Министерство науки и высшего образования российской федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
Ульяновский государственный технический университет

Методические рекомендации

Дисциплина (модуль):
Б1.В.05 Архитектурное моделирование в проектировании АС

Уровень образования: магистратура

Квалификация: магистр

г. Ульяновск, 2021г.

П.И. Соснин

Архитектурное моделирование автоматизированных систем

Учебное пособие

2008

УДК 319.766 (075)

Рецензенты: доктор технических наук,

Соснин П.И.
С66 Архитектурное моделирование автоматизированных систем: учебное пособие/ П.И, Соснин. –

Учебное пособие предназначено для студентов, обучающихся по направлению «Информатика и вычислительная техника», а также по специальностям «Вычислительные машины, комплексы, системы и сети» и «Информационные системы и технологии». Пособие может быть также использовано студентами других специальностей, профиль которых связан с разработками автоматизированных систем, интенсивно использующих программное обеспечение.

УДК 319.766 (75)
ББК Я7

Содержание

Оглавление

Введение

В соответствии со стандартами 34-ой серии автоматизированные системы (АС) представляют собой «организационно-технические системы, обеспечивающие выработку решений на основе автоматизации информационных процессов в различных сферах деятельности или в их сочетании». В современных международных стандартах (например, в стандарте IEEE 1471) такой тип систем принято называть «software-intensive systems» («системами, интенсивно использующими программное обеспечение»), что указывает на «существенное влияние программного обеспечения ПО, входящего в их состав, на проектирование, конструирование, материализацию и эволюцию АС»[89].

В разработках АС накоплен определённый опыт, который позволяет считать такой вид деятельности одним из наиболее сложных и непредсказуемых видов работ, которые слишком часто завершаются с превышением запланированного на них времени и/или бюджета. Кроме того, разработки АС часто завершаются в более бедной версии результата или же прекращаются без достижения каких-либо положительных результатов. Наиболее убедительно и с попытками выявления причин эта особенность разработок АС отражена в отчётах Standish Group, одной из наиболее известных экспертных компаний по проблемам информационных технологий [70].

К числу основных причин такого положения дел, которое по основным составляющим сохраняется по настоящее время [15], относятся:

- недостаточное вовлечение в разработку пользователей и других лиц, заинтересованных в создании АС;

- неполнота требований и спецификаций;

- неуправляемое (и часто из-за субъективных причин) изменение требований и спецификаций по ходу разработки АС;

- недостаточная автоматизированная поддержка исполняемых действий;

- использование «незрелых» (слабо проверенных на практике и не дающих устойчивые позитивные эффекты) технологий;

- недостаточность выделенных или привлекаемых ресурсов;

- нереалистичные ожидания от разработки;

- использование неявных и/или смутных целей;

- нереальные временные рамки исполнения разработок.

Опасное влияние указанных причин и ряда других причин на успешность разработок АС обусловлено, в первую очередь, их сложностью с позиций программного обеспечения. Разработка такого обеспечения для конкретной АС обычно носит длительный, дорогостоящий и уникальный характер, что не способствует целенаправленному накоплению опыта подобных разработок. По сути дела в разработках АС создаются их образцы, в каждом из которых достигнута определённая степень успешности.

Степень успешности разработки конкретной АС адекватно оценивается и подтверждается только в процессе её практического использования. Успешные АС используются как источники разнородных образцов для их повторного использования, в первую очередь как источники образцов для проектных решений при разработке программного обеспечения. Среди таких образцов важное место занимают образцы архитектурных проектных решений, без применения которых немыслима разработка программного обеспечения современных АС.

Разработка архитектуры и включение её в процесс разработки АС способствует снижению стоимости разработки (за счёт улучшения коммуникативного взаимодействия между разработчиками, открывает возможность более ранней оценки альтернатив и рисков), снижает время выхода на рынок (за счёт возможности повторного использования предыдущих решений и управляемой эволюции продукта), снижает стоимость сопровождения (за счёт объединения в группы изменений, поддающихся предвидению); способствует улучшению качества продукта и приводит к другим позитивным эффектам (раскрываются ниже).

Принципиальная роль архитектуры АС была осознана всего лишь около 15 лет назад. За эти 15 лет в этой предметной деятельности было создано так много, что этот период среди специалистов по разработкам АС получил название «золотого века архитектуры АС». Одно из достижений этого века – отдельная

профессиональная дисциплина (самостоятельная предметная область) и учебная университетская дисциплина, часто с названиями «Архитектура автоматизированных систем» или «Архитектура программных систем».

Во втором издании учебного пособия раскрывается специфика предметной области «Архитектура автоматизированных систем» по зарубежным (в основном) и российским источникам, включающим стандарты, монографии, статьи и отчёты, представленным в Интернете. Отбор материалов проводился в основном по двум-трём первым страницам Yandex, Rumbler и Google для ключей доступа, раскрывающих архитектуру программного обеспечения и архитектуру систем, интенсивно использующих программное обеспечение. Для предмета пособия в книге используется общий термин «архитектура».

Учебный материал пособия распределён по четырём главам. Содержание трёх первых глав идентично содержанию первого издания, а четвёртая глава содержит информацию о совершенствовании нормативной базы архитектурного моделирования за последнее десятилетие.

В первой главе определяется класс «автоматизированных систем, интенсивно использующих программное обеспечение», с акцентом на специфику архитектуры таких систем. Приводится ряд определений архитектуры с учётом её значимости, раскрываются место и роль архитектуры как формы концептуального существования АС. Представляется ретроспектива исследований и разработок в области архитектуры АС за последние 15 лет.

Во второй главе внимание акцентируется на архитектурных образцах, стандартах и архитектурных концептуальных схемах. Раскрывается представление архитектуры АС в форме системы архитектурных видов, согласованных с интересами групп лиц, заинтересованных в разработке АС. Обобщённо демонстрируются архитектурные схемы Дж. Захмана, DoDAF, MoDAF, TOGAF и FEAF. Проводится сопоставление рабочих архитектурных схем, используемых в технологиях разработки АС различными корпорациями.

Материал третьей главы связан с вопросами разработки архитектур АС. С позиций разработки предлагается рассматривать архитектуру как специфический вид автоматизированных систем, интенсивно использующих программное обеспечение. Представляются базовые архитектурные парадигмы (объектно-

ориентированная, компонентно-ориентированная и сервисно-ориентированная парадигмы), варианты архитектур (в том числе с ориентацией на события, модели и паттерны) и архитектурные стили.

Особое внимание уделяется вопросам качества АС, языкам описания архитектур и методам их проектирования, а также вопросам оценки и документирования результатов архитектурного моделирования. Обобщённо представляются идеи аспектно-ориентированного анализа и проектирования АС. Каждая из глав заканчивается списком контрольных вопросов, на каждый из которых приведён ряд потенциальных ответов.

В четвертой главе, расширяющей первое издание пособия, опубликованного в 2007 году, представлены материалы по нормативному совершенствованию стандарта IEEE-1471, которое закреплено в стандарте **ISO/IEC/IEEE 42010:2011** и его русифицированной версии **ГОСТ Р 57100—2016.**

В четвёртой главе раскрыты причины: расширения ответственности нормативов архитектурного моделирования **(от систем, интенсивно использующих программное обеспечение до систем любого типа**); введения дополнительных структур в концептуальную модель (фреймворк) стандарта IEEE-1471; дополнения новой версии концептуальной модели уточнениями в виде совокупности подчинённых моделей; уточнения терминологии.

Ещё одним расширением материалов первого издания является включение Приложения, которое содержит шаблон архитектурного описания системы.

Учебное пособие предназначено для студентов, обучающихся по направлению «Информатика и вычислительная техника», а также по специальностям «Вычислительные машины, комплексы, системы и сети» и «Информационные системы и технологии». Пособие может быть также использовано студентами других специальностей, профиль которых связан с разработками автоматизированных систем, интенсивно использующих программное обеспечение. Совсем близко время, когда других систем в окружении человека просто не будет.

ГЛАВА ПЕРВАЯ. АРХИТЕКТУРА АВТОМАТИЗИРОВАННЫХ СИСТЕМ

1.1. Автоматизированные системы

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 *e*-бизнеса и *e*-систем другого типа, в том числе систем, обеспечивающих информатизацию в рамках предприятий и организаций; различного рода встроенных систем; CAD-CAM-CAE–систем, обслуживающих разработку систем различного рода, 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.

Каждая АС имеет ту или иную архитектуру. Если при разработке АС её архитектурные представления использовались явно, то, как показывает практика, это повышало степень успешности разработок и приводило к другим положительным эффектам. Если же такое внимание к архитектуре отсутствовало, то у АС формировалась **случайная** (accidental) архитектура [14], позитивный или негативный **вклад** которой в разработку **не планировался и не контролировался**. Случайная архитектура может оказаться и очень удачной, но вероятность этого мала. Часто (и с разными целями) случайная (удачная) архитектура после завершения разработки регистрируется, например, для целей обучения обслуживающего персонала или для целей повторного использования (разработки очередных версий или родственных систем).

В зависимости от типа АС различают «Архитектуру электронного государства», «Архитектуру предприятия», «Архитектуру системы», «Архитектуру программного обеспечения» и другие варианты архитектур систем или их подсистем и их представлений с различных точек зрения. Однако, основные про-

блемы 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».

Акцент на архитектурах ПО обусловлен тем, что содержание вариантов употребления терминов «Архитектура ПО» и «Архитектура автоматизированных систем, интенсивно использующих ПО», сложилось за последние 15 лет. Это содержание является «молодым» по сравнению с другими вариантами содержания понятия «архитектура». В то же время новые архитектурные понятия и их материальные воплощения оказали существенное влияние на процессы разработки АС любых типов.

Вопрос специфики АС в пособии раскрыт не на уровне разработки, а на уровне использования архитектур АС. Таким образом, ниже проводится обзор архитектурных представлений АС с позиций их программного обеспечения, разумеется, в контексте АС как целого. Заметим, что достаточно часто архитектуру АС приходится строить (и использовать) с учётом её представления (рис.1.1) в контексте «предприятия».

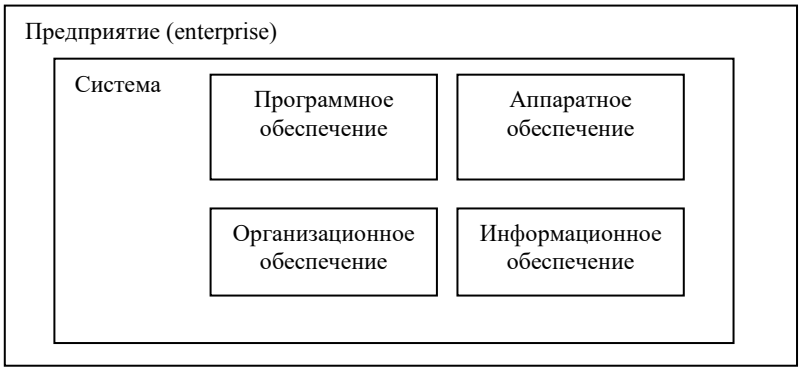


Рис.1.1. Обобщённые отношения между архитектурами разных уровней

Такой учёт, раскрытый в [25], выводит на взаимосвязанную совокупность архитектур:

- **программного обеспечения** (software architecture), на архитектурах которого и будет сосредоточено содержание обзора;

- **аппаратного обеспечения** (hardware architecture), в базисе основных единиц компьютерной и телекоммуникационной техники;

- **информационного обеспечения** (information architecture);

- **организационного обеспечения** (organizational architecture), раскрывающего связность организационных структур (в том числе на уровне персонифицированных ролей и ответственности) с бизнес-процессами;

- **предприятия** (enterprise architecture) с позиций бизнеса, в общем случае выходящего за рамки компании.

Вопросы и решения по архитектурам аппаратного, информационного и организационного обеспечения, а также по архитектурам предприятий в пособии будут затрагиваться по мере необходимости для раскрытия архитектур ПО. В то же время предварительно отметим следующее:

1. Архитектуры аппаратного обеспечения традиционная и достаточно устоявшаяся форма, используемая в представлении АС. Опосредовано она находит отражение в моделях развертывания ПО.

2. В основе архитектуры информационного обеспечения лежат аналогии с АС, интенсивно использующими данные, что находит отражение в архитектурных стилях ПО.

3. Архитектурные решения по организационному обеспечению учитываются во многих моделях и артефактах, сопровождающих разработку современного ПО.

4. Предприятия, интенсивно использующие ПО, являются определённым классом АС. Архитектуры таких систем нашли богатое представление в образцах и стандартах [77-80,89], которые используют заимствования из опыта архитектурных представлений и, в то же время служат источниками полезной информации и решений как при разработках АС других типов, так и при разработках ПО. Следует отметить, что достаточно полный обзорный материал по «enterprise architecture» представлен в [85].

5. Достаточно полный обзор по опыту «электронных государств» и используемых архитектурных решениях в АС такого типа приведён в [84].

1.2. Определения архитектуры и её значимость

Осознанному и конструктивному применению терминов «архитектура ПО» и «архитектура систем, интенсивно использующих ПО», около 15 лет [43]. За этот период его варианты употребления детализировались дифференцировались и уточнялись. Представительные коллекции вариантов употреблений термина приведены в [47]. Так, например, WEB-коллекция определений SEI (Software Engineering Institute of Carnegie Mellon) содержит около 70 определений, объединённых в разделы, включающие раздел, в котором каждое заинтересованное лицо может зарегистрировать собственное понимание и определение «архитектуры ПО».

В профессиональной литературе по программной инженерии наиболее часто ссылаются на следующие определения:

1. (SEI) Архитектура программы или АС – это структура (или набор структур) системы, которая включает программные элементы, наблюдаемые из вне свойства этих элементов, и взаимодействие между ними.

2. (RUP – Rational Unified Process) Архитектура включает: существенные решения по организации АС; выделенные структурные элементы с их интерфейсами и функциями, обеспечивающими взаимодействие элементов в рамках АС; композиции структурных и функциональных элементов в подсистемы, в соответствии с архитектурным стилем, который определяет их организацию, элементы, интерфейсы, взаимодействие и композицию в более сложные образования. Архитектура затрагивает не только структуру и поведение, но также использование, функциональность, и другие аспекты.

3. (IEEE – Institute of Electrical and Electronic Engineers) В соответствии с рекомендуемой практикой архитектура определяется как фундаментальная организация системы, связывающая её компоненты, их взаимодействие между собой и с окружающей их средой, а также принципы управления проектированием и развитием.

4. Архитектура, в первую очередь, – это совокупность принципов структурирования, которые предоставляют возможность в контексте АС как целого

представить её с помощью набора более простых систем, каждая из которых определена в соответствующем локальном контексте.

В работе [26] проведён анализ третьего (из представленных выше вариантов) определения архитектуры со следующим толкованием его отдельных позиций:

1. Архитектура в любой версии определяет структуру. Разумеется, архитектура не сводится только к структурам (деление на части, элементы, интерфейсы, связи, взаимодействие), но структурный аспект в её содержании и его материализации является наиболее существенным. Функции элемента структуры могут выполнять модуль, подсистема, процесса, библиотека и т. д., причём каждый из таких элементов может быть реализован различным образом. Например, коннектор может быть сокетом, синхронным или асинхронным, ассоциированным с определённым протоколом и т. д.

2. Архитектура определяет поведение. Архитектура определяет не только структуры АС через элементы и связи, но также и взаимодействие элементов структур, то есть взаимодействие, осуществляемое во времени. В этом плане архитектура представляет поведение АС в терминах процессов их функционирования. Для выражения динамики процессов в архитектурных описаниях используют различные средства, например «диаграммы последовательностей» на языке UML.

3. Архитектура фокусирует свои интересы на существенных элементах структуры и поведения. В построениях архитектуры АС абстрагируются до самого существенного в АС как целом. Детали целого и детали частей АС раскрываются в других формах существования АС, в первую очередь в её «концептуальном проекте», разработку которого проводят опираясь на архитектуру АС.

4. **Архитектура находит баланс для совокупности требований лиц, заинтересованных в её разработке.** В типичном случае разработки конкретной АС система требований, выявленная и сформированная до построения её архитектуры, нуждается в адаптации к реалиям средств, выделенных на разработку, и к реалиям будущего использования АС, учитывающим её сопровождение и эволюцию. Основные решения по адаптации, включающей построение сбалансированной совокупности требований, принимаются при построении архитек-

туры АС. Наиболее сложна балансировка нефункциональных требований к АС, отвечающих за её качество и/или за качество процесса разработки.

