При использовании материалов статьи необходимо использовать данную ссылку:
Щекочихин О.В., Черкасова Н.В. Анализ шаблонов проектирования информационных систем клиент-серверной архитектуры // Информационно-экономические аспекты стандартизации и технического регулирования. 2019. № 5. (51). С. 26-29.

УДК 654.05

АНАЛИЗ ШАБЛОНОВ ПРОЕКТИРОВАНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ КЛИЕНТ-СЕРВЕРНОЙ АРХИТЕКТУРЫ

Щекочихин О.В., Черкасова Н.В.

Рассмотрены варианты шаблонов клиент-серверной архитектуры, которые используются при проектировании информационных систем. Для каждого шаблона описаны подходы к их реализации с позиции критериев обеспечения масштабируемости информационной системы, доступности и целостности данных.

Ключевые слова: шаблон архитектуры, архитектурная модель, информационная система, клиент-серверная архитектура.

Для обеспечения эффективной работы информационных систем при их проектировании применяются ряд архитектурных шаблонов. Отсутствие формализации обоснования архитектурных решений может привести к разработке системы не отвечающей требованиям масштабируемости гибкости, расширяемости функционала при возникновении новых задач и т.п.

В настоящей статье рассмотрены существующие модели проектирования информационных систем на основе клиент-серверной архитектуры, их достоинства и недостатки.

Архитектурная модель (или шаблон) – это набор правил, формирующих ИС как крупно-модульную структуру, который базируется на принципе разбиения и повторного использования типовых конструкций. Модель проектирования для ИС определяет методы и средства построения программных компонентов, а также способы их взаимодействия между собой.

Ранее главенствовал принцип монолитности архитектуры информационной системы. У монолитных ИС отмечается несколько достоинств, которые могут стать их серьёзными недостатками:

- Масштабируемость. При увеличении скорости и производительности таких ИС, может потребоваться разветвление функционала на более современное и быстрое оборудование, а это бывает очень трудно спрогнозировать на начальном этапе планирования будущей системы.

- Надёжность и доступность. При возникновении каких-либо ошибок в приложениях ИС, она может перейти в режим перезапуска служб, а значит быть в это время недоступной.

- Приспосабливаемость. Объём ИС увеличивается при постоянном добавлении всё новых функций в процессе эксплуатации, тогда как новые версии выпускаются редко, поэтому работать с ними становится всё сложнее. Чтобы переработать их для подключения к новым, усовершенствованным сервисам, потребуются дополнительные ресурсы и финансовые затраты.

В качестве строительных блоков для всех видов ИС целесообразно использовать объекты, формирующие структуру и поведение ИС. Объекты собирают различные свойства и задачи, взаимодействуют друг с другом посредством предопределенных механизмов и интерфейсов.

Современные подходы к проектированию ИС отказались от принципа монолитности в пользу систем с открытой архитектурой [1-4].

Щекочихин Олег Владимирович, кандидат технических наук, доцент, инженер информационной безопасности ООО «ММТР технологии», г. Кострома
Черкасова Наталья Владимировна, аспирант, ФГБУН ВИНИТИ РАН, г. Москва

Далее рассматриваются их наиболее популярные варианты.

Объектно-ориентированная архитектура (OOA, Object-oriented architecture) стала популярна среди разработчиков информационных систем именно по причинам наглядности, чёткости и логичности.

Архитектура на основе механизма компонентов (CBA, Component-based assembly architecture) подразумевает разбиение крупных приложений на множество более мелких компонентов. Каждый из них предоставляет свой чёткий интерфейс для работы с другими компонентами. Компоненты являются многоразовыми, независимыми, взаимозаменяемыми, расширяемыми и не привязываются к определённой структуре. Для связи и управления зависимостями могут использоваться другие шаблоны проектирования.

ОБЪЕКТНО-ОРИЕНТИРОВАННАЯ АРХИТЕКТУРА (OOA, OBJECT-ORIENTED ARCHITECTURE) СТАЛА ПОПУЛЯРНА СРЕДИ РАЗРАБОТЧИКОВ ИНФОРМАЦИОННЫХ СИСТЕМ ИМЕННО ПО ПРИЧИНАМ НАГЛЯДНОСТИ, ЧЁТКОСТИ И ЛОГИЧНОСТИ.

Аспектно-ориентированное программирование (AOP, Aspect-oriented programming) включает в себя методы и инструменты программирования, повышающие модульность задач на уровне исходного кода. Данный стиль позволяет разделить бизнес-логику ИС на значения или связанные области функциональности. В результате, сложность исходного кода снижается, а эффективность кодирования возрастает. Аспекты представлены, как сквозные проблемы и вводятся по мере необходимости. По всем правилам программирования, в разработке поддерживается некоторый уровень группировки и инкапсуляции значений в независимые сущности путем создания функций, процедур, модулей, классов, методов и т. д. Они могут быть использованы для реализации или составления различных задач. Однако некоторые задачи идут вразрез многим функциональным инструкциям, заданным в программе. Такие задачи называются межсекторальными или горизонтальными.

