Кондратьев Павел Сергеевич

УлГТУ ФИСТ, ИВТАСмд-11

АРХИТЕКТУРА И СОСТАВ ПО АС

В наиболее широком плане к категории АС относятся все системы, интенсивно использующие программное обеспечение. Такой тип систем обеспечивает информатизацию процессов: в рамках «Электронных государств»; систем eбизнеса и e-систем другого типа, в том числе систем, обеспечивающих информатизацию в рамках предприятий и организаций; различного рода встроенных систем, а значит, и систем типа АС, а также других систем разнородных типов.

Реальность такова, что эволюция систем, используемых человеком, движется в направлении всё большей информатизации, увеличивающей класс систем АС типа.

В результате ПО начали делить на системное и прикладное (специальное). Системное ПО – это комплекс программ, обеспечивающий эффективное управление компонентами вычислительной системы, такими как процессор, оперативная память, каналы ввода вывода, сетевое оборудование. При этом системное ПО выступает как «межслойный интерфейс», с одной стороны которого – аппаратура, а с другой – приложения пользователя.

В отличие от прикладного программного обеспечения, решающего конкретные прикладные задачи, системное лишь обеспечивает работу других программ, управляет аппаратными ресурсами вычислительной системы и т.д.

Однако, основные проблемы разработки АС проявляются на уровне разработки их программного обеспечения (ПО), что привело автора к решению спуститься в учебном пособии с уровней архитектур государственных и отраслевых АС и даже с уровня архитектур «предприятия» на уровень архитектур АС без указания специфики АС. Поэтому в последующем тексте термин «архитектура» часто используется в смысле «архитектура ПО».

Таким образом, на сегодняшний день различаются следующие категории функций ПО АС:

* формализованные и стандартизованные, такие как управление вычислениями, связывание и синхронизация процессов обработки данных разного уровня, управление экраном, обработка движений мыши и нажатия клавиш клавиатуры, управление данными и другими ресурсами БД, манипулирование данными БД. Этот слой ПО называется общим программным обеспечением (ОПО);
* недостаточно формализованные, однако для них разработаны стандартные интерфейсы доступа, реализующие взаимодействие с другими слоями ПО, называется общесистемным программным обеспечением (ОСПО);
* неформализованные, реализующие функциональные потребности органа управления на конкретных этапах их работы при решении задач управления, обеспечивающие взаимодействие должностных лиц с АС, составляют специальное программное обеспечение (СПО). СПО должно соответствовать основным этапам логикоаналитической и расчетной деятельности соответствующего органа управления.

**Место архитектурных решений**

Содержание определений архитектуры ПО явно и неявно указывает на место архитектурных решений и их роль в разработке АС.

В своей презентации доклада «Архитектура ПО» G.Booсh указывает на место разработки архитектуры (рис.1) в итерационном процессе типа RUP.

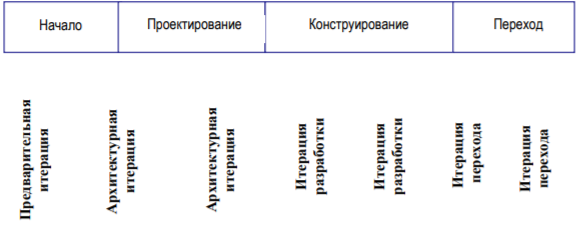


Рисунок 1 Итеративный процесс разработки

Подчёркивая особую важность Use-Case моделей в процессе разработки АС, И. Якобсон включает архитектуру в процесс, как показано на рис.2.

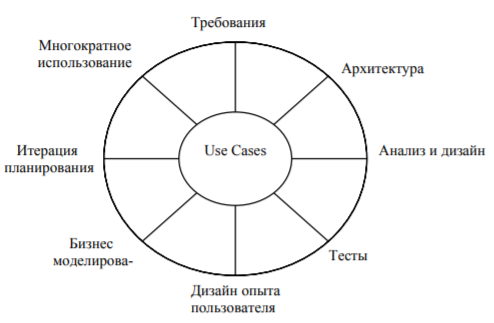


Рисунок 2 Use-case центрированная разработка

В современной практике разработки АС широко используется спиральноитеративная модель жизненного цикла, что приводит к необходимости возврата к вопросам разработки архитектуры (рис.3) с каждым циклом спирали.



