**АРХИТЕКТУРНЫЕ СТИЛИ**

*Понятие архитектурного стиля.* Архитектура может соответствовать некоторому архитектурному стилю. Большинство архитектур построены на основе систем, использующих похожие решения. Сходство может быть определено как архитектурный стиль, который, в свою очередь, можно рассматривать как особый вид паттерна (шаблона).

Архитектурный стиль представляет собой кодификацию опыта проектирования ИТ-систем. Примеры архитектурных стилей включают:

* распределенный стиль,
* стиль «каналы и фильтры»,
* стиль с централизованной обработкой данных,
* стиль, построенный на правилах, и т.д.

Архитектурный стиль можно определить как семейство систем в терминах шаблона организации структуры. Архитектурный стиль определяет номенклатуру компонентов и типов соединительных звеньев, а также набор условий, в соответствии с которыми они могут соединяться.

Архитектурный стиль определяется набором типов компонентов, во время счета выполняющих некоторую функцию, топологической раскладкой компонентов с указанием их взаимосвязей во время выполнения, набором семантических ограничений, набором соединителей, служащих средой сообщения, координации и сотрудничества между компонентами.

Иногда вместо термина архитектурный стиль используется термин архитектурный паттерн (шаблон). Однако следует отметить наличие принципиальных различий между архитектурными стилями и паттернами. Паттерн – это, по существу, фрагмент кода на конкретном языке программирования, а архитектурный стиль – это подход к проектированию.

Несмотря на многочисленные попытки, до сих пор отсутствуют стандартные языки описания архитектур. Принято выделять двенадцать базовых архитектурных стилей, которые делятся на пять групп:

* потоки данных (Data Flow Systems);
* вызов с возвратом (Call-and-Return Systems);
* независимые компоненты (Independent Component Systems);
* централизованные данные (Data-Centric Systems);
* виртуальные машины (Virtual Machines).

К системам, работающим по принципу потоков данных, относят системы двух архитектурных стилей:

* системы пакетно-последовательной обработки (Batch Sequential Systems) и системы типа конвейеры,
* фильтры (Pipe and Filter Architecture).

К вызову с возвратом относят четыре архитектурных стиля:

* программа-сопрограммы (Main Program and Subroutines),
* объектноориентированные системы (Object-Oriented Systems),
* клиент-серверные системы (Client-Server Systems),
* иерархические многоуровневые системы (Hierarchically Layered Systems).

К системам, работающим по принципу независимых компонентов, относят системы двух стилей

* взаимодействующих процессов (Communicating Sequential Processes) и системы,
* управляемые событиями (Event-Based Systems).

К системам, использующим централизованные репозитория данных, относят:

* системы, основанные на использовании централизованной базы данных (Database Systems),
* системы, использующие принцип классной доски (Blackboard Systems).

К системам, работающим по принципу виртуальной машины, относят:

* интерпретаторы (Interpreters),
* системы, основанные на правилах (RuleBased Systems).

*Потоки данных, вызов с возвратом*. К системам, работающим по принципу потоков данных, относят:

* системы пакетно-последовательной обработки,
* системы типа конвейеры и фильтры.

Системы пакетно-последовательной обработки представляют собой набор связанных программных модулей, образующих линейную структуру. Задача делится на отдельные подзадачи. При этом выходные данные, сформированные одной подзадачей, используются в качестве входных данных для другой подзадачи. Обычно подзадачи выполняются последовательно. Данные могут передаваться либо через системную память, либо через внешние файлы.

Для управления вычислительным процессом обычно используются скриптовые языки. Типичным примером такого подхода являются программы, написанные на языке Unix shell. Архитектурный стиль «конвейеры и фильтры» близок к системам пакетно-последовательной обработки и может рассматриваться как обобщение пакетно-последовательной обработки.

Система, построенная с использованием стиля «конвейеры и фильтры», представляет собой множество модулей, каждому из которых ставится в соответствие один или несколько процессов. Модули могут быть как одинаковыми, так и разными и могут выполняться как на одном, так и на различных хостах. Данные с выходов одного модуля могут поступать на входы одного или нескольких других модулей.