В нахождении баланса следует принимать в расчёт то, что:

- конечный пользователь заинтересован в интуитивно понятном и предсказуемом поведении АС при взаимодействии с ней, в таких характеристиках качества как «удобство», «понятность» и «обучаемость»;

- **системный администратор** заинтересован в интуитивно понятных средствах для его работы, в том числе в средствах, помогающих в задачах мониторинга состояний АС и её поведения;

- **маркетолог** заинтересован в выигрышных функциях АС на рынке родственных (для АС) систем, достаточном для маркетинга времени, ясном позиционировании среди родственных систем и цене;

- **покупатель** заинтересован в цене, стабильности работы АС и плане поставки и введения в эксплуатацию;

- **разработчик** заинтересован в ясных и непротиворечивых требованиях к АС, а также в возможности использования простого подхода к проектированию;

- **руководитель проекта** заинтересован в предсказуемости развертывания работ и их результатов, в трассируемости проекта, плане действий, производительных средствах, доступных ресурсах, в первую очередь финансовых средств;

- **лицо, ответственное за сопровождение АС**, заинтересовано в понятности АС и решений, вложенных в её разработку, а также в документированности и легкости модификации.

5. Архитектура интегрирует существенные решения на основе принципов рациональности. Существенные решения и формы их интеграции в архитектуре АС должны быть рационально представлены и обоснованы. Решения должны не только декларироваться, но и объясняться с позиций «Почему им было отдано предпочтение среди альтернатив».

6. Архитектура должна быть согласована с архитектурным стилем или их совокупностью. В разработках архитектур следует использовать накопленный опыт успешных разработок, в первую очередь опыт, вложенный в архитектурные стили, каждый из которых является образцом общих типовых решений.

Понятие архитектурного стиля, образцы стилей и их классификация представлены в пособии в п.3.4.

7. На архитектуру оказывает воздействие её среда. Конкретная АС разрабатывается в конкретной среде разработки и для конкретной среды использования. Факторы этих окружений не могут не влиять на архитектурные решения. К числу основных факторов, оказывающих влияние на архитектуру, относятся: предназначение (миссия) АС; лица, заинтересованные в разработке АС; ограничения разных типов; уровень квалификации лиц, вовлечённых в процессы использования АС.

8. Архитектура воздействует на структуру команды, которая разрабатывает АС. Для реализации того, что вложено в архитектуру, требуются вполне конкретные умения квалифицированных исполнителей. По архитектуре можно заключить, какие специалисты и в каком количестве потребуются для разработки АС, что и должно использоваться при формировании коллектива разработчиков.

9. Архитектура присутствует в любой АС. Даже если при разработке АС вопрос о создании её архитектуры, по тем или иным причинам, не поднимался, разработанная АС будет иметь архитектуру, но эта архитектура будет носить случайный (по принятым проектным решениям) характер.

10. Архитектура имеет определённые границы. Как уже отмечалось выше и было представлено на рис.1.1, различают архитектуру в разных объемах: программного обеспечения, аппаратного обеспечения, информационного обеспечения; организационного обеспечения, предприятия. И всё же архитектуры ПО являются тем центром, около которого или от которого строят архитектуры других названных объёмов.

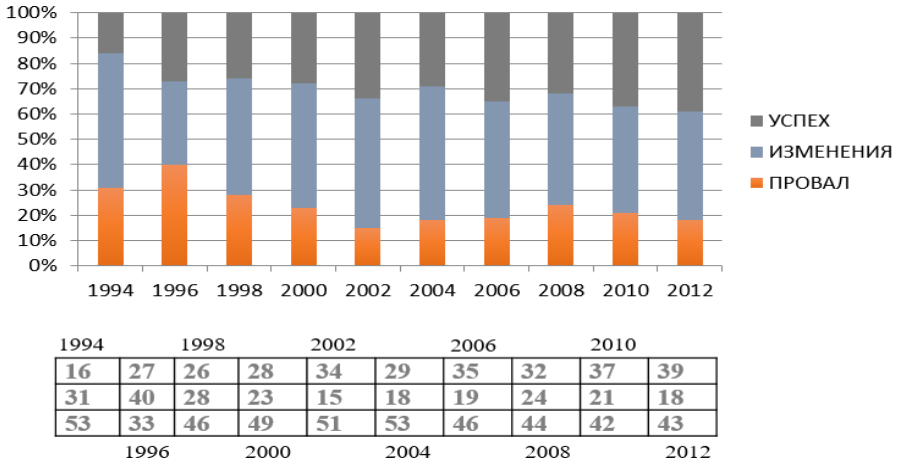
Завершая пункт, отметим, что проведённое толкование 3-го определения архитектуры применимо и для других названных выше и неназванных определений архитектуры.

1.3. Архитектура как форма концептуального существования АС

Как уже отмечалось, число определений архитектуры велико. Одно и безоговорочно принятое разработчиками АС определение архитектуры АС отсут-

ствует. Даже представленный выше анализ построен как перечень существенных проявлений архитектуры, а не как ряд проявлений, исходящих из одной сути. Поскольку авторские определения архитектуры поощряются, представим ещё одно понимание сути архитектуры АС, конструктивное воплощение которой в разработке АС носит принципиальный характер.

На рис.1.2 представлена обобщённая картина успешности разработок АС, интегрирующая положение в этой области за 10 лет. Степень успешности, выраженная в процентах числа проектов, завершившихся в соответствии с исходными замыслами и планами, чрезвычайно низка (около 30%). Значительное число разработок либо прекращается, либо превышает запланированное время и /или средства, либо завершается в более бедной версии. Факт низкой успешности в этой области означает, что разработчики АС до сих пор не получили очень важных инструментально-технологических средств.



1994		1998		2002		2006		2010	
16	27	26	28	34	29	35	32	37	39
31	40	28	23	15	18	19	24	21	18
53	33	46	49	51	53	46	44	42	43
	1996		2000		2004		2008		2012

Рис.1.2. Степень успешности разработок АС (Standish Group)

Исходя из специфики проблем разработки АС, роль таких средств могли бы выполнить средства искусственного интеллекта (ИИ), поскольку слишком многое в разработке АС опирается на взаимодействия с «опытом» и «знанием», в первую очередь, для «принятия решений» и «решения задач». Обращаясь к опыту ИИ, следует ожидать, что, возможно, существующие средства ИИ придётся адаптировать к специфике разработки АС, а, возможно, придётся разрабатывать и новые средства ИИ.

По глубокому убеждению автора степень успешности разработок АС можно существенно повысить за счёт инструментально-технологических средств, позволяющих оперативно, как только в этом возникает необходимость, интегрировать интеллектуальные усилия лиц, заинтересованных в разработке АС. В основе этого убеждения лежит следующее:

1. Интеллект создавался природой для решения вполне определённых проблем сохранения вида, за счёт приобретения и использования внегенетического опыта, полученного конкретным лицом или группой лиц при взаимодействии с окружающей средой.

2. Для расширения круга задач, доступных интеллекту и, тем самым, для развития индивидуального и коллективного опыта, обеспечивающего полезное взаимодействие с окружающей средой, интеллект научился использовать различного рода средства, повышающие его «коэффициент полезного действия».

3. Специфика разработки современных АС заключается в том, что по ходу разработки встают задачи (когда и сколько – вопрос второй), для решения которых необходимо оперативно (когда в этом возникает необходимость) интегрировать интеллектуальные усилия групп лиц, заинтересованных в разработке АС. Более того, сложность задач разработки АС превышает уровень сложности задач, для решения которых у современного интеллекта уже есть инструментальные помощники.

4. Реальность разработок АС такова, что за каждую задачу процесса разработки ответственность возлагается на конкретного разработчика, и именно на базе его индивидуального интеллекта следует осуществлять оперативную интеграцию необходимых (для решения задачи) интеллектуальных усилий.

5. В интегрированном интеллекте должны поддерживаться базовые интеллектуальные активности, в первую очередь, интегрированное сознание и интегрированное понимание.

6. Феномен «сознания», отвечающий за «работу со-знанием», проявляется в процессах взаимодействия с «опытом». На «сознание» и «знание» в процессах взаимодействия с «опытом» интеллект возлагает обеспечение доступа к «прецедентам» (базовым единицам опыта, кодирующим типовое реагирование в типовых ситуациях), определённым образом закодированным и включённым в си-

стему «опыта». От осуществления такого доступа существенным образом зависит полезность обращения к «опыту».

7. Специфический вид «работ со-знанием» целесообразно контролировать, что и выполняет интеллектуальная активность, получившая название «понимание».

8 Базовой формой проявления работы сознания, в том числе и интегрированного, являются рассуждения, представления которых могут быть вынесены за пределы мозговых структур, в том числе и для инструментальной обработки.

9. Наиболее важным результатом понимания как средства контроля за работой сознания, а значит, и как средства контроля за рассуждениями, является целостное представление в образном виде (в виде схемы или «картины» другого рода) выполненной сознанием (рассуждениями) работы.

10. Коллективные «рассуждения» целесообразно считать продуктом интегрированного «сознания» только в том случае, когда для каждого индивидуального интеллекта они могут быть представлены и использованы в оперативных целях как «индивидуальные рассуждения + модель интегрированных рассуждений».

11. «Образные представления» можно считать продуктом интегрированного «понимания» только в том случае, когда для каждого индивидуального интеллекта они могут быть представлены и использованы в оперативных целях (в рассуждениях и их контроле) как «индивидуальные образные представления, которым есть место и роль в составе модели интегрированного образного представления».

12. ***Интегрированное концептуальное представление сущности АС, регистрирующее в образной форме всё необходимое для контроля (с помощью понимания) рассуждений по существенным темам, связанным с процессами разработки и использования АС, и есть основное предназначение архитектуры АС.***

Заключительный 12-й пункт перечня, представленного выше, и есть авторское определение «архитектуры АС». Другими словами, **«архитектура есть форма концептуального существования АС, обслуживающая контроль (с помощью понимания) процессов жизненного цикла АС»**.

Авторское определение будет использоваться ниже с различными целями. Сопоставляя его с тем, что представлено выше, можно отметить, что авторское определение согласовано с теми определениями архитектуры, которые уже были приведены, и с толкованиями 3-го определения.

В то же время авторское определение архитектуры акцентирует внимание на принципиальной роли активности интегрированного сознания в формах интегрированных рассуждений.

Оценивая современные технологии разработки АС с позиций инструментальной поддержки интегрированных рассуждений (в рамках оперативной интеграции интеллектуальных усилий разработчиков), можно констатировать, что эта позиция практически упущена. Таких средств в технологиях разработки АС практически нет, в то время как в искусственном интеллекте проблемам и средствам моделирования рассуждений уделяется большое внимание. Так, например, в тематике регулярных европейских конференций по искусственному интеллекту (ECCAI-XXXX) присутствует около двадцати позиций, связанных с моделированием рассуждений разных типов.

Тот факт, что инструментальной поддержке рассуждений, их моделированию и управлению рассуждениями в коллективной разработке уделяется слишком мало времени, наводит на мысль, что в построениях архитектуры не используются важные составляющие (как информационные, так и деятельностные). Учёт таких составляющих способен повысить степень успешности разработок АС.

Поясним включение в определение архитектуры ссылки на **«темы»** рассуждений. В процесс разработки в разные моменты времени включаются лица с разными интересами, способными выразить их адекватно на своём естественно-профессиональном языке. Такие интересы определяют темы для рассуждений, в которые вовлекаются не только специалисты по содержанию темы. В рамках тем происходит концептуальное смешивание различных профессиональных языков, что существенно усложняет понятийный контроль коллективных рассуждений по теме. А значит, по каждой теме следует оперативно формировать толковый словарь по теме и семантические образцы (концептуальные схемы), обслуживающие понимание.

Ещё одно пояснение по образцам для понимания. Подбирая подходящие средства, помогающие пониманию, логично использовать опыт контроля из различных предметных областей, в частности, опыт контроля из теории и практики измерений. В таком виде практик принято использовать образцы, их совокупности и системы образцов, с которыми проводят сопоставление контролируемого результата. Что из себя представляет конкретный контролируемый результат и процесс сопоставления является вторичным.

А значит, для понимания необходимы образцы, с которыми следует проводить сравнения. Следовательно, для выполнения определённой работы сначала следует создать соответствующие ей образцы понимания, их совокупности и систему образцов, с которыми в последующих действиях следует сравнивать последующие акты понимания. Функции таких образцов способны выполнить адекватные концептуальные представления того, что требуется понять. Одной из принципиальных характеристик таких представлений является их целостность.

Разработка АС представляет собой специфическую деятельность, для актов понимания в которой необходимы образцы для понимания не только АС как целого, но и её частей, а также образцы понимания процесса разработки АС и частей этого процесса.

1.4. Проблема сложности

1.4.1. Причины сложности

Процесс разработки АС и его результаты имеют принципиально сложный характер, в основном из-за сложности ПО. Именно сложность является причиной, приводящей к задачам разработки АС, для решения которых приходится интегрировать интеллектуальные усилия.

То, что «Сложность ПО является его существенным свойством, причём не случайным свойством», детально раскрывается в работе [12]. В этой публикации представлены комментарии по следующим факторам, оказывающим воз-

действие на технологии разработки систем и приводящим к сложности ПО (а значит, и АС):

1. Законы физики.
2. Законы ПО.
3. Изменение алгоритмов.
4. Трудности распределения.
5. Проблемы проектирования.
6. Проблемы функциональности.
7. Важность организации.
8. Воздействие экономики.
9. Влияние политики.
10. Недостаток воображения.

Представим каждый из этих факторов детальнее:

1.**Законы физики.** Любая конкретная АС локализована и функционирует в определённом физическом пространстве (например, в рамках распределённой компьютерной сети) и, следовательно, должна подчиняться законам физики (скорость распространения электромагнитных волн, задержки передачи информации между компьютерами, электромеханика и теплотехника компьютерных средств, возможность физических дефектов, вплоть до выхода физических средств из строя и т. д.). Зависимость от физики особенно принципиальна для АС реального времени, сложность которых в существенной мере определяется требованиями к АС, отражающими её связь с реальностью. По темам физики в процессе разработки придется рассуждать, а результаты рассуждений овеществлять в составе АС.

2. **Законы ПО**. Известно, что не все задачи, решение которых необходимо в определённых АС, имеют алгоритмическое решение. Очень часто известные алгоритмические решения «не по карману» разработчикам АС . Практически в любой разработке сталкиваются с новыми задачами, решение которых сначала необходимо построить. Достаточно типичны ситуации «комбинаторного взрыва». Программ без ошибок практически не существует. Даже в «средних» программных системах число возможных вариантов развития вычислений практически необозримо. По темам ПО в процессе разработки придётся рассуждать

больше всего, а результаты рассуждений овеществлять как программное обеспечение АС.

3. **Изменение алгоритмов.** В ситуациях потенциального комбинаторного взрыва приходится вводить управляемые изменения «точных алгоритмов», вводя в него различного рода эвристики и, тем самым, отказываясь от гарантий построения положительного результата. Кроме того, реальность разработки такова, что алгоритм приходится изменять и не раз, адаптируясь к решаемой задаче или к ресурсам, вовлечённым в процесс разработки. По устоявшемуся мнению автора «программирование отличается от других способов решения задач тем, что оно «разрешает» изменять постановки задач. В разработках ПО достаточно типична ситуация, в которой постановка решения задачи принимает свою завершённую формулировку на завершающем этапе разработки АС. Рассуждения по темам изменений являются особо опасными.

4. **Трудности распределения.** Реальность такова, что современные АС обычно распределены в пространстве (корпоративная сеть, телекоммуникационная среда) и обслуживают множество пользователей. Они относятся к классу распределённых (в сети) систем, на разработку которых неявно воздействуют следующие заблуждения: сеть надёжна, задержки равны нулю, пропускная способность каналов неограниченна, сеть защищена, топология сети не изменяется, у сети только один администратор, цена транспортировки равна нулю, сеть однородная. На самом деле это не так. Проблемы распределения приводит к дополнительным темам для рассуждений/

5. **Проблемы проектирования.** В основу проектирования АС положено упрощение процесса проектирования и его результата. В то же время простота понимается и воспринимается по-разному конечными пользователями, разработчиками, обслуживающим АС персоналом и другими заинтересованными лицами. Для конечных пользователей простота связывается с соответствием опыта пользователей и средств их взаимодействия с АС. Для разработчиков АС простота ассоциируется с простотой архитектуры АС. Для тех, кто развёртывает АС, простота связана с простотой средств инсталляции АС. Необходимо следовать известному изречению А. Эйнштейна «Любую вещь необходимо делать простой, но не проще».