Агентно-ориентированная программная инженерия (AOSE, Agent-oriented software engineering) - это архитектурное решение сосредоточено на концепции программных агентов. В отличие от OOA, в основе которого лежат объекты, AOSE имеет внешне определенных агентов с интерфейсами и возможностями обмена сообщениями в своей основе. Обмениваемые сообщения интерпретируются получающими агентами способом, специфичным для его конкретного класса агентов. Программный агент – это самостоятельная компьютерная программа, работающая в рамках своей среды, без постоянного контроля, выполняя конкретные функции для конечного пользователя или для другой программы.

К объектно-ориентированному подходу также относится и *архитектура доменного проектирования (DDD, Domain-driven design architecture)*. В данном случае домен представляет собой совокупность понятий: бизнес-область, ее элементы и поведение, а также отношения между ними. Архитекторы должны хорошо разбираться в бизнес-сфере для моделирования и интенсивно взаимодействовать с экспертами предметной области бизнеса, чтобы смоделировать систему с безупречным функционалом. Процесс проектирования архитектуры домена преследует цель также улучшить и усовершенствовать язык домена. Данный подход актуален, когда речь идёт о сложной области, и необходимо улучшить общение и понимание в команде разработчиков. Или же он может быть идеальным вариантом, при наличии больших и сложных сценариев корпоративных данных, которыми сложно управлять с помощью существующих технических приёмов.

Клиент-серверная архитектура (Client/server architecture) разделяет систему на два основных приложения, в которых клиент отправляет запросы на сервер, получая от него ответы. В основном, сервер представляет собой базу данных с логикой приложения, представленной в виде хранимых процедур. Эта модель помогает проектировать информационные системы, которые включают клиентскую часть, серверную и связующую сеть. Простейшая форма клиент-серверной архитектуры, или двухуровневая архитектура, представляет собой серверное приложение, к которому обращаются сразу несколько клиентов. Веб-серверы и серверы приложений могут выполнять роль сервера для получения клиентских запросов, их обработки и отправки ответов обратно клиентам.

Подмножеством клиент-серверной архитектуры может выступать *модель приложений одноранговой сети (peer-to-peer)*,

которая позволяет клиенту и серверу поменяться ролями, чтобы распределять и синхронизировать файлы и информацию между несколькими клиентскими машинами. Данная модель расширяет стиль клиент-сервер за счет множественных ответов на запросы, общих данных, а также устойчивости к удалению участников.

Среди основных преимуществ модели клиент-серверной архитектуры можно выделить: безопасность, централизованный доступ к данным, простота обслуживания. Поскольку все данные хранятся в одном месте, а именно на сервере, их легче обслуживать, администрировать, обновлять, контролировать и обеспечивать такой элементы безопасности как конфиденциальность и целостность информации. Серверное приложение и база данных могут быть запущены на одном компьютере, а могут быть реплицированы на несколько, чтобы обеспечить легкую масштабируемость и высокую доступность. В этом случае принято считать объединение всех машин с серверной ролью в кластер. Кроме того, серверное приложение корпоративного уровня может состоять из нескольких подсистем, и каждая может быть запущена на отдельном сервере в общем кластере.

У двухуровневой модели архитектуры клиент-сервер также существуют и недостатки: недостаточные расширяемость, гибкость, масштабируемость, надёжность. Это связано с тем, что при отказе сервера, доступ к данным блокируется, а попытки преобразования системы могут оказаться не бесконечными из-за ограничений вместимости сервера. Таким образом, клиент-серверная архитектура преобразовалась в более общую трехуровневую (или N-уровневую) архитектуру.

Компоненты *многоуровневой распределенной вычислительной архитектуры приложения* могут быть развернуты на нескольких машинах. Также они могут быть интегрированы с помощью сообщений или удаленных вызовов процедур, методов, общей архитектуры брокера объектных запросов (CORBA), корпоративных компонентов Java и т.д. Распределенное размещение служб приложений обеспечивает высокую доступность, масштабируемость, управляемость. Эта архитектурная модель используется при развёртывании веб, облачных, мобильных и других клиенто-ориентированных приложений.

Многоуровневая (многоярусная) архитектура (Multi-tier distributed computing architecture) – это улучшенная версия модели клиент-сервер и наиболее часто используемый архитектурный шаблон. В основном, корпоративная ИС, как уже упоминалось ранее, включает в себя три или более уровней: пользовательского интерфейса, бизнес-логики и хранения данных. В эту архитектуру, по мере необходимости, могут быть легко добавлены также и дополнительные уровни для возможности интеграции со сторонними приложениями. Обычно это модули систем управления базами данных. Средний уровень содержит приложение и веб-сервер, а уровень представления - это приложения с пользовательским интерфейсом, которые ещё называют «толстые клиенты», или веб-браузеры, соответственно, «тонкие клиенты», а также мобильные браузеры.