Рисунок 3 Спиралевидный процесс разработки

В процессе разработки АС границы между этапами жизненного цикла и итерациями в рамках одного и того же этапа «размыты». Это особенно типично для этапов анализа требований (результатом которого является Система Требований − СТ) разработки архитектуры (результатом которого является Архитектурное Описание − АО). Более того, для практики (рис.3) типичны и возвраты от вопросов архитектуры к вопросам о требованиях, поскольку нормативы на АО являются очень важным источником требований

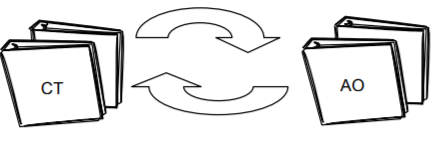


Рисунок 4 Взаимодействие требований и архитектуры

Разумеется, каждый из этапов «анализа» и «архитектуры» приводит к определённой форме существования АС в рамках её жизненного цикла, что (с некоторыми деталями) показано на рис.5.

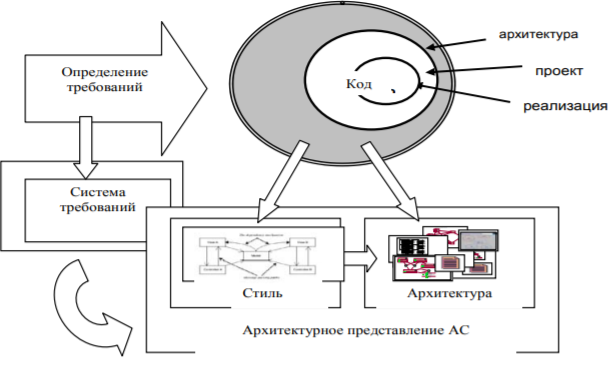


Рисунок 5 Архитектура как форма концептуального существования АС

На рисунке отражён тот факт, что формы существования АС на ранних этапах разработки являются концептуальными, то есть представляют собой связную совокупность текстовых (в том числе табличных) документов и графических моделей, и диаграмм.

**Роль архитектурных решений**

Явное построение и использование архитектуры в разработке АС не раз демонстрировало важные позитивные эффекты. Обобщённое представление позитивных эффектов представлено на рис.6.

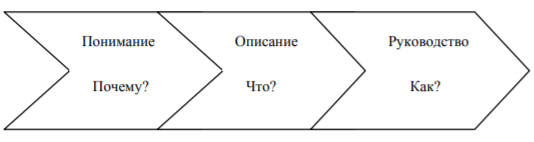


Рисунок 6 Обобщённое предназначение архитектуры

Более детально, архитектура:

* вносит вклад в извлечение тре6бований и формирование их системы;
* ясно показывает совокупность ранних проектных решений;
* предписывает организационную структуру АС (хорошо структурированные системы полны образцов);
* является общим архитектурным базисом для линеек программных продуктов;
* даёт возможность учитывать, согласовывать и предсказывать характеристики качества, в том числе и по результатам её исследования;
* даёт информацию о распределении работ и их календарных планах;
* даёт возможность для более точной оценки стоимости и расписания работ;
* снижает риски;
* - является первой формой существования АС, которая может быть проверена (испытана) как целое;
* предоставляет возможность для переноса или повторного использования стилей и архитектурных каркасов;
* приносит выгоду в ограничении «словаря» альтернатив проекта или его частей;
* помогает в эволюционном прототипировании;
* оформляет концептуальную целостность АС и процесса её разработки;
* обслуживает понимание и взаимопонимание в индивидуальной и коллективной работе;
* есть средство выражения мыслей и рассуждений в коммуникации лиц, вовлечённых в разработку АС;
* предоставляет возможность рассуждать о потенциальных изменениях по ходу разработки АС и вносить вклад в управление такими изменениями;
* может быть базисом для изучения системы.