В проектировании как нигде велика роль подходящего абстрагирования и образцов. В его основе лежит использование опыта проектирования, как коллективного, так и индивидуального. Основная задача опыта (и взаимодействия с ним) – помочь его владельцам и пользователям во взаимодействии с окружающим миром, в рассматриваемом случае помочь разработчикам АС и их пользователям в их действиях. Представить опыт в среде проектирования и обеспечить взаимодействие с ним является исключительно сложной задачей. Рассуждения являются основным средством взаимодействия с опытом проектирования.

6. **Проблемы функциональности.** Проблемы функциональности АС начинаются на этапе выявления требований и определения спецификаций. Достаточно часто этот этап начинается с обобщённого видения проблемы, смутной идеи и ряда описаний применения АС. Этап является наиболее критичным к возможным ошибкам, которые вводятся в разработку из-за неправильного понимания исходных потребностей, затребовавших разработку АС (неправильного понимания заказчиками АС и её разработчиками, дефектов взаимопонимания между ними и другими лицами, вовлечёнными в разработку). Такого рода проблемы функциональностей АС существуют и на последующих этапах их разработки. Рассуждения о функциональностях для процесса разработки являются особо принципиальными.

7. **Важность организации.** Как уже отмечалось разработка современных АС и их использование носит коллективный характер. Кроме того, разработка осуществляется в распределённой корпоративной среде коллективом разработчиков и другими заинтересованными лицами. Типичная группа разработчиков состоит из 4–8 членов. Но в реальный процесс разработки обычно вовлечены группа групп или более сложная структура (например, иерархическая) из групп групп и их членов. Всё это существенно усложняет продукт разработки и процесс его создания (приходится распределять информацию и функциональности, а также распределять работы и ответственности).

В типичном случае группы разработчиков специализированы, а их члены выполняют различные роли так, чтобы эффективно использовать коллективный и индивидуальный опыт. В своей работе они используют различные формаль-

ные языки и естественно-профессиональные языки (настроенные, например, на тестирование состояний процесса разработки и его продуктов или на документирование архитектуры АС). Задачи формирования таких групп и организации их работы относятся к категории очень сложных.

На их решение оказывают существенное воздействие: цена их решения; планирование работы и вклад каждого участника в общую работу, в том числе в рамках взаимодействия с другими лицами, вовлечённым в процесс разработки АС; проблемы коммуникации в группах, включая разделение знаний и опыта, полномочий и памяти; недостаток времени; время, потерянное на программное и аппаратное обеспечение, которое не работает. Каждое воздействие открывает тему для рассуждений и не одну.

8. **Воздействие экономики.** Любая разработка имеет определённую стоимость. Следует различать суммы, одна из которых соответствует финансовым средствам, которые выделены на разработку АС, а вторая соответствует средствам, потенциально необходимым для выполнения работ в рамках оговоренных требований и ограничений. Практика разработок показывает, что первая сумма (и исходно запланированное время на разработку) обычно в два раза меньше, чем реальные нужды [44].

Для оценок исполнимости разработки по усилиям или времени принято использовать формулу Б. Боэма

$$\textbf{Исполнимость} = \textbf{Сложность}^{\textbf{Процесс}} * \textbf{Команда} * \textbf{Инструменты},$$

в которой: «Исполнимость» = «Усилия» или «Время»,
«Сложность» = «Количество кода, разработанного человеком»,
«Процесс» = «Зрелость процесса или описания»,
«Команда» = «Умения, опыт и мотивации» и
«Инструменты» = «Степень автоматизации процесса разработки».

9. **Влияние политики.** Команды разработчиков и других лиц, заинтересованных в разработке АС, действуют в рамках организаций и предприятий, с их корпоративной системой отношений и организационной политикой, в контексте других политик (отраслевых, государственных, международных). Всё это вно-

сит в проблему сложности дополнительные темы для рассуждений и дополнительные составляющие в АС или её окружение.

10. **Недостаток воображения.**

Достаточно типична ситуация, когда приходится разрабатывать АС, у которой нет аналога. А значит, приходится строить её будущее образное представление, в первую очередь как образец для понимания, за счёт феномена «воображение». У того лица или лиц, которым будет поручена такая работа, их индивидуальных и коллективных возможностей воображения может просто не хватить, что, разумеется, приведёт к проблемам. Следовательно, феномену «воображения» необходимы инструментальные помощники, повышающие результативность и качество работы феномена. В число таких помощников в обязательном порядке входят рассуждения.

Названные факторы сложности и ряд других, приводящих к сложности ПО, названы также в докладе Г. Буча на конференции [14], в котором указан ряд «сил, оказывающих давление» на процесс разработки АС (рис.1.3).

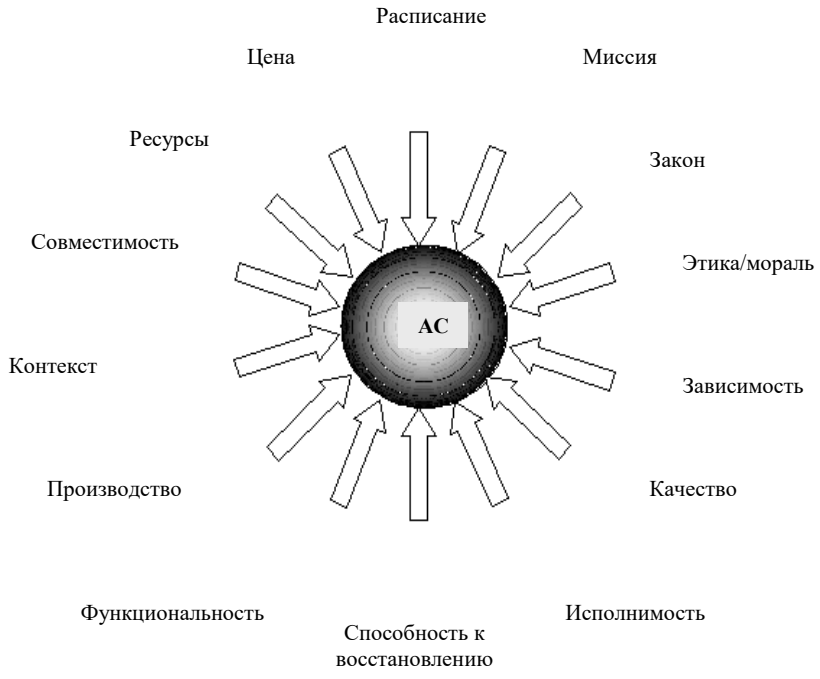


Рис.1.3. «Давление» на процесс разработки АС

В докладе отмечено, что разработчикам АС приходится преодолевать следующие «силовые давления», обусловленные факторами, раскрытыми выше в п.п.1–9:

1. **Давление на процесс** разработки со сторон:

Производства, включающего реализуемость АС (особенно если это новая разработка), управляемость (процесса и продукта) и устойчивость (к возможным воздействиям на части разрабатываемой системы).

Способности к восстановлению ((упругости, эластичности) охватывающей необходимость обеспечения таких характеристик качества, как *поддерживаемость (*supportability*), сопровождаемость (*maintainability*), адаптируемость (*adaptability*), масштабируемость (*scalability), *переносимость между платформами* (portability*), расширяемость (*extensibility*) и предрасположенность системы к изменениям(churn*) так, чтобы её можно было использовать в новых технологиях.

2. **Давление среды**, в составе которого важны:

- *Ресурсы*, включающие такие составляющие, как вычислительные мощности, электрическая энергия, физическое пространство и, возможно, другие персонально ограничивающие факторы.

- *Совместимость* с внешними элементами, недостаточная степень которой может привести к отклонениям от норм (процесса и технических стандартов; требований по комплексированию с другими системами).

- *Контекст* в рамках экономических , исторических и политических факторов, учёт которых носит обязательный характер.

3. **Давление со стороны будущего использования** АС, включающего:

Необходимость достижения запланированных функциональностей АС с позиций поведения системы и её практичности (usability).

Исполнимость (Performance) и характеристики качества с позиций других (а не только уже названных выше) нефункциональных требований к АС.

4. **Давление бизнеса**, включающее *Цену, Расписание (План) и Миссию* (особенно в формах долговременных деятельностных обязательств организации или предприятия перед партнёрами и/или клиентами) относят к числу наиболее важных давлений на разработчиков АС. Они связаны со временем, материалами и людьми, реальное наличие которых определяет возможность или невозможность любой конкретной разработки и её соответствие миссии.

1.4.2. Подходы к структурированию

Сложность ПО обусловлена не только названными выше факторами и давлениями на процесс его разработки и использования. Однако перечисленного уже достаточно для того, чтобы согласиться с необходимостью применения любых средств, позволяющих снизить сложность как конкретной АС, так и процесса её разработки.

В системной и программной инженерии накоплен определённый опыт, позволяющий повысить степень успешности в разработках АС. Наиболее традиционным способом снижения сложности является её структуризация, то есть разделение «сложности» на части, связанные в единое целое. В этом плане «сложность» может выступать в разных формах [12], например:

- в формах описания «сложного образования», что согласовано с метрикой сложности по Колмогорову;

- или в формах совокупности разнообразий состояний «сложного образования», выводящей на энтропийные меры сложности.

Обе приведённые формы пригодны, по крайней мере, для оценок в действиях по уменьшению сложности ПО:

- на уровне кода (исходного или исполняемого) существуют программные объекты, состоящие из десятков и сотен миллионов строк кода;

- ПО допускает автоматную интерпретацию, которая для больших программных объектов приводит к автоматным представлениям с практически бесконечным числом состояний (обойти которые, например, при тестировании, практически невозможно).

В системной инженерии системы принято делить на подсистемы и т. д. до уровня базовых единиц, используемых для структуризации подсистем, которые не содержат других подсистем. В программной инженерии деление кода на части часто проводят в формах «модуляризации» (modularity), использование которой приводит к структурам, элементами которой являются «модули». Для структуризации АС используют как подсистемы, так и модули. Достаточно часто конкретную АС (или её представление) разбивают на подсистемы, а структуризацию каждой из подсистем проводят на уровне модулей, которые понимаются и разрабатываются в соответствии с их определением в объектно-ориентированных языках программирования.

Существуют различные подходы и способы структуризации АС и их программной части [14, 31, 64]. С рядом типовых результатов структуризации связывают образцы архитектурных стилей, наиболее распространённые образцы которых приводится ниже в п. 3.4.

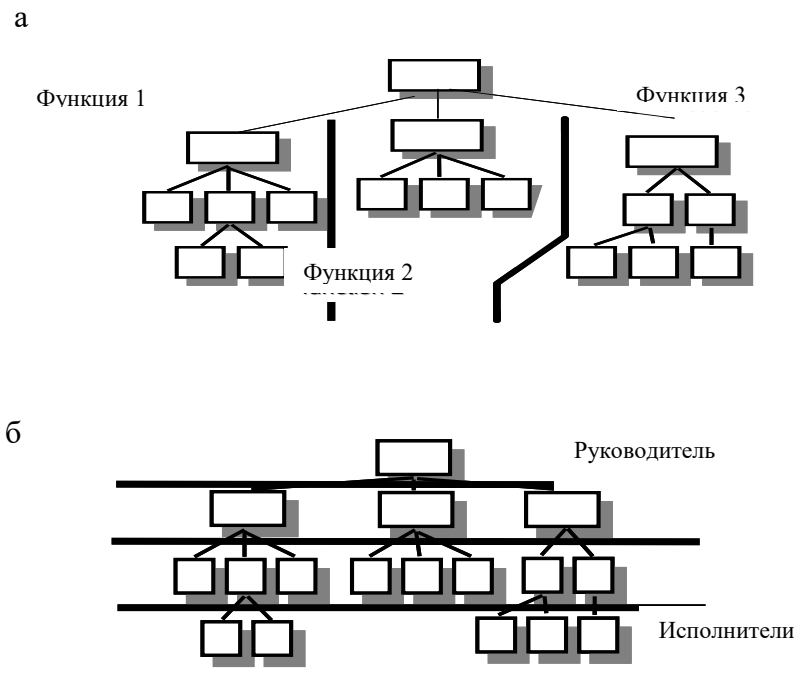


Рис.1.4. Деление на части (а – горизонтальное, б – вертикальное)

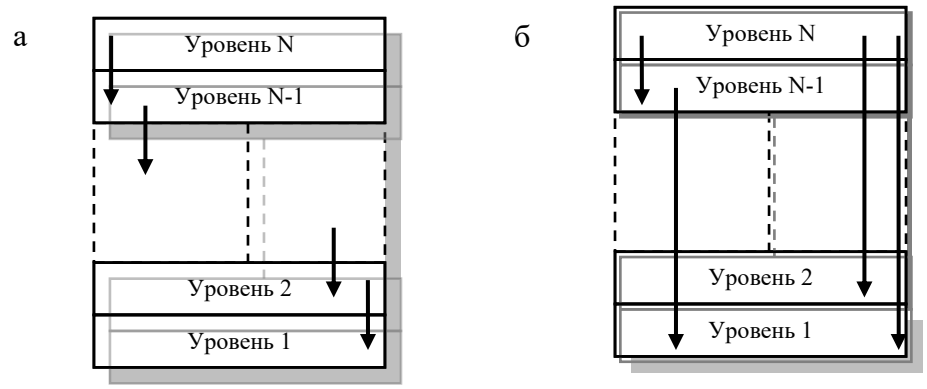


Рис. 1.5. Многослойные структуры (а – закрытого типа, б – открытого типа)

В закрытых многослойных структурах сообщения могут быть посланы только смежному нижнему уровню, в то время как в открытых – сообщения могут быть посланы любому нижнему уровню.

В любом из эскизов структуры разработчикам приходится использовать, а значит, по крайней мере, обобщённо раскрывать связи между элементами структуры. Для АС наиболее типичными вариантами связей являются связи по данным (достаточно часто выход данных одного элемента используется как вход другого), а также по управлению типа call (элемент вызывает активность другого элемента) или типа «событие» (элемент активизируется при определённых событиях, которые происходят в окружающей его среде, в том числе и из-за активность других элементов системы).

Разумеется, представленная структуризация является не единственным способом снижения сложности представления АС. С каждым из названных выше факторов, с каждым «давлением» на разработку связаны определённые подходы и способы. Наиболее общими приёмами для перехода от «сложного» к более «простому» являются:

- **абстрагирование** для отделения существенного для АС (или её части) от несущественного при запланированном или ситуативном взаимодействии со сложным (например, при принятии проектных решений) [12];

- **построение вида** (или совокупности видов) на АС с позиций определённого фактора или «давления» [14, 89];

- **разделение аспекта** (причины взаимодействия со сложным в АС) на составляющие и их распределение (например, метрик практичности) по частям АС [2].

Обобщённо раскроем сущность каждого из приёмов, используя гипотетическое представление АС в базе данных с объектно-ориентированной структуризацией БД ($ОБ_1(a_1, a_1,\ldots,a_i,\ldots, a_I),\ldots,ОБ_j(a_j, a_{j+1},\ldots,a_k,\ldots, a_K),\ldots, ОБ_1(a_n, a_{n+1},\ldots,a_m,\ldots, a_M)$) . Архитектура АС является представлением АС на уровне абстракции, для которого учитываются только существенные атрибуты, причём отражающие АС с позиций её целостности.

Если по таким атрибутам построить целостную проекцию для БД, то в результате будет образована новая (абстрагированная от БД) база данных $БД^A$, в которой:

- часть атрибутов из представления $ОБ_j(a_j, a_{j+1},\ldots,a_k,\ldots, a_K)$ будет исключена;

-некоторые объекты типа $ОБ_j(a_j, a_{j+1},\ldots,a_k,\ldots, a_K)$, возможно, в абстрактное представление АС не попадут;

- некоторые совокупности объектов типа $ОБ_j(a_j, a_{j+1},\ldots,a_k,\ldots, a_K)$ будут объединены в более «крупный» объект.