ГЛАВНЫМ ПРЕИМУЩЕСТВОМ МНОГОУРОВНЕВОЙ АРХИТЕКТУРЫ ЯВЛЯЕТСЯ РАЗДЕЛЕНИЕ ЗАДАЧ. ЭТО ЗНАЧИТ, ЧТО КАЖДЫЙ СЛОЙ МОЖЕТ ВЫПОЛНЯТЬ ТОЛЬКО СВОЮ ЗАДАЧУ. МНОГОСЛОЙНАЯ МОДЕЛЬ АРХИТЕКТУРЫ ПОЗВОЛЯЕТ ПРИЛОЖЕНИЮ БЫТЬ РЕМОНТОПРИГОДНЫМ, ЛЕГКО ТЕСТИРУЕМЫМ, ПРОСТЫМ ДЛЯ НАЗНАЧЕНИЯ КОНКРЕТНЫХ И ОТДЕЛЬНЫХ РОЛЕЙ, А ТАКЖЕ ДЛЯ СОВЕРШЕНСТВОВАНИЯ СЛОЁВ ПО-ОТДЕЛЬНОСТИ.

Главным преимуществом многоуровневой архитектуры является разделение задач. Это значит, что каждый слой может выполнять только свою задачу. Многослойная модель архитектуры позволяет приложению быть ремонтопригодным, легко тестируемым, простым для назначения конкретных и отдельных ролей, а также для совершенствования слоёв по-отдельности.

С помощью этой модели можно быстро и без риска разрабатывать вэб-приложения, облачные и также промышленного уровня. Приложения могут быть изменены на любом из уровней. Компоненты могут меняться местами в ИС или заменяться другими компонентами. Данная модель оказывается очень удобной при внедрении новых технологий или изменении бизнес-логики для ранее разработанных систем, так как новые разработки могут служить как раз теми компонентами, которые легко интегрируются в уже функционирующую систему.

Существуют также, открытые слои, которые можно обойти во время определённых запросов, то есть возможно переходить непосредственно к слою ниже.

Метод запроса и ответа является основным для взаимодействия между клиентами и серверами в соответствии с архитектурным шаблоном клиент-сервер. Однако, сообщение является синхронным. Для этого клиенты и серверы должны быть доступны в сети, чтобы инициировать и выполнить функции ИС. Когда запросы на обслуживание обрабатываются и выполняются серверными машинами, запрашивающие услуги клиенты должны ждать получения ответа от серверов и не могут выполнять какую-либо другую работу. То есть когда происходит событие (приходит запрос), приложения должны получать информацию о нём и немедленно приступать к его выполнению. В случае, когда связь становится асинхронной нет необходимости, чтобы участвующие приложения были постоянно доступны онлайн.

Шаблоны клиент-серверной архитектуры не позволяют в современных условиях удовлетворять требованиям по обеспечению масштабируемости ИС, доступности и целостность данных.

Помимо этого, есть несколько прикладных и предметно-ориентированных архитектур, которые формируются, проверяются и утверждаются исследователями во всем мире и представляются в качестве исследовательских материалов. Таким образом, область архитектурных шаблонов постоянно растет, чтобы поддерживать область разработки ИС. iea

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ:

1. Vijini Mallawaarachchi. 10 Common Software Architectural Patterns in a nutshell / Towards data science, статья от 04.09.2017, [Электронный ресурс] URL: https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013;
2. Pethuru Raj. Architectural Patterns. / Pethuru Raj, Anupama Raman, Harihara Subramanian. – Birmingham, United Kingdom of the Great Britain: Packt Publishing, 2017 – 437 с.
3. Izza, S. Integration of industrial information systems: From syntactic to semantic integration approaches // Int. J. Enterp. Inf. Syst, 2009, 3 (1), pp. 1-58.
4. Mitchell, V. Knowledge integration and information technology project performance // MIS Q, 2006, 30 (4), pp. 919-939.

ANALYSIS OF TEMPLATES OF DESIGNING INFORMATION SYSTEMS OF CLIENT-SERVER ARCHITECTURE

Shchekochikhin Oleg V., Candidate of Technical Sciences, Associate Professor, Information Security Engineer of MMTR Technologies LLC, Kostroma
Cherkasova Natal'ya V., graduate student, FSBI VINITI RAS, Moscow

Variants of client-server architecture patterns that are used in the design of information systems are considered. For each template, approaches to their implementation are described from the perspective of criteria for ensuring the scalability of the information system, the availability and integrity of data.

Keywords: architecture pattern, architectural model, information system, client-server architecture.

REFERENCES:

1. Vijini Mallawaarachchi. 10 Common Software Architectural Patterns in a nutshell. Towards data science. Available at: URL: https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013;
2. Pethuru Raj. Architectural Patterns. Pethuru Raj, Anupama Raman, Harihara Subramanian. – Birmingham, United Kingdom of the Great Britain: Packt Publishing, 2017 – 437 p.
3. Izza, S. Integration of industrial information systems: From syntactic to semantic integration approaches // Int. J. Enterp. Inf. Syst, 2009, 3 (1), pp. 1-58.
4. Mitchell, V. Knowledge integration and information technology project performance // MIS Q, 2006, 30 (4), pp. 919-939.