Система работает по принципу конвейера. Данные между отдельными ступенями конвейера могут передаваться разными способами, в частности, посредством использования механизмов межпроцессорного взаимодействиями, такими как PIPE в Unix. Обработка носит преимущественно линейный характер, хотя конвейеры могут иметь обратные связи.

Примером данного подхода может служить компилятор. На вход компилятора поступает исходный код компилируемой программы. Функции первого фильтра выполняет лексический анализатор. В качестве второй ступени выступает семантический анализатор, качестве третьей ступени – оптимизатор, в качестве четвертой – генератор кода. Данный стиль широко используется при построении систем обработки сигналов и изображений.

К системам, работающим по принципу вызова с возвратом, обычно относят:

* системы, построенные по принципу программа-со-программы,
* клиент-серверные системы,
* объектно-ориентированные системы,
* иерархические многоуровневые системы.

Отличительной особенностью систем, относящихся к данной группе, является то, что это синхронные программные архитектуры, в которых клиентская программа приостанавливает свое функционирование на то время, пока поставщик сервиса обслуживает запрос. По завершении обработки запроса сервер возвращает результат вычислений клиенту.

Архитектуры, работающие по принципу вызова с возвратом, могут иметь произвольное число уровней вложенности. Самым старым архитектурным стилем, относящимся к данной группе, является архитектурный стиль типа программа-сопрограммы. Программа выполняет функции контроллера, который управляет вычислительным процессом, в то время как функциональность реализуется в сопрограммах. Сопрограммы могут выполняться как на локальном, так и на удаленном хосте. В последнем случае речь идет о вызове удаленных процедур.

Отличительной особенностью данного стиля является то, что программа имеет только одну нить управления. Данный стиль является, по существу, реализацией идей структурного программирования.

В качестве разновидности стиля программа-сопрограммы можно выделить архитектуры типа ведущий-ведомый (Master-Slave Architecture). Обычно такие архитектуры не выделяют в отдельный стиль и рассматривают как параллельную версию стиля программа-сопрограммы. Отличительной особенностью данных архитектур является то, что основная программа и сопрограммы работают одновременно (параллельно). В данном случае на основную программу (ведущий) возлагаются функции диспетчеризации процесса вычислений. Сопрограмма (ведомый) получает задание, выполняет его, а по завершению задания запрашивает ведомого о новом задании. Данная архитектура может реализовываться как в рамках многопроцессорных систем, так и в сетевой среде с произвольной топологией.

Клиент-серверные системы можно рассматривать как специальный случай стиля программа-сопрограммы. Основное различие состоит в том, что клиент и сервер находятся на разных хостах, хотя в принципе клиент и сервер могут работать на одном хосте.

Клиент – это процесс, который формирует запрос на обслуживание. Сервер – это процесс, который реализует сервис. В простейшем случае клиент посылает серверу команды и ожидает окончания выполнения запроса. Результатом выполнения запроса могут быть либо данные, либо подтверждение выполнения команды.

Большинство серверов работает с множеством клиентов. Широко используемая разновидность клиент-серверных систем – транзакционные системы, к которым могут быть отнесены, например, системы продажи билетов.

В системах, ориентированных на выполнение транзакций, число и типы транзакций фиксированы. Серверы в клиент-серверных системах при увеличении числа запросов могут масштабироваться. Принято выделять два типа клиент-серверных систем: с толстым и тонким клиентом.

Толстым называют клиентское приложение, которое содержит наряду с кодом, отвечающим за представление данных, достаточно большой объем кода, реализующего бизнес-логику. Тонкий клиент содержит код, который реализует исключительно функции, связанные с представлением информации. Обычно тонкий клиент – это Web-браузер.

Каждый из этих подходов имеет собственные достоинства и недостатки. Например, при использовании тонкого клиента достаточно просто модифицировать код приложения, поскольку нет необходимости обновлять код на многочисленных клиентских хостах. Код сервера обычно организуется по принципу супервизор-рабочие процессы. Супервизор обслуживает единую точку доступа к сервисам. Рабочие процессы обрабатывают каждый конкретный запрос. Обычно реализуется следующий алгоритм: сервер открывает «хорошо известный порт» – принимает запросы к открытому порту – при поступлении запроса передает его одному из рабочих процессов и ожидает следующего запроса.