Такого рода приём можно нацелить на построение проекции АС с точки зрения определённого фактора или интереса к АС, в результате чего в $БД^A$ будут выделены проекции $БД^A_1$, $БД^A_2,\ldots$, $БД^A_r,\ldots$, $БД^A_R$, каждая из которых является целостным видом на АС, то есть её проекцией на определённую точку зрения. Именно эта идея вложена в стандарт IEEE-1471–2000.

Реальность разработок АС такова, что определённые существенные атрибуты (например, $a_i, a_j,\ldots,a_k$) образуют группы, члены которых распределены в группе объектов. То есть из элементов таких групп (обычно отражающих качество АС как целого или качество частей АС) нельзя образовать объект. Именно такая ситуация связана с разделением аспекта на составляющие, ответом на ко-

торую является разработка и использование аспектно-ориентированного проектирования [36] и аспектно-ориентированного программирования [2].

Подводя итог пункту, отметим, что для работы со **сложностью** в системной и программной инженерии накоплен полезный опыт, помогающий разделить сложные (обычно неявные представления систем) на связную совокупность простых представлений. Этот опыт приходится применять в разных состояниях существования АС, в том числе и в тех, когда АС существует как замысел или в виде концепции (или в виде архитектуры или в виде концептуального проекта). Для разработки архитектуры АС такой вид работ определяет её сущность.

1.4.3. Специфика структурирования АС

Для АС как систем, интенсивно использующих программное обеспечение, при структуризации их форм существования (таких как концепция, архитектура, концептуальный проект, …, физическая реализация, …, модифицированная версия) необходимо учитывать специфику АС как систем определённого класса, в первую очередь специфику структуризации ПО.

Практика разработок архитектур АС, представленная в [64], фиксирует целесообразность использования в задачах структуризации следующей специфики:

1. Позитивный опыт использования в ПО модульной структуризации (Module structure).

2. Сложившийся опыт использования программных единиц (компонентов), взаимодействие которых осуществляется (с помощью коннекторов) в компьютерных средах (Component-and-connector structures) .

3. Используемую на практике дополнительность «аппаратного и программного», приводящую к размещение программных структур в физическом пространстве (Allocation structures).

В основе модульной структуризации (рис.1.6) лежит следующее:

– декомпозиция (Decomposition): опирается на отношения между модулями и их связи в рамках иерархии, выделение модулей является стартовой точкой для проектирования АС, модули обычно ассоциируются с продуктами (или ар-

тефактами) процесса разработки, модульная структура согласована с «модифицируемостью» АС;

–использование (Uses) как специальный механизм, определённый в объектно-ориентированном программировании:

– разбиение на уровни (Layered);

– использование классов (Class or generalization), детализирующих модульные структуры с помощью отношений наследования и «экземпляр класса»

Рис.1.6. Специфика модульной структуризации

В основе структуризации «Component-and-Connector» лежит специфика поведения единиц структуры АС или её частей (рис.1.7):

– специфика процесса (Process) или взаимодействия процессов (синхронизация, коммуникация, отработка исключительных ситуаций);

– специфика параллельного исполнения (Concurrency);

– динамическое разделение данных (Shared Data, например, в рамках репозитария);

– специфика клиент-серверных отношений (Client-Server).

Рис.1.7. Структуризация динамики

В основе распределения составляющих АС в физическом пространстве и назначении им форм материализации (рис.1.8) лежит следующее:

1. **Размещение (Deployment):**

– показывает, как ПО назначаются аппаратные средства и средства коммуникации;

– представляет элементы ПО (обычно элементы процесса из вида «компоненты и коннекторы»), аппаратные единицы и коммуникативные связи;

– выражает отношение «allocated-to», показывающее, на какой физической базе элементы ПО размещены;

– предназначено для инженера, для того чтобы он мог рассуждать об исполнении кода, интеграции данных, доступности и защите;

– представляет особый интерес для распределённых параллельных систем.

2. Реализация (Implementation):

– показывает, как элементы ПО (обычно модули) распределены в файловых структурах в процессе разработки АС, интеграции её модулей и управляемого конфигурирования среды;

– представляет особый интерес для управления действиями разработчиков АС и развертывания процесса разработки.

3. Распределение работ (Work assignment):

- распределяет ответственность за реализацию и интеграцию модулей (частей) между членами команды разработчиков.

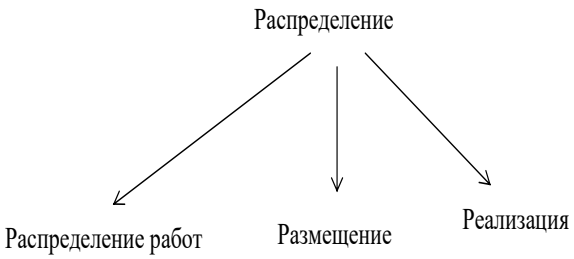


Рис.1.8. Структуризация распределения

Следует отметить, что: архитектурные структуры предоставляют различные перспективы системы: они не являются взаимоисключающими и часто дополняют друг друга; некоторая структура может быть «погружена» в другую; одна из структур может оказаться доминантной; чем сложнее система, тем более полезно представлять её рядом взаимодополняющих структур.

1.5. Место и роль архитектурных решений в разработке АС

1.5.1. Место архитектурных решений

Содержание определений архитектуры ПО явно и неявно указывает на место архитектурных решений и их роль в разработке АС. Место архитектурных решений и документов представим рядом рисунков, в основном заимствованных из публикаций, на основе которых составлено пособие.

В своей презентации доклада «Архитектура ПО» [10] G.Booch указывает на место разработки архитектуры (рис.1.9) в итерационном процессе типа RUP.

Рис.1.9. Итеративный процесс разработки

Подчёркивая особую важность Use-Case моделей в процессе разработки АС, И. Якобсон включает архитектуру в процесс, как показано на рис.1.10.

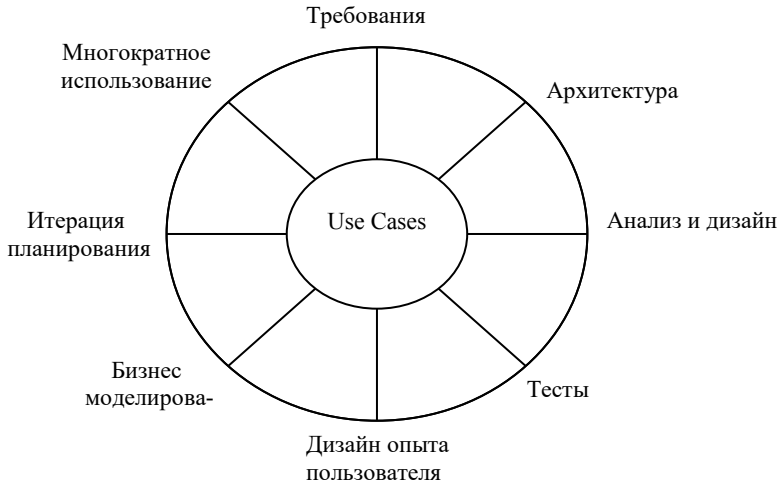


Рис.1.10. Use-case центрированная разработка

В современной практике разработки АС широко используется спирально-итеративная модель жизненного цикла [87], что приводит к необходимости возврата к вопросам разработки архитектуры (рис.1.11) с каждым циклом спирали.

Рис.1.11.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.1.11) типичны и возвраты от вопросов архитектуры к вопросам о требованиях, поскольку нормативы на АО являются очень важным источником требований.

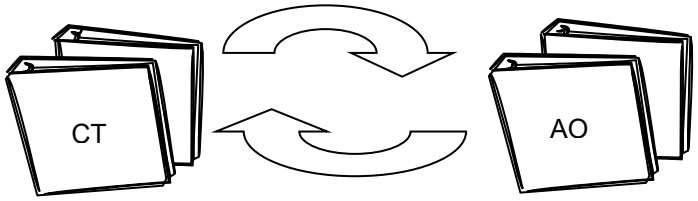


Рис. 1.12.Взаимодействие требований и архитектуры

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.1.13.

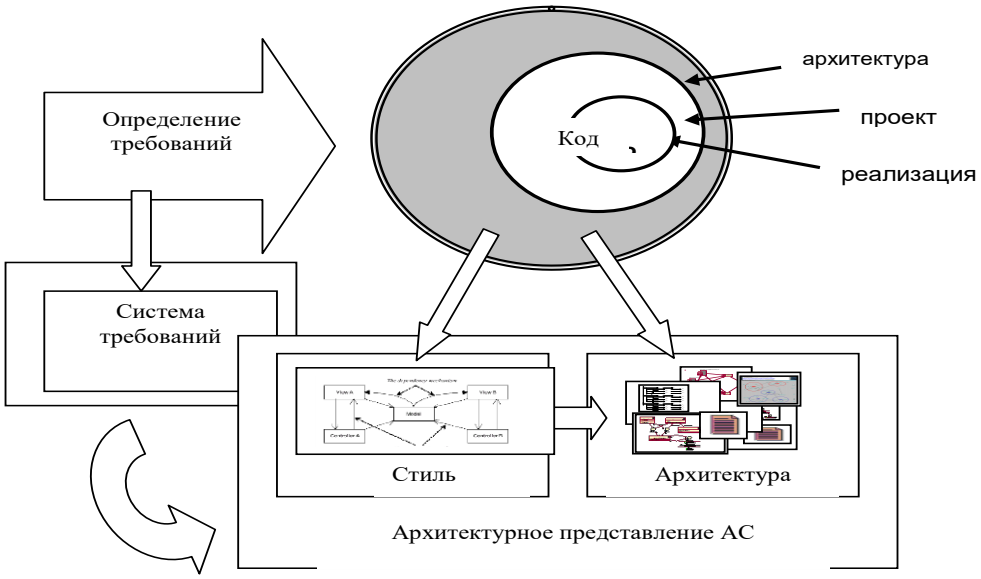


Рис.1.13. Архитектура как форма концептуального существования АС

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 (очень часто использующих для своего представления язык UML).

1.5.2. Роль архитектурных решений

Явное построение и использование архитектуры в разработке АС не раз демонстрировало важные позитивные эффекты [29] . Обобщённое представление позитивных эффектов представлено на рис.1.14.

Рис.1.14. Обобщённое предназначение архитектуры

Более детально, архитектура:

- вносит вклад в извлечение треббований 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.

Именно по этой причине Rational Unified Process построен как архитектурно-центрированный процесс разработки АС [39].

1.6. Ретроспектива развития предметной области

Как уже отмечалось, «архитектура ПО» как определённая предметная область сложилась за последние 15 лет. Историю этих лет, а также предполагаемое будущее представим с помощью публикаций [52, 32, 43] и [62]. Многие считают, что первая из них положила начала архитектуре ПО как отдельной дисциплины в рамках программной инженерии. Вторая является публикацией, констатирующей состояние исследования и разработок по архитектуре приблизительно в середине пройденного пути. А две последних публикации 2006 года (одна из них [43] переведена на русский язык) были представлены в февральском номере IEEE Software с целью раскрыть историю вопроса.

1. Словосочетание «архитектура ПО» в свободном от разъяснений его употреблении или в форме рабочих определений (в различных исследованиях и публикациях) использовалось задолго до публикации, которая, как считают многие, положила начало «архитектуре ПО» (и «архитектуре АС») как отдельной дисциплине в рамках программной инженерии.

В статье была предложена знаменитая формула «{элементы, формы, объяснения} = программная архитектура» и её обобщённые выражения в терминах «архитектурных стилей». Раскрыта целесообразность заимствования из других предметных областей понятия «вида» и наполнения этого понятия содержанием, соответствующим задачам программной инженерии. Были намечены первоочередные шаги для создания предметной области «Архитектура». Эта публикация существенно повысила интерес к исследованиям и разработкам, раскрывающим различные аспекты архитектуры ПО, в том числе и её приложений к архитектурам АС различной степени интеграции, вплоть до архитектуры предприятий.

2. В 2000 году вышла публикация [32], фиксирующая состояние в предметной области «Архитектура» и определявшая «дорожную карту» для исследований и разработок в этой области.

После обобщённого представления истории программной инженерии по схеме рис.1.15 фиксируется современное состояние, в рамках которого выделяются (как успешные) три позиции: языки архитектурного описания, линейки продуктов и архитектурные образцы, в первую очередь образцы архитектурных стилей. Будущее связывается с направлениями в области сетевых приложений и с всеобъемлющем приложении ПО к различным предметным областям.

3. История и современное состояние предметной области «архитектура ПО и АС» раскрыто в публикации [43], которая начинается с представление разделов предметной области:

Архитектурный проект*:* как мы создаем архитектуру?

Анализ*:* как мы на основе архитектуры отвечаем на вопросы о свойствах конечного продукта?

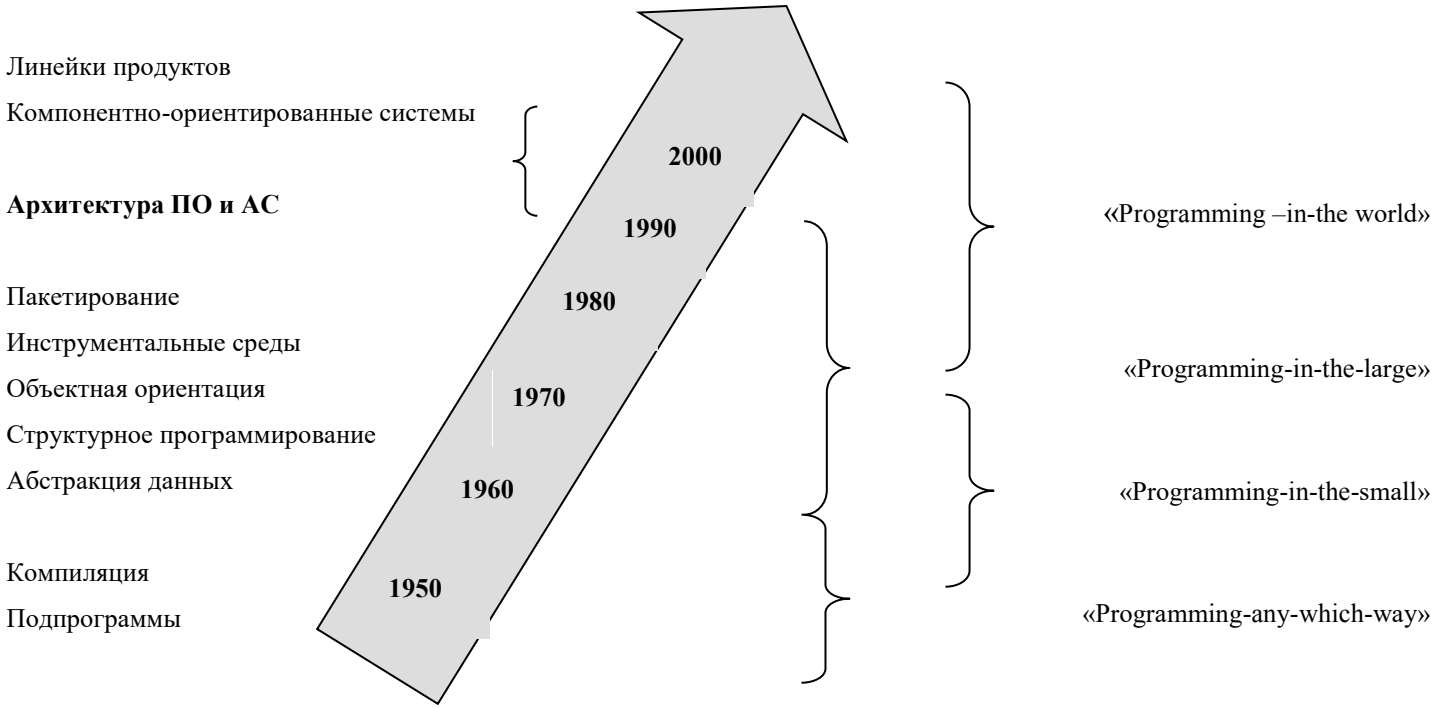


Рис.1.15. Дорожная карта программной инженерии

***Реализация**:* как мы создаем систему на базе архитектурного описания?

***Представление**:* как мы создаем надежные артефакты, чтобы «донести» архитектуру до людей и машин?

***Экономика**:* какие архитектурные проблемы влияют на бизнес-решения?

В истории становления предметной области «Архитектура» авторы выделяют этапы «до 1995 года», «1995-1998 г. г.», «1998-2005 г. г.» и «Сегодня», каждый из которых раскрывается с позиций основных достижений в теории и практике. Такая история обобщённо представлена на рис.1.16, содержащем в том числе публикации, используемые в пособии.