Приведённый перечень от разработки архитектуры впечатляет и, поэтому, трудно объяснить «Почему архитектуре до сих пор уделяется так мало внимания в разработке АС?». Разумеется, разработка архитектуры дорого стоит, но не дороже чем негативы от её отсутствия.

**Архитектура ПО современный АС**

Появление корпоративных информационных систем, реализованных на компьютерных сетях, и естественная распределенность

организационных структур множества учреждений и предприятий

привело к развитию информационных систем, клиентские приложения которых располагались на рабочих местах пользователей. Децентрализованный подход, лежащий в основе этих систем, позволяет располагать приложения там, где они наиболее востребованы. Такой подход существенно повышает эффективность доступа к данным системы. Но это требует проведения фрагментации кода приложения и программного обеспечения в целом для распределенного размещения по сети.

**Двухзвенная архитектура «клиент-сервер»**

Первым шагом в направлении фрагментации монолитной архитектуры АС можно считать разработку архитектуры «клиент-сервер», становление которой связано, прежде всего, с применением систем управления базами данных (СУБД). СУБД обеспечивала унифицированное решение задач доступа к информационным источникам, поэтому возникла необходимость перераспределения функций между прикладной программой и СУБД. При двухзвенной клиент-серверной обработке каждое приложения разделяют выполняемые функции между двумя частями:

* клиентской, отвечающей за целевую обработку данных и организацию взаимодействия с пользователем. К клиентской части относятся различные программы, написанные как пользователями, так и поставщиками СУБД, внешние или встроенные по отношению к СУБД. Программа-клиент организована в виде приложения, работающего «поверх» СУБД, и обращающегося для выполнения операций с данными к компонентам СУБД через интерфейс внешнего уровня;
* серверной, обеспечивающей хранение данных и обработку запросов. Серверная часть – это программа, реализующая функции СУБД: определение данных, запись-чтение данных, поддержка схем внешнего, концептуального и внутреннего уровней, диспетчеризация и оптимизация выполнения запросов, защита данных.

Таким образом, термин «клиент-сервер» применяется к архитектуре ПО, которое описывало распределение процесса вычисления по принципу взаимодействия двух программных процессов, один из которых в этой модели назывался «клиентом», а другой – «сервером». Клиентский процесс является «активной программой» и запрашивает некоторые услуги, а серверный процесс, являясь «пассивной программой», обеспечивает их выполнение и возврат клиентской программе результатов выполненной программы. При этом предполагается, что серверный процесс может обслуживать множество клиентских процессов.