Клиент-серверные архитектуры имеют несколько разновидностей. Большинство ранних версий клиент-серверных приложений имели двухслойную организацию, в которой на сервере размещался репозиторий данных. На сервер возлагаются функции, связанные с управлением данными, а на клиентской стороне реализуется логика приложения (бизнес-логика) и логика представления.

Трехслойная архитектура отличается от двухслойной тем, что бизнес-логика размещается на отдельном сервере. Процессы становятся более устойчивыми, поскольку работают независимо от клиентов и от серверов.

Иногда используется и четырехслойная архитектура, которая может включать в себя следующие слои:

* контроллер домена,
* Web-сервер,
* сервер приложений,
* сервер баз данных.

Сервер может общаться с клиентом как с установлением соединения, например по протоколу TCP/IP, так и без установления соединения, например по протоколу UDP. Сервер может либо сохранять, либо не сохранять информацию об обслуживаемом клиенте. Информацию о клиенте в этом случае называют состоянием. Информация о состоянии может быть полезна, если сеанс работы с клиентом включает обмен несколькими сообщениями.

Объектно-ориентированные системы можно также в определенном смысле рассматривать как частный случай систем типа «основная программа-сопрограммы», при этом объекты могут находиться как на одном, так и на разных хостах. Отличительной особенностью объектов является то, что объекты полностью инкапсулируют код и данные, и общение между объектами происходит либо посредством вызова процедур, либо посредством механизма сообщений. Объектно-ориентированные системы также поддерживают механизмы наследования и/или делегирования.

Одна из проблем, возникающих в процессе функционирования объектно-ориентированных систем, состоит в том, что объект должен знать, где находится тот объект, с которым он хочет взаимодействовать и какими интерфейсами он обладает. Привязка к объекту может осуществляться либо в статике, либо в динамике. Основное достоинство объектно-ориентированных систем – возможность сокрытия данных от пользователя. Это позволяет изменять внутреннее представление объекта, не уведомляя об этом клиента.

Другим достоинством объектно-ориентированных систем является то, что они естественным образом поддерживают процесс распараллеливания вычислений. Объектно-ориентированная система может рассматриваться как коллекция взаимодействующих агентов. Каждый объект при этом может быть как клиентом, так и сервером. Объекты могут располагаться в одном процессе, храниться в библиотеках, находиться в разных процессах, работающих на одном хосте и, наконец, располагаться на разных хостах. Важная разновидность объектов – компоненты, которые являются объектами со специальными свойствами. Широко известны такие компонентные системы, как COM, CORBA, EJB.

Иерархические многоуровневые системы используются преимущественно для построения крупномасштабных приложений. Система содержит несколько слоев. Каждый слой можно рассматривать как виртуальную машину для вышележащего слоя. Каждый из слоев можно рассматривать также как набор сервисов для вышележащего слоя, таким образом, вышележащий слой работает в режиме клиента, а нижележащий – в режиме сервера. Обычно каждый из слоев взаимодействует с соседними слоями через четко определенные интерфейсы. Данный стиль повсеместно используется для построения стеков протоколов, в частности коммуникационных протоколов.

Другим возможным применением данного стиля являются операционные системы. Структура операционной системы с трехуровневой организацией включает три уровня:

* уровень ядра системы,
* уровень системных утилит,
* уровень, на котором работает прочее программное обеспечение. Данный подход был настолько популярен, что в процессорах фирмы Intel, начиная с модели Intel 286, заложена аппаратная поддержка работы четырехуровневой системой.

Следует отметить, что современные ОС, такие как Unix и все ОС фирмы Microsoft, используют ОС, работающие только на двух уровнях, однако для построения стеков протоколов данный стиль продолжает широко использоваться. Основной недостаток состоит в том, что не все алгоритмы можно реализовать в виде многослойной архитектуры. Основное достоинство данного стиля состоит в том, что он позволяет вести разработку кода для каждого из слоев независимо и иметь несколько вариантов реализации кода, кроме того, имеется возможность безболезненно модифицировать код послойно.