Отмечается, что концепция программной архитектуры как отдельной дисциплины сформировалась в период 1990–1992 годы, завершившийся основополагающей статьей Д. Перри и А. Уолфа [52]. В 1994 году вышла первая книга по программной архитектуре, написанная бывшими сотрудниками IBM Бернардом Виттом, Терри Бейкером и Эвереттом Мерритом [71].

В 1995-м начался настоящий бум программной архитектуры. События ускорились за счет многочисленных «вкладов» отраслевой и академической науки. Заметными явлениями стали метод анализа программной архитектуры SAAM (Software Architecture Analysis Method — первый из серии методов, предложенных SEI [6]), подходы на базе нескольких представлений (такие как модель представления «4+1» компании Rational [41] и четыре представления Siemens), а также специальные шаблоны для разработки программной архитектуры [9, 30]. Корпорации Siemens , Nokia , Philips , Nortel, Lockheed Martin, IBM и другие крупные разработчики программного обеспечения начали уделять внимание программной архитектуре.

В 1999 году состоялась первая конференция по программной архитектуре, были основаны рабочая группа IFIP Working Group 2.10 и институт Worldwide Institute of Software Architects. Появились и сформировались методы SAAM, BAPO и ATAM, а к уже имевшемуся общему стандарту архитектуры RM-ODP [88] добавился IEEE 1471 [89]. Одновременно в SEI продолжали выпускать книгу за книгой [5,18,19].

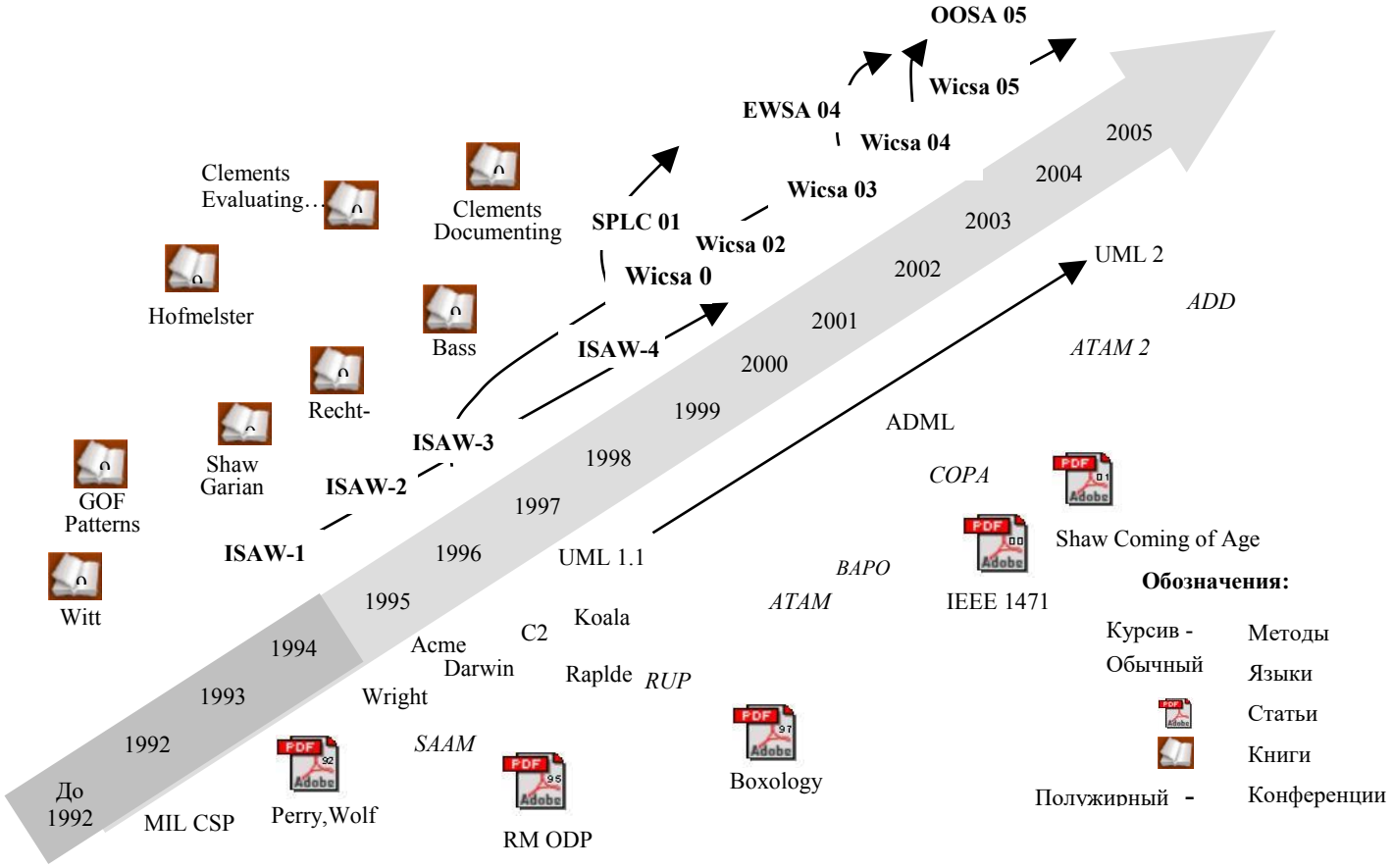


Рис.1.16. Ретроспектива проблемной области «Архитектура АС»

Современное состояние предметной области «Архитектура» авторы раскрывают с разных позиций, которые представлены и в пособии. Фиксируется, что сумма знаний о программной архитектуре изложена более чем в 25 монографиях и многочисленных научных статьях. Сформировалось активное сообщество специалистов, десятки университетов по всему миру преподают программную архитектуру, множество организаций предлагают курсы подготовки архитекторов (представлен обширный список публикаций, сайтов профессиональных сообществ, указателей на конференции по проблематике предметной области). Одним словом, дисциплина не только подтвердила своё право на существование, но и достигла достаточной научно-технической зрелости.

4. Ещё одна версия истории развития предметной области «Архитектура» представлена в публикации M. Show и [61]. Авторами выбран подход к вопросу с позиций увеличения степени зрелости этой области. За образец взята схема становления зрелости, которую часто повторяют новые технологии.

Такая схема обычно развёртывается на 15-летнем интервале и включает этапы:

1. Базовое исследование
2. Формулировка концепции
3. Развитие/расширение
4. Внутреннее расширение/исследование
5. Внешнее расширение/исследование
6. Популяризация

Становление зрелости проблемной области «Архитектура АС и ПО» проходило именно по такой схеме, причём так, что позволило авторам назвать то, что стоит за схемой становления, «Золотым веком Архитектуры».

Вопросы по первой главе

Q1. Какие системы относятся к классу «автоматизированных систем, интенсивно использующих программное обеспечение»?

Системы автоматизации проектирования.
Системы управления предприятиями.
Автоматизированные обучающие системы
Системы коллективного принятия решений.

Q2. Что представляет собой архитектура АС?

Концептуальную форму существования АС

Средство представления структуры АС

Средство выражения поведения АС

Q3. В каких интеллектуальных действиях при разработке АС используются архитектурные представления?

Рассуждения в актах принятия проектных решений.

Контроля в формах понимания за проектными решениями.

Согласования принятых решений.

Q4. Какие факторы давления на процесс разработки можно отнести к факторам среды?

Контекст

Исполнимость

Расписание

Способность к восстановлению

Q5. Какие факторы давления на процесс разработки можно отнести к факторам бизнеса?

Ресурсы

Закон

Цена

Качество

Q6. Какие факторы давления на процесс разработки можно отнести к факторам будущего использования?

Миссия

Зависимость

Функциональность

Совместимость

Q7. Чем характеризуется структуризация АС посредством многослойных структур закрытого типа?

Части одного уровня обмениваться между собой сообщениями и посылать их смежному верхнему уровню

Сообщения могут быть посланы любому нижнему уровню

Уровни закрыты друг от друга и действуют как автономные системы

Сообщения могут быть отправлены только смежному нижнему уровню

Q8. Что лежит в основе модульной структуризации?

Размещение, которое показывает как ПО назначаются аппаратные средства и средства коммуникации

Специфика параллельного исполнения

Декомпозиция

Q9. Что лежит в основе структуризации "Component-and-Connector"?

Динамическое разделение данных

Использование классов

Специфика клиент-серверных отношений

Q10. Что лежит в основе структуризации распределения?

Разбиение на уровни

Распределение работ

Специфика процесса или взаимодействия процессов

Q11. Какое место занимает разработка архитектуры в жизненном цикле АС?

Следует за этапом формирования системы требований.

Предшествует этапу детального проектирования.

Входит в состав концептуального проектирования.

Q12. Какой этап следует за Разработкой при спирально-итеративной модели жизненного цикла?

Анализ

Реализация

Тестирование

Q13. Какова роль архитектурных решений?

Вносит вклад в извлечение тре6бований и формирование их системы;

Предписывает организационную структуру АС

Помогает в эволюционном прототипировании;

Может быть базисом для изучения системы.

Q14. К каким позитивным эффектам приводит использование архитектуры?

Повышает степень успешности разработок АС

Снижает риски.

Оформляет концептуальную целостность АС и процесса её разработки.

Обслуживает понимание и взаимопонимание в индивидуальной и коллективной работе.

ГЛАВА ВТОРАЯ. АРХИТЕКТУРНЫЕ НОРМАТИВЫ

2.1. Архитектурные образцы

Как уже отмечалось в п. 1.3, архитектура является формой существования АС в виде её концептуального представления, регистрирующего и обслуживающего понятийный контроль за рассуждениями лиц, вовлечённых в процессы разработки и использования АС.

Процесс контроля на понимание можно организовать в различных версиях, в том числе и в версиях сравнения индивидуального и коллективного понимания с **образцами понимания**.

Образцы для контроля создают в различных областях деятельности, как для контроля действий, так и для контроля их результатов. Такие образцы, разумеется, зависят от того, что с их помощью контролируется. В тех случаях, когда контролируются естественно-профессиональные рассуждения, функции образцов понимания принято возлагать:

- на средства логической формализации рассуждений, использующей их перевод на подходящий искусственный язык и автоматический или автоматизированный контроль формул перевода;

- на результаты «перевода» рассуждений в образные конструкции (схемы, «картины»), выражающие целостное содержание сказанного в рассуждениях;

- на толкование рассуждений с помощью их построений в другой версии знакового выражения и/или с помощью примеров.

В первом варианте контроля рассуждений достаточно много сложностей, в первую очередь обусловленных разнообразием тематик, в рамках которых приходится рассуждать об АС, а значит, и разнообразием логических формальных средств, возможно и очень сложных. Использование единообразных формальных средств, например, языков архитектурного описания, по крайней мере пока, не находит широкого применения в практике разработок АС.

В практике разработок АС для контроля рассуждений в основном используются образные конструкции, в построениях которых используют языки визуального моделирования, причём, очень часто язык UML.

Третий вариант контроля в большей мере годится для оперативного контроля фрагментов рассуждений. Так, например, одной из проверок фрагмента рассуждений на правильность, является их проверка на корректное употребление понятий проекта АС. Примеры в проверках по третьему варианту выполняют функции целостных образцов, но частного характера.

Рассмотрим детали второго варианта контроля как наиболее распространённого на практике. Одной из принципиальных характеристик визуальной образной конструкции, обслуживающей процессы понимания, является её целостность, согласованная с определённой темой или набором родственных тем для рассуждений, сопровождающих разработку АС.

В искусственном интеллекте с типовыми темами связывают понятие фрейма (frame). Если типовая тема сопровождает определённую работу (work), то её логично называть «framework». Фреймы, а значит и frameworks, принято представлять семантическими структурами, состоящими из узлов и связей между ними. Узлы фрейма соответствуют определённым «понятиям», а связи – отношениям между «понятиями».

Концептуальные образования типа «framework» являются типовым нормативным средством (нормативными концептуальными схемами) для представления архитектур АС. Широкое применение нормативных архитектурных концептуальных схем (architecture frameworks) обусловлено следующими причинами:

- во-первых, любая architecture framework (как схема) является целостным концептуально-образным образцом, согласованным с рассуждениями, которые, на определённом уровне абстракции (на мета уровне описания архитектуры, которому соответствует нормативная «картина» архитектуры), применяют при построении и использовании архитектуры конкретной АС;

- во-вторых, любая architecture framework управляет построениями экземпляра архитектуры как средства, регистрирующего понимание и взаимопонимание, согласованное в коллективной работе (в оговоренных рамках проще договориться об общем понимании, то есть проще достичь консенсуса);

- в-третьих, architecture framework и построенная на её базе конкретная архитектура являются системой концептуально-образных образцов, с которыми следует сверять последующие (после построения архитектуры) рассуждения, разумеется, в рамках тем, вложенных в архитектуру.

Разработаны и применяются на практике различные схемы типа architecture framework. Это, чаще всего, обусловлено различиями предметных областей (например, разработки корпорации Microsoft существенно отличаются от АС, выполняемых по государственным заказам для военных), а для родственных предметных областей корпоративностью или традициями. Наиболее известные по литературе или по их практическому применению схемы типа architecture framework, рассматриваются ниже.

2.2. Стандарт IEEE-1471–2000 и его сущность

2.2.1. Основные понятия

В статье [52], которая относится к числу базовых фундаментальных публикаций, выделивших в программной инженерии «архитектуру ПО» как отдельную и важную проблемную область, была указана необходимость использования в архитектурных представлениях механизма «видов» на ПО, подобных «видам», применяемым в инженерной практике (например, в строительстве). Эта идея получила развитие, что привело к стандарту IEEE 1471-2000 Recommended Practice for Architectural Description of Software-Intensive Systems [89] (Рекомендуемое для практики описание архитектуры систем, интенсивно использующих ПО).

В стандарте представлены обзор его области применимости (границы, цели, предполагаемые пользователи и соглашения по рекомендуемой практике), ссылки на родственные стандарты, концептуальный каркас (контекст архитектурного описания, заинтересованные в разработке АС лица, архитектурная деятельность в рамках жизненного цикла АС, использование архитектурных описаний), практика архитектурных описаний (документирование архитектуры, идентификация заинтересованных лиц и их интересов, селекция архитектурных

точек зрения, архитектурные виды, соответствие в совокупностях архитектурных видов, обоснования архитектур, образцы использования) и ряд приложений (библиография, замечания по терминологии, образцы точек зрения, отношения с другими стандартами).

Стандарт позиционирован его разработчиком (IEEE Architecture Planning Group) как «рекомендуемая практика». Он не является стандартом архитектуры, процесса архитектуры или метода разработки архитектуры, а предоставляет разработчикам АС рекомендации для описания архитектуры (Architectural Description, AD). Его рекомендации используют модальности «должен» (без обязательности) и «можно» (как вариант). Но рекомендации базируются на большом опыте успешных разработок АС. Рекомендации приняты мировыми лидерами индустрии ПО и АС.

Стандарт IEEE 1471 [89]:

- нацелен на его применение для архитектурного описания (АО) систем, интенсивно использующих программное обеспечение, в которых программная составляющая оказывает существенное влияние на проектирование, конструирование, развертывание и эволюцию системы как целого;

- предполагает, что организации и предприятия, решившие использовать стандарт, должны сами решать, в каком объёме и как он будет внедряться в разработки АС;

- дает широкие рамки интерпретации архитектуры, как средства, пригодного для процесса разработки АС;

- исходит из того, что описание архитектуры может быть согласовано широким кругом лиц, вовлечённых в разработку АС, в то время как система, проект, процесс и организация работ такой возможности обычно не предоставляют (из-за специфики разнородной профессиональной компетенции и языка);

- определяет концептуальную структуру (систему понятий) для описания архитектуры и его использования ;

- устанавливает термины и понятия для архитектурного мышления;

- устанавливает концептуальный каркас и словарь для рассуждений об архитектурной форме существования АС, в виде архитектурного описания;

- обеспечивает средства для рассуждений об архитектурном описании в контексте лиц, вовлечённых в процесс разработки АС, жизненного цикла АС и использования АО;

- служит как базис для развития АС на ранних этапах разработки, когда доступна только общая терминология об АС, которая ещё не материализована;

- вводит в системы требований для разработки АС специальный раздел «архитектурных требований»;

- идентифицирует и устанавливает тождество в описаниях архитектур и пропагандирует озвученное в архитектурной практике;

- открывает возможность для эволюции таких практик, как достигших достаточной инструментально-технологической зрелости;

- служит как базис для развития предметной области, в которой существует только общая терминология;

- вносит надежду разработчикам АС, что следование его рекомендациям внесёт в разработку такие позитивы деятельности как «быстрее», «лучше» и «дешевле».