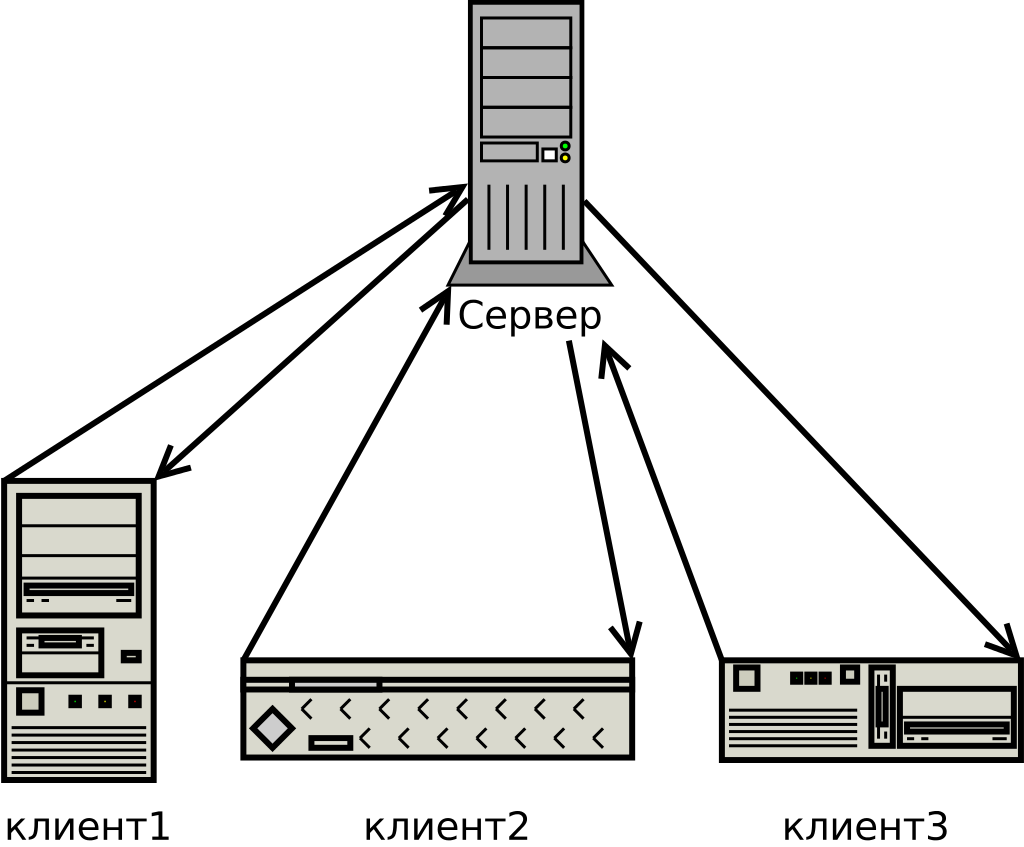


Рисунок 7 Пример двухуровневой архитектуры

**Недостатки**

1. Неработоспособность сервера может сделать неработоспособной всю вычислительную сеть. Неработоспособным сервером следует считать сервер, производительности которого не хватает на обслуживание всех клиентов, а также сервер, находящийся на ремонте, профилактике и т. п.
2. Поддержка работы данной системы требует отдельного специалиста – системного администратора.
3. Высокая стоимость оборудования.

Необходимость масштабирования систем по мере добавления новых функций управления, а также потребности доступа к ресурсам других АС выявили слабые стороны двухуровневой архитектуры «клиент-сервер». На сегодняшний день стремительно усложняющиеся приложения потенциально требуют развертывания их программного обеспечения на сотнях и тысячах компьютеров конечных пользователей.

Поэтому был предложен новый вариант архитектуры «клиент-сервер», который был, в первую очередь, направлен на решение проблемы масштабируемости приложений.

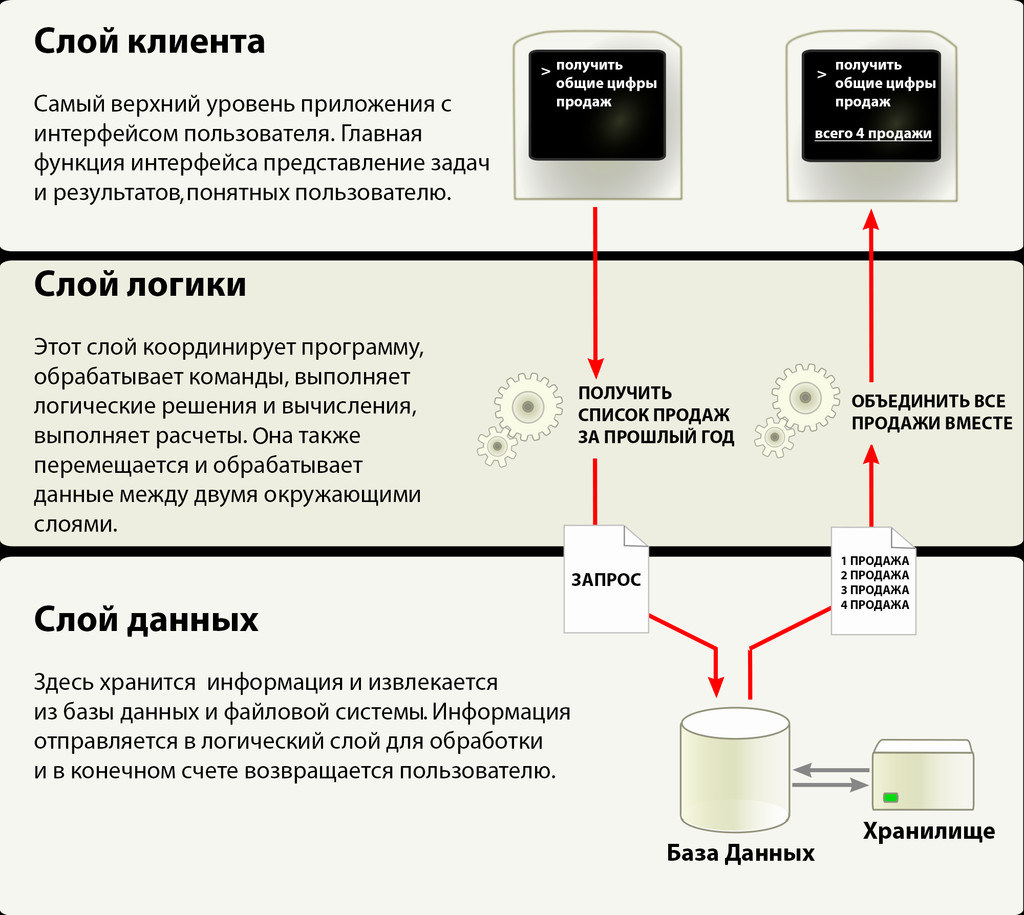


Рисунок 8 Пример трехуровневая схема

Трёхуровневая архитектура – архитектурная модель программного комплекса, предполагающая наличие в нём трёх компонентов: клиента, сервера приложений (к которому подключено клиентское приложение) и сервера баз данных (с которым работает сервер приложений).

**Клиент** (слой клиента) – это интерфейсный (обычно графический) компонент комплекса, предоставляемый конечному пользователю. Этот уровень не должен иметь прямых связей с базой данных (по требованиям безопасности и масштабируемости), быть нагруженным основной бизнес-логикой (по требованиям масштабируемости) и хранить состояние приложения (по требованиям надёжности). На этот уровень обычно выносится только простейшая бизнес-логика: интерфейс авторизации, алгоритмы шифрования, проверка вводимых значений на допустимость и соответствие формату, несложные операции с данными (сортировка, группировка, подсчёт значений), уже загруженными на терминал.

**Сервер** **приложений** (средний слой, связующий слой) располагается на втором уровне, на нём сосредоточена большая часть бизнес-логики. Вне его остаются только фрагменты, экспортируемые на клиента (терминалы), а также элементы логики, погруженные в базу данных (хранимые процедуры и триггеры). Реализация данного компонента обеспечивается связующим программным обеспечением. Серверы приложений проектируются таким образом, чтобы добавление к ним дополнительных экземпляров обеспечивало горизонтальное масштабирование производительности программного комплекса и не требовало внесения изменений в программный код приложения.

**Сервер баз данных** (слой данных) обеспечивает хранение данных и выносится на отдельный уровень, реализуется, как правило, средствами систем управления базами данных, подключение к этому компоненту обеспечивается только с уровня сервера приложений.

Особенно четко преимущества трехзвенной архитектуры проявляются при интеграции разнородных ресурсов распределенной системы, так как современное программное обеспечение промежуточных слоев включает в себя функциональность, необходимую для удаленного вызова процедур с целью выполнения операций доступа к данным узлов распределенной системы.