД, xalan для преобразования XML документов в другие форматы при помощи XSLT, openssl для обеспечения безопасности информационного обмена.

Заключение

Практическая значимость работы заключается в том, что описанный СПНК может найти применение не только в СУД, но и в различных ИС в качестве связующего ПО, являющегося посредником запросов к объектам.

Кроме того, следует отметить возможность быстрой модификации самих алгоритмов и повторного использования компонентов СПНК за счет модульной структуры системы. Использование рассмотренного в статье принципа построения СПНК позволит снизить издержки в процессе создания и повысить качество функционирования СУД.

DOMAIN INDEPENDENT SERVER FOR DOCUMENT MANAGEMENT SYSTEM

M.V. STERJANOV, I.V. BAIDAKOV

Abstract

Domain independent server for document management system is detailed. Server plays a role of basic software which allows to extend functionality of the system. Main objective of suggested server is processing requests from representation layer. Server's architecture and functionality are also reviewed.

Список литературы

1. Recommended Practice for Architectural Description of Software-Intensive Systems, 2000.
2. *Русаков М.А.* Многоэтапный анализ архитектурной надежности в сложных информационно-управляющих системах: дисс. канд. техн. наук: 05.13.01. Красноярск, 2005.
3. *Басс Л, Клементс П., Кацман Р.* Архитектура программного обеспечения на практике. Питер, 2006.
4. *Stein L., MacEachern D.*Writing Apache Modules with Perl and C. Inc., 1999.
5. *Rivest R.L., Shamir A., Adleman L.* // Communications of the ACM. 1978. Т. 21, №2. С. 120–126.