2.2.2. Содержание стандарта

Перейдем к основе и деталям рекомендуемой практики, обслуживающей построение архитектурных описаний, документирующих архитектуры.

Основу рекомендуемой практики определяет концептуальный каркас (framework), представленный на рис.2.1 и раскрывающий то, что целесообразно включать в АО. Концептуальный каркас в стандарте оформлен в виде семантической сети, связывающей определёнными отношениями основные понятия архитектуры АС. В пояснениях к каркасу не используются спецификации для форматов описания и/или других средств, обеспечивающих построение АО. За отбор подходящих средств и их использование несёт ответственность архитектор.

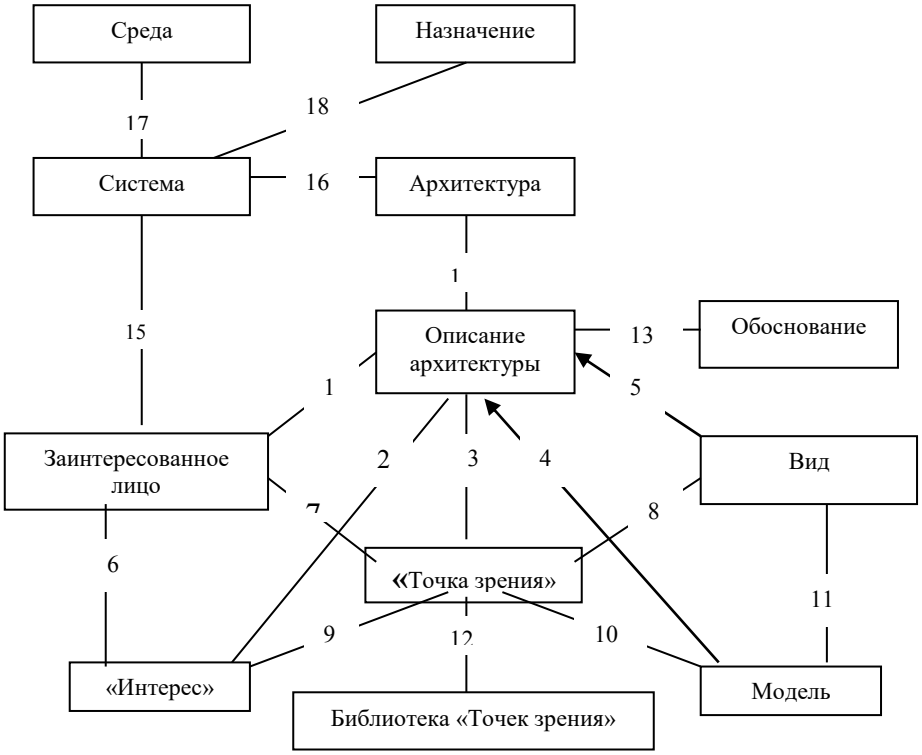


Рис.2.1.Стандартное описание архитектуры (отношения: 1 – различается, 2 – различается, 3 – выбирается, 4 – аггрегируется, 5 – состоит из видов, 6 – имеет, 7 – адресуется, 8 – реализуется, 9 – покрывает, 10 – устанавливает форму, 11 – состоит из моделей, 12 – содержит, 13 – убеждает, 14 – представляет, 15 – использует, 16 – имеет, 17 – рзмебщена, 18 – обеспечивает)

В рамках концептуального каркаса указаны понятия (и их связи), относящиеся к понятию «архитектурное описание» («заинтересованные лица», «интерес заинтересованных лиц», «вид», «точка зрения» и другие детали) и контексту его употребления (система, среда, миссия, архитектура). За каждым понятием стоит его (потенциальное или реальное) материальное воплощение определённой составляющей (или ряда составляющих) процесса и/или результата разработки АС. Факт отсутствия такого материального воплощения указывает на отклоне-

ние от схемы, которое (как показывает практика разработок АС) может привести к негативным последствиям разработки.

Отметим, что на рис.2.1 указаны связи между узлами, но не приведены (чтобы не загромождать рисунок) характеристики степени связи, то есть кратности «один к одному (1..1)», «один ко многим» (1..N) и др.. Степень связи легко установить, проведя оценочный анализ.

Выборочно представим содержание узлов схемы. Толкование ряда особо важных понятий каркаса (архитектурной концептуальной схемы АС) проведём с позиций тех требований, которые они выражают в системе требований к АС:

1. Различные значения понятия «Лица, заинтересованные в разработке АС. Stakeholders» вводятся для того, чтобы на множестве таких лиц определить типовые роли, исполнение которых осуществляют, в общем случае, группы лиц. Необходимо всегда помнить и принимать в расчёт, что архитектура разрабатывается для отмеченных лиц, архитектура должна отражать их требования и интересы, адекватное определение группы повышает шансы на успех разработки.

К типовым ролям относятся :

- лица (Acquirers), приобретающие систему;

- эксперты-консультанты (Assesors), проверяющие целесообразность приобретения;

- пропагандисты (Communicators), распространяющие сведения об этом через документы и обучение;

- разработчики (Developers), создающие систему;

- сопровождающие (Mainteners), внедряющие, сопровождающие и развивающие систему;

- снабженцы (Suppliers), поставляющие «комплектующие части»;

- пользователи (Users), обеспечивающие использование системы по назначению;

- администраторы (Administrators), обеспечивающие функционирование системы;

- тестировщики (Testers), проверяющие корректность работы системы.

В публикациях и конкретных инструментально-технологических системах разработки АС называют и используют не только названные роли, но и многие другие. Так, например, в RUP различается около 70 ролей stackeholders.

Определение группы и конкретных лиц выполнено правильно, если эти лица информированы (что позволяет им принимать хорошие решения), способны принять на себя обязательства и ответственность (даже если решения трудные) и авторитетные (что повышает степень доверия к их решениям).

2. За понятием «Интерес (Concern)» стоит то, что принято называть интересом лица, находящегося в определённой роли к процессу разработки АС, например, «интерес к тем функциям, которые исполнимы в АС, а значит к поведению АС», «интерес к защищённости АС по отношению к несанкционированному доступу» или «интерес к позитивным эффектам, достижимым при использовании АС». В общем случае конкретный «интерес» может выражать определённую заинтересованность группы лиц, исполняющих типовую роль, например, групп «разработчики» или «пользователи».

Выявить совокупность таких «интересов» и их значений, необходимых и достаточных для успешной разработки АС, является одной из первоочередных и важнейших задач концептуального проектирования АС, ответственного за «систему требований», «архитектуру» и «концептуальный проект».

3. Понятие «Точка зрения. Viewpoint» раскрывает позицию, с которой определённая группа лиц или группа групп, заинтересованных в разработке АС, желает, привыкла, готова или в состоянии представить АС как целое. С определённой «Точкой зрения» в архитектурном описании связывают, в общем случае, интегрированную совокупность родственных «Интересов», например,

4. Понятие «Вид. View» является родственным для «Видов», используемых в машиностроительном или строительном черчении. Примером совокупности видов, используемых в других приложениях, служит набор схем, представленных на рис.2.2.

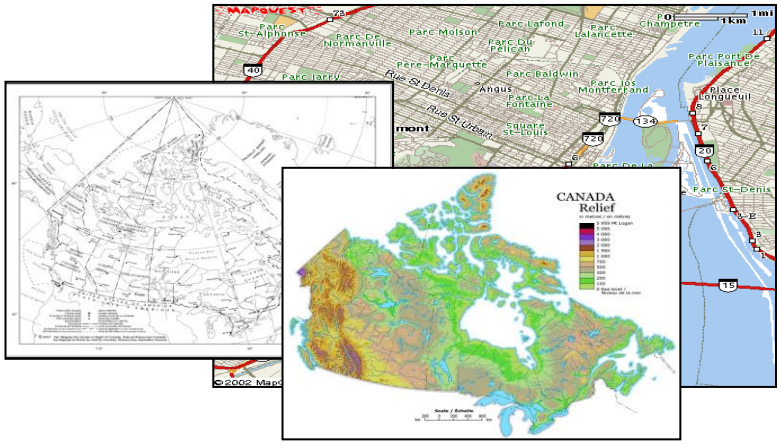


Рис.2.2. Набор картографических видов

Понятие «вид» в архитектурных описаниях АС соответствует «проекции АС» на определённый интерес или интересы. Для представления архитектурного вида используют, в общем случае, совокупность концептуальных моделей и документов, раскрывающих их содержание или дополняющих содержание, выраженное в моделях. Архитектурные виды АС принято визуализировать в форме, подобной визуализации, представленной на рис.2.3.

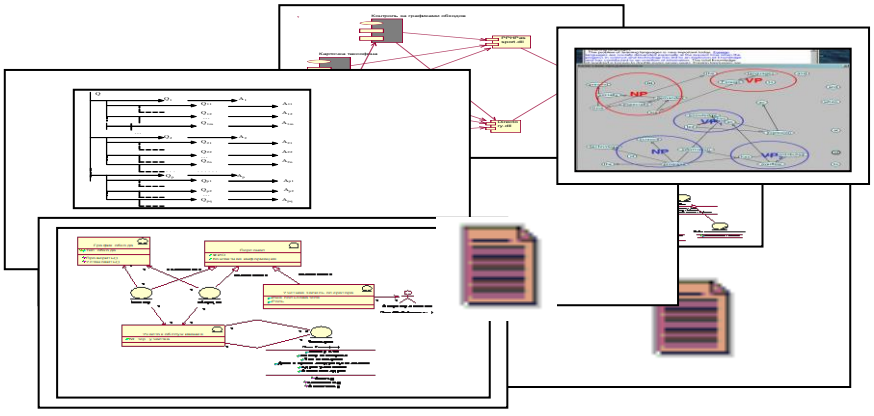


Рис.2.3. Визуализированный набор архитектурных видов

Виды в конкретной технологии разработки АС оформлены на определённом языке (например, на языке UML и/или других языках описания архитектуры, Architecture Description Language) и визуализированы в соответствии с определённым набором правил: каждый вид соответствует вполне определённой точке зрения (рис. 2.4); точка зрения является образцом для конструирования видов, причём точка зрения определяет правила не только для построения видов, но и для их использования.

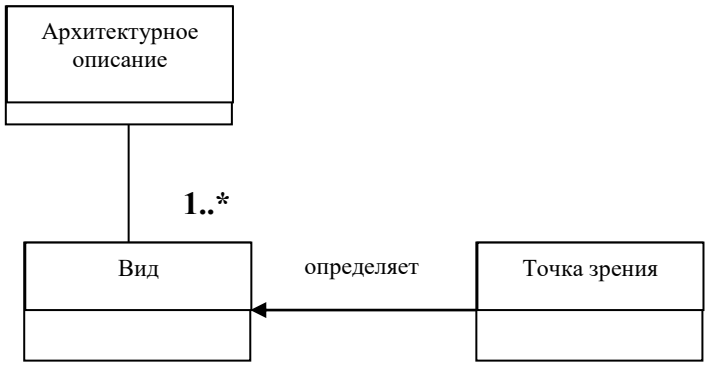


Рис.2.4.Фрагмент архитектурной семантической сети

Стандарт не фиксирует определённый набор видов, поскольку число разнородных АС и их приложений многообразно. В рамках вопроса об архитектурах АС не может быть окончательно решён вопрос об окончательном и полном наборе точек зрения и видов. В то же время точка зрения должна быть объявлена до её использования.

Виды в конкретной технологии разработки АС хорошо оформлены: каждый вид соответствует вполне определённой точке зрения; точка зрения является образцом для конструирования видов; точка зрения определяет правила не только для построения видов, но и для их использования.

Стандарт не фиксирует определённый набор видов, не может быть окончательно решён вопрос, откуда ещё могут прийти точки зрения, точка зрения должна быть объявлена до её использования.

Отношения между «точкой зрения», «интересами» {ci} «видами» «моделями» {mj} и документами {Dk}, обобщённо представленные на рис.2.5, завершают толкование основных понятий стандарта IEEE-1471–2000.

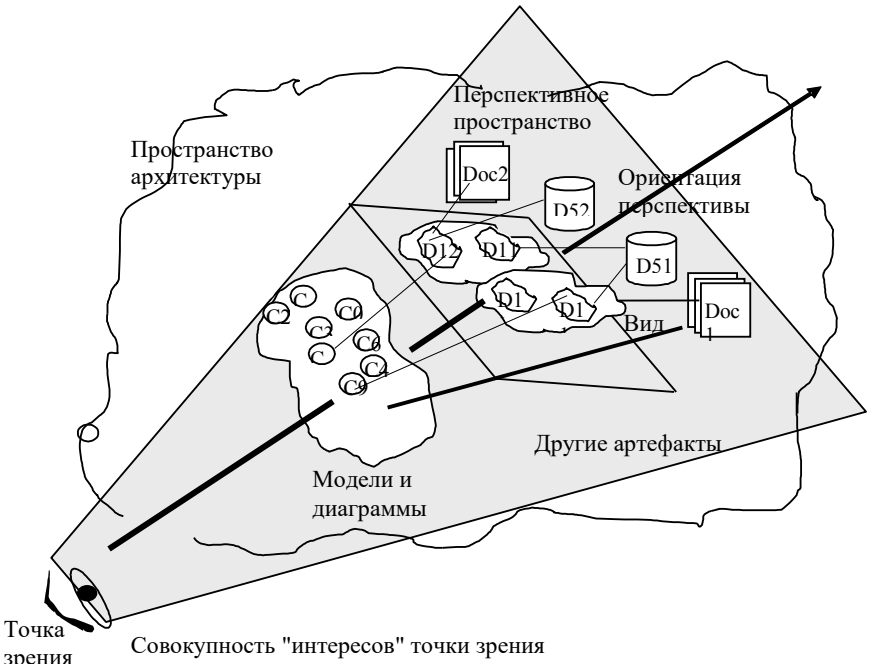


Рис.2.5. Образное представление отношений

Намерение использовать варианты употребления понятий стандарта IEEE-1471 в архитектурном описании предписывает определиться с их значениями, или, что то же самое, ввести архитектурные требования в систему требований АС и специфицировать эти требования. Следовательно, необходимо определиться:

- с «типовыми группами лиц», интересы которых следует или целесообразно учесть в разработке АС;

- с «интересами» типовых групп, провести их анализ и систематизировать;

- с «точками зрения» (возможно, выбрав подходящие в библиотеке «точек зрения»), на основании которых будет строиться архитектурное описание АС;

- с совокупностями «видов» на АС (возможно, выбрав подходящие в библиотеке «видов»), которые будут составлять архитектурное описание.

Решения, принятые относительно этих требований, определят самое важное для действий по проектированию архитектуры АС, в результате которого архитектурное описание наполнится деталями, в первую очередь подходящими концептуальными моделями и документами.

Для отмеченной работы с архитектурными требованиями полезны специальные табличные шаблоны (табличные образцы). Так, например, для объявления «точки зрения» можно использовать шаблон (таблица 2.1), указывающий на необходимость действий для заполнения его содержанием атрибутов.

Состав атрибутики и её значений для всех конструктов архитектурного описания, а не только для «точек зрения» должен быть таким, чтобы он позволял решать задачи на базе АО, в частности задачу выбора подходящих образцов для последующих (за этапом построения архитектуры) этапов жизненного цикла разработки АС.

Таблица 2.1.

Атрибут	**Значение**
Имя «точки зрения»	
Тип группы	
Интересы лиц	
Язык описания	
……	
Критерии целостности	
Критерии оценки	
Эвристики	
Образцы	
Руководства	

Стандарт IEEE 1471 отмечает необходимость использования архитектуры системы для решения таких задач, как следующие:

1. Анализ альтернативных проектов системы.
2. Планирование перепроектирования системы, внесения изменений в её организацию.
3. Общение по поводу системы между различными организациями, вовлеченными в её разработку, эксплуатацию, сопровождение, приобретающими систему или продающими ее.
4. Выработка критериев приёмки системы при её сдаче в эксплуатацию.
5. Разработка документации по её использованию и сопровождению, включая обучающие и маркетинговые материалы.
6. Проектирование и разработка отдельных элементов системы.
7. Сопровождение, эксплуатация, управление конфигурациями и внесение изменений и поправок.
8. Планирование бюджета и использования других ресурсов в проектах, связанных с разработкой, сопровождением или эксплуатацией системы.
9. Проведение обзоров, анализ и оценка качества системы.

2.2.3. Представления схемы IEEE-1471

Стандарт IEEE-1471 находит широкое применение на практике, что привело к его осмысливанию и переосмысливанию с различных позиций и применений. В таких действиях из семантического каркас стандарта либо извлекают часть сети, либо дополняют её или часть дополнительными фрагментами, либо строят родственные семантические сети.

На рис.2.6 представлено извлечение из семантической сети стандарта его наиболее существенной части, в которой раскрываются отношения между основными понятиями описания архитектуры. Фрагмент (рис.2.6) имеет более простой вид, удобный для практического использования в оперативных действиях при разработке архитектуры, для формирования документации на архитектуру и при изучении стандарта.

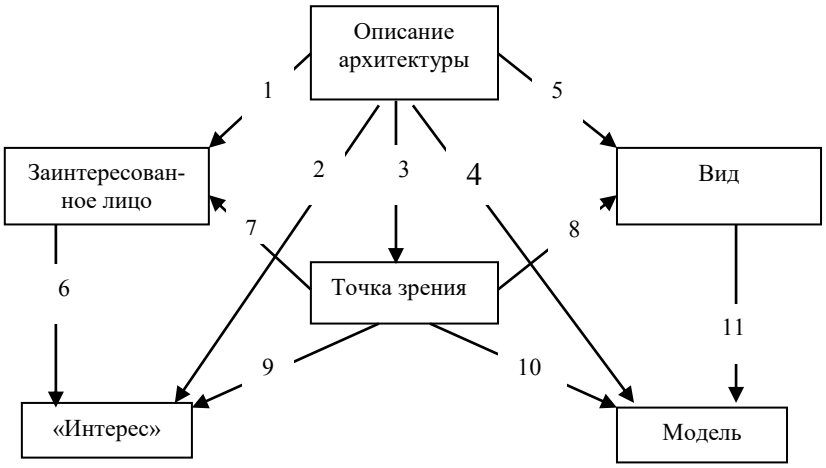


Рис.2.6. Фрагмент семантической сети IEEE-1471

В ряде публикаций и докладов G. Booch [10,14] связал со стандартом ряд мета моделей и рекомендовал их использовать в архитектурной практике. Одна из таких моделей представлена на рис.2.7. Число узлов и связей этой модели (по отношению к базовой концептуальной схеме IEEE-1471) расширено.

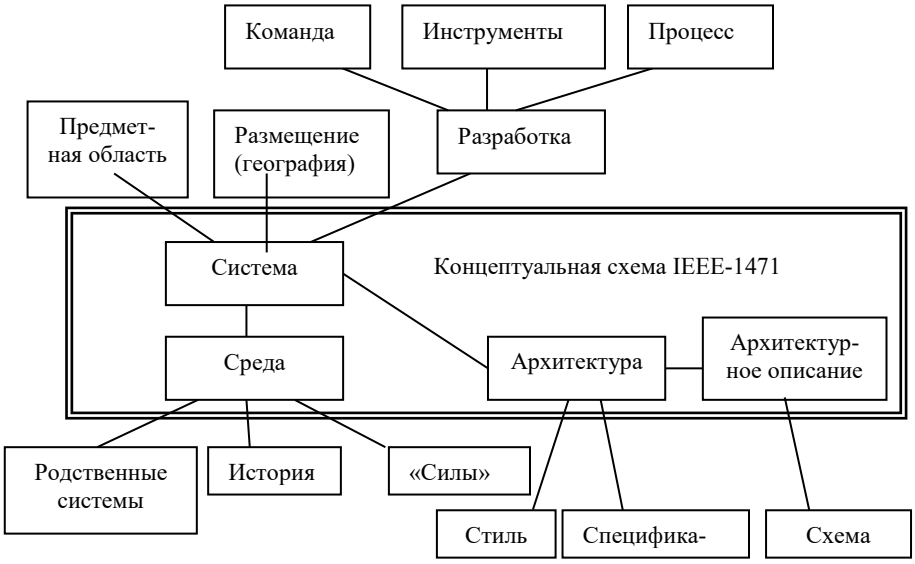


Рис.2.7.Расширенная семантическая сеть

Дополнительные узлы и связи, размещённые за рамками (двойная линия) схемы IEEE-1471, связаны с определёнными узлами схемы IEEE-1471. Добавлены:

- для узла «система» её отношение с «предметной областью», к которой она принадлежит, пространственное «размещение» системы и её «разработка»;

- узел «разработка» детализирован до «команды», проводящей разработку, «процесса» её деятельности и применяемых «инструментов»;

- в узел «архитектура» как составляющая добавлен «стиль» (в общем случае интегрированная совокупность стилей) и отражён тот факт, что архитектура имеет определённые «спецификации»;

- для «архитектурного описания» показано, что оно осуществляется по определённой «схеме» или, что то же самое, согласно определённому шаблону (template);

- по узлу «среда» отмечена целесообразность учёта «родственных систем», её «истории» и «силовых давлений», в том их смысле, который был раскрыт в п.1.3.

2.3. Архитектурные концептуальные схемы

2.3.1. Определение и ретроспектива

Как уже отмечалось выше, стандарт IEEE-1471является концептуальной схемой, по образцу которой рекомендуется описывать архитектуру АС. У этого стандарта были предшественники, а он сам использовался при построении других подобных конструктов. Таким образом, в практике архитектурных описаний АС разного типа сложился специфический класс «Архитектурных концептуальных схем» (Architecture Frameworks, AF) , принадлежность к которому конкретной «Концептуальной схемы» определяется по наличию у схемы вполне определённого набора признаков.

По сложившемуся пониманию конструкты AF являются обобщёнными прагматически руководствами:

- для описания архитектур, предоставляющими возможности не только для представления архитектур, но и для сопоставления описаний в совокупности архитектурных альтернатив;

- которые определяют системы видов и моделей, их целесообразно использовать при построении архитектуры, в том числе и для того, чтобы разделять, понимать и сравнивать архитектурную информацию;

- которые подсказывают, какими способностями архитектор/проектировщик должен овладеть, и как эти способности должны быть использованы в процессах построения архитектуры и её использования.

К числу особо важных составляющих AF относятся:

- «Виды», как средства для понимания, взаимопонимания и коммуникативного взаимодействия лиц, заинтересованных в разработке АС;

- «Методы», предоставляющие дисциплину, для того чтобы собирать и организовывать данные и конструкты «видов» способом , помогающим гарантировать целостность, точность и завершённость архитектурного представления АС;

- «Обучение/ опыт», что обеспечивает разумное и конструктивное применение методов и обслуживающих их инструментов.

2.3.2. Архитектурная концептуальная схема Дж. Захмана

Особое место среди архитектурных концептуальных образцов занимает концептуальная схема Дж. Захмана [35], представленная на рис. 2.8. Схема изначально нацеливалась на системное представление задач информатизации, в которых приходится учитывать специфику АС.

С момента своего создания (1987 год) схема Дж. Захмана постоянно усовершенствовалась, оставаясь определённого рода «вопросником» о существенных аспектах АС. Схема построена в форме табличной структуры, для формирования и использования которой необходимо построить систему ответов, каждый из которых заполняет определённую позицию таблицы. Ответ может быть как простым, так и сложным. Одной из наиболее распространённых форм ответов является визуализируемая концептуальная модель.

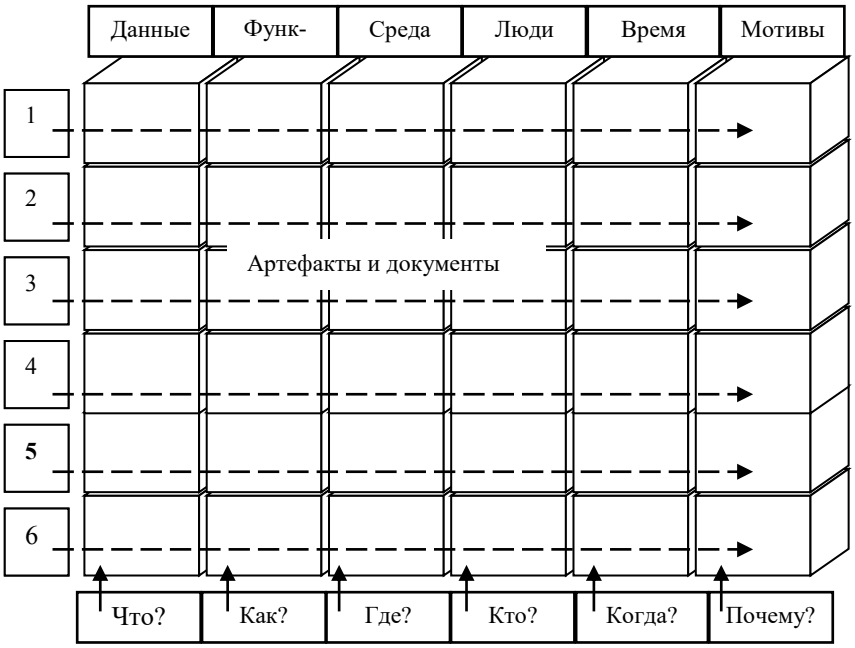


Рис.2.8. Архитектурная схема Дж. Захмана

То содержание, которое вкладывается в каждый ряд и каждый столбец схемы, представлено в таблицах 2.2 и 2.3 .

Таблица 2.2.

Ряд схемы	**Содержание**
1. Цели и границы (Вид планировщиков)	Определяет основные цели и границы, в рамках которых должны приниматься архитектурные решения
2. Модель предприятия, системы (Вид владельцев, ответственных за систему)	Определяет сущность предприятия (системы), включая структуры, процессы и организационные структуры
3. Модель базового понимания (информационного представления, Вид архитектора)	Определяет в более строгих терминах содержание ряда 2

4. Технологическая модель (Вид проектировщиков)	Определяет, как и какая технология должна быть использована для того, чтобы запросы третьего ряда были удовлетворены
5. Детализированное представление (Вид разработчика)	Определяет детали проектирования, применяемые языковые средства, структуры данных и другие средства
6. Функциональности системы (Вид пользователей)	Определяет, как должны работать системы в рамках предприятия (в контексте окружения)

Таблица 2.3.

Столбец	**Содержание**
1. Данные (Структура, Что?)	Фокусирует внимание на сущностях, объектах, компонентах и отношениях между ними
2. Функции (Активность, Как?)	Фокусирует внимание на той деятельности, которая осуществляется организацией и поддерживается автоматизированными системами
3. Сеть работ (Месторасположение, Где?)	Фокусирует внимание на географическом (физическом) распределении активностей
4. Люди (Кто?)	Фокусирует внимание на тех лицах, которые вовлечены в бизнес-процессы
5. Время (Когда?)	Фокусирует внимание на эффектах, связанных со временем (планирование, события)
6. Мотивации (Почему?)	Фокусирует внимание на целях, стратегиях и ограничениях, специфических для реализации системы

Использование этой схемы предполагает, что лица, отвечающие за архитектуру системы, в том числе и типа АС, заполнят «клетки» подходящим содержанием. Разумеется, содержание «клеток» включает спецификации (на уровне архитектуры) тех решений, которые по этой «клетке» приняты. В результате будет образована целостная архитектурная картина представляемой системы,

например АС, которая может быть использована в решении большинства из названных выше (п.2.3.1) задач.

Заметим, что Дж. Захман разрабатывал свою схему для предприятия, использующего любые информационные технологии, но схема находит широкое применение в теории и практике АС. Так, например, в работе [35] предложено развитие схемы Дж. Захмана до трёх измерений (рис.2.9), за счёт перехода к многослойной структуре системы моделей.

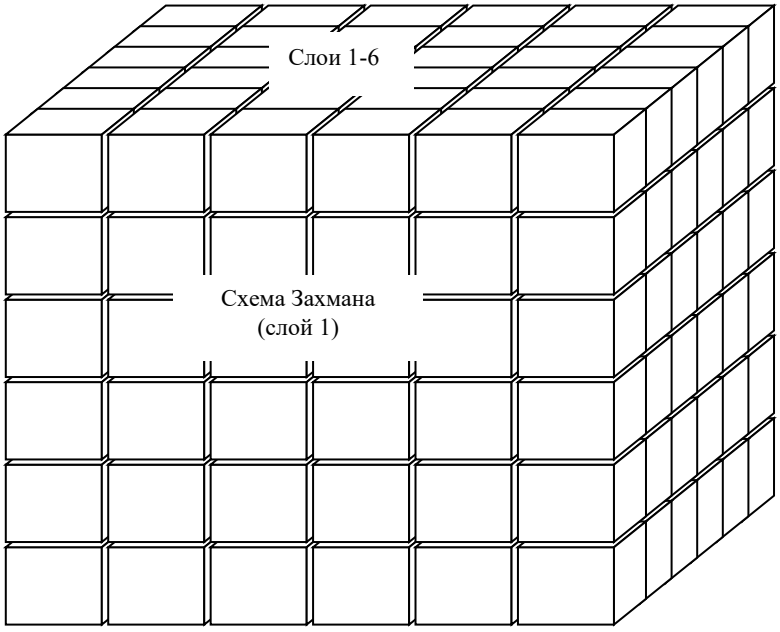


Рис.2.9. «Трёхмерная» схема Захмана

В трёхмерную структуру введены следующие дополнения, детализирующие содержание артефактов, которые приписываются «клеткам схемы Дж. Захмана»:

– слой разработки, раскрывающий «Что?» должно быть сделано для каждого артефакта, приписанного к «клетке» схемы Дж. Захмана;

– слой метаданных, включающий описание информации, которая должна быть собрана и «Как?» она должна быть организована;

– слой документирования, раскрывающий типовые формы документов, в которых (Где?) будет зафиксирована информация;

– слой ответственности, фиксирующий, «Кто?» из команды разработчиков будет исполнять требуемые работы;

– слой жизненного цикла, показывающий, «Где?» в процессе разработки будет создан соответствующий артефакт;

– слой целей, описывающий, «Почему?» следует включать артефакт в состав архитектурного описания.

Для предлагаемого расширения автором трехслойной модели разработаны детали для каждого слоя (и для каждой клетки слоя).

2.3.3. Архитектурная концептуальная схема DoDAF

Эволюция архитектурной концептуальной схемы DoDAF (Department of Defense Architecture Framework), созданной по заказу Департамента обороны США [77], учитывает всё новое, что появляется в теории и практике предметной области «Архитектура АС».

Архитектурное описание АС в DoDAF строится на базе модели данных ядра архитектуры (Core Architecture Data Model, CADM) и артефактов архитектуры. CADM представляет собой стандартизованную систему понятий для определения видов и их составляющих в базе данных.

Вид понимается традиционно и представляет определённый объем операций, систем и технических стандартов. Каждый вид является хорошо определённым множеством элементов данных в составе CADM. Архитектурные артефакты документируются с использованием языка UML.

Виды объединены в четыре группы. Первая группа называется «Все виды» (All Views) и содержит обзор, обобщения и интегрированный словарь по архитектуре DoDAF. Остальные группы – это «Операционные виды» (*Operational Views*), «Системные виды» (System Views) и «Технические виды» (*Technical Views*). Обобщённая картина видов представлена на рис.2.10.

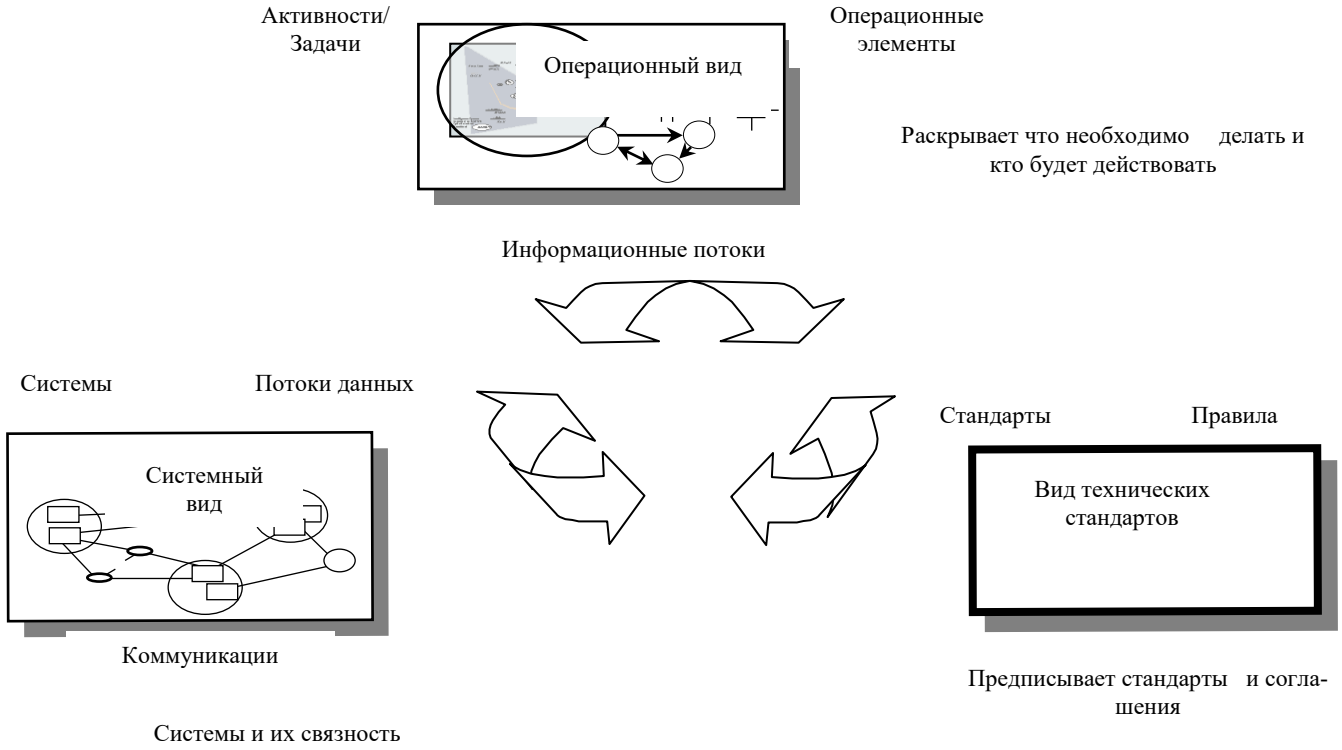


Рис.2.10. Архитектурная схема DoDAF (52 вида)

Операционные виды описывают специфику деятельности и операции, узлы операций, узлы соединений, информационный обмен, организационные отношения, правила операций, последовательности событий и логическую модель данных. Описания, содержащиеся в этой группе, распределены по 24 подвидам. (каждый подвид является видом по определению IEEE-1471).

Системные виды описывают систему и её компоненты, системные интерфейсы, коммуникативное взаимодействие, матрицу отношений на множестве систем и подсистем, функциональности, матрицу связности (traceability), эволюцию, использование и технологические предсказания. Группа состоит из подвидов.

Технические виды (группа из 12 подвидов) описывают текущий стандартный профиль и предсказания по его изменениям.

Архитектурная концептуальная схема DoDAF разработана для военных нужд и в этом плане она является предметно-зависимой. Следует отметить, что DoDAF предоставляет ограниченные возможности по моделировании конфигурации ПО и моделированию нефункциональных требований.

Ближайшим родственником архитектурной схемы DoDAF является схема MoDAF, разработанная европейскими союзниками США. В число основных потребностей, приводящих к отличиям схемы MoDAF от схемы DoDAF, входят следующие потребности:

1. Потребность решать и моделировать задачи приобретения разного рода комплектующих аппаратного и программного типов.
2. Потребность моделировать трансформационные программы и их взаимозависимости.
3. Потребность моделировать компетентность и другие виды способностей лиц, вовлечённых в разработку и использование АС.
4. Потребность моделировать необходимые для разработки АС кадровые ресурсы, подобно моделированию технических ресурсов.
5. Потребность объединять информационные представления в традиционные модели архитектуры, чтобы обеспечить необходимой информацией работу архитекторов предприятия.

Средства DoDAF согласованы с концептуальной схемой Захмана. Объём пересечений между схемами (отмеченный овалами) отражён на Рис.2.11. В версиях DoDAF указывается, что они согласованы со стандартом IEEE-1471.

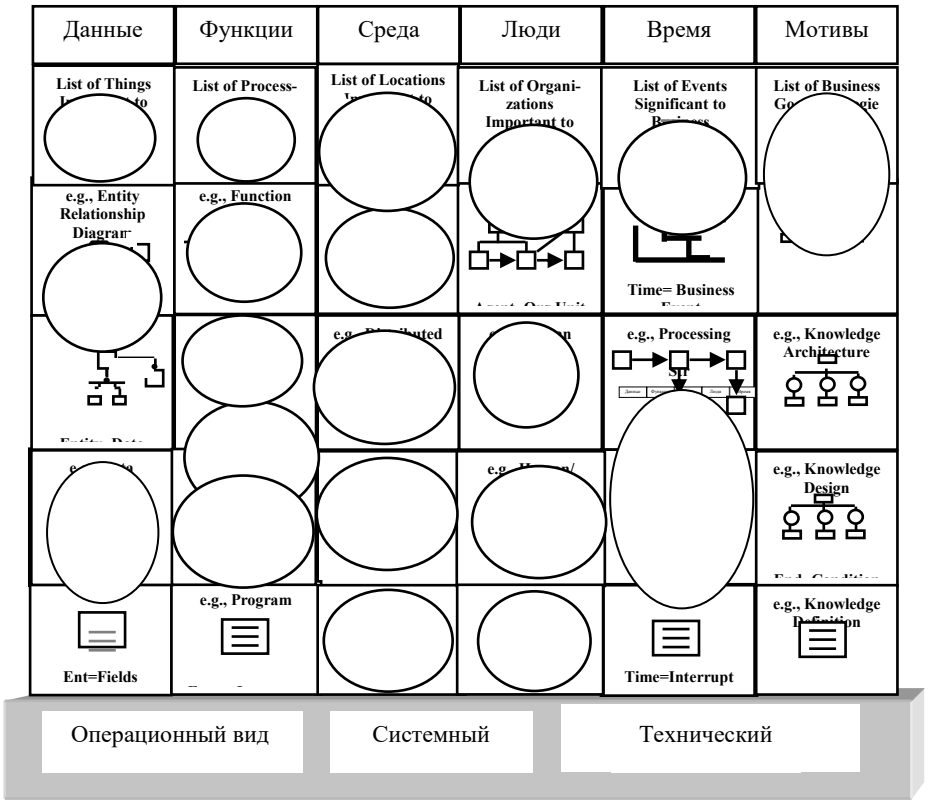


Рис.2.11. Проекция системы DoDAF на схему Дж. Захмана

2.3.4. Архитектурная концептуальная схема TOGAF

С момента её первого опубликования в 1995 году архитектурная концептуальная схема TOGAF (The Open Group Architecture Framework) развивается и используется. В настоящий момент времени доступна её версия TOGAF 8.1.1.

Сущность схемы TOGAF хорошо отражает жизненный цикл «Архитектурного описания АС» (рис.2.12), построенный в соответствии с рекомендациями этой схемы.

Рис.2.12. Жизненный цикл архитектуры АС по TOGAF

За каждым этапом (обозначены овалами) жизненного цикла стоит проверенная методика и руководство по использованию методики. Детальное описание концептуальной схемы TOGAF, включающее описание каждой из методик, представлено на сайте разработчиков TOGAF [80].

Применение TOGAF осуществляется как итеративный процесс. Итерации TOGAF (которые в руководствах называются циклами) обычно отличаются большей длительностью, чем итерации RUP, и объединяют несколько фаз реализации и обслуживания архитектуры предприятия. Это обусловлено тем, что область архитектуры предприятия шире области одного проекта RUP.

2.3.5.Архитектурная схема FEAF

История схемы FEAF (Federal Enterprise Architecture Framework) началась в 1998 году. Её специфика (рис.2.13) связана с её предназначением – разработка АС в рамках системы задач государственного масштаба для США. Подобные задачи в нашей стране намечено решать в рамках «Электронной России». Последняя версия FEAF доступна по адресу www.eaframeworks.com/**FEAF**.

Федеральная Архитектура – это концептуальная модель описания в координированной, структурированной форме деятельности федерального правительства и государственных организаций с функциональной точки зрения, вне зависимости от организационных структур, реализующих соответствующие функции, с целью улучшения их деятельности за счет использования информационных технологий.

По сути дела, это новый способ описания, анализа и улучшения деятельности государства и госорганизаций, а также расширения их возможностей по обслуживанию граждан. Федеральная Архитектура – это стратегический информационный актив, который определяет функции (бизнес) государственных организаций, информацию и технологии, необходимые для реализации функций, а также процессы преобразований, необходимые для внедрения новых информационных технологий в ответ на изменяющиеся бизнес-потребности. Отдельные государственные организации должны использовать эту общую модель для описания своих собственных архитектур.

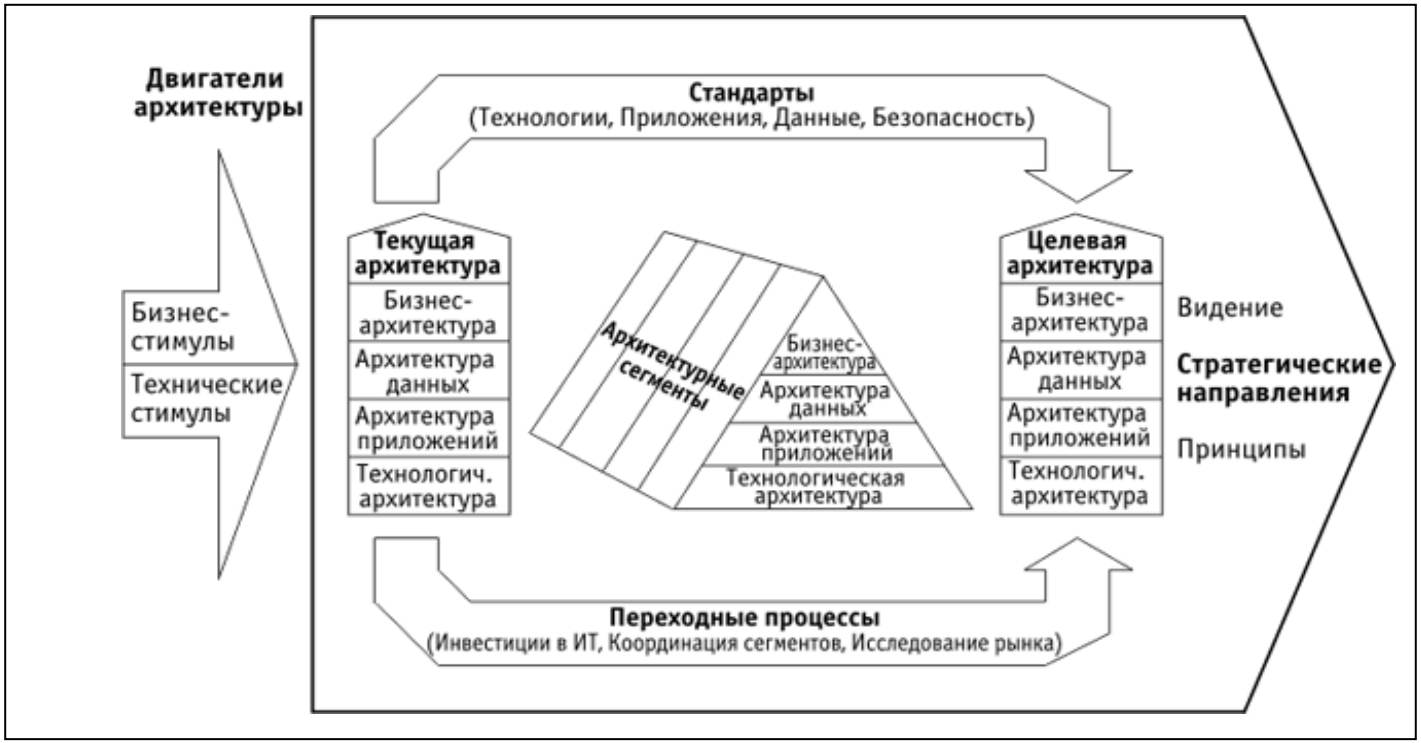


Рис.2.13. Архитектурная схема FEAF

2.4. Сравнительное сопоставление систем архитектурных видов

2.4.1. Проблема стандартной концептуальной схемы

Представленные выше архитектурные концептуальные схемы указывают на то, что организации и группы создают и поддерживают эволюционное развитие «собственных» понятийных схем. Одним из объяснений этого служит разнообразие предметных областей АС, каждая из которых имеет специфику, которую и стараются учесть в «собственной » концептуальной схеме. Ещё одной причиной, возможно, является состояние теории и практики архитектуры АС, которые ещё не достигли состояния, в котором можно было бы создать единую архитектурную концептуальную схему, детализирующую, например, стандарт IEEE-1471 до типологии видов, моделей и документов. К решению такой задачи можно продвинуться через сопоставление различных концептуальных схем и различных систем видов. Два примера таких сопоставлений приведены ниже.

2.4.2. Сопоставление систем видов

2.4.2.1. Архитектура «4+1»

Архитектура конкретной АС описывается множеством ее **представлений. В** архитектуре АС фиксируются главные решения проектирования структуры, показывающие, как приложение, вложенное в АС, разделено на **компоненты,** и как связаны эти компоненты для получения полезной **конфигурации.** Эти решения должны вытекать из **требований** функциональности и других факторов. С другой стороны, эти решения устанавливают дальнейшие **ограничения** на требования и на будущие проектные решения нижнего уровня.

При использовании подхода «4+1» разработчики АС начинают формирование архитектуры с типичного набора видов, представленного на рис.2.14.

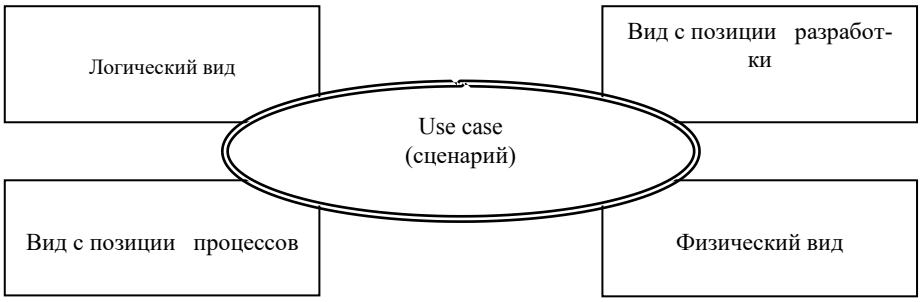


Рис. 2.14.Модель «4+1»

Набор видов включает:

1. Use case вид , который содержит прецеденты и сценарии, охватывающие архитектурно существенное поведение, классы или технические риски. Вид является подмножеством модели прецедентов.

2. Логический вид, который содержит наиболее важные классы проекта, их организацию в пакеты и подсистемы различных уровней. Здесь содержится уже некоторая реализация прецедента. Это представление является подмножеством модели проекта

3. Вид с позиции разработки, который содержит краткий обзор модели выполнения в терминах модулей пакетов и уровней. Здесь описывается также распределение пакетов и классов (из логического представления) в пакетах и модулях представления выполнения. Это представление является подмножеством модели выполнения.

4. Вид с позиций процесса, содержащий описание задач (процессов и нитей), их взаимодействия и конфигурации и распределения объектов и классов по задачам. Потребность этого представления возникает, только если система имеет существенную степень параллелизма. В Rational Unified Process 5.1 представление процесса – это подмножество модели проекта.

5. Физический вид, который содержит описание различных физических узлов для наиболее типичных конфигураций платформы, и расположение задач (из представления процесса) по физическим узлам. Потребность этого представления возникает, только если система распределена.

Дополнительные представления, например для представления интерфейса пользователя, представления защиты и представление данных не отвергаются, но ответственность за их связывание с базой «4+1» ложится на разработчиков АС.

Перечисленные выше представления могут изобразить проект системы в целом. Но архитектура интересуется только некоторыми определенными аспектами:

Структура модели – организационные элементы, например, иерархическое представление.

Существенные элементы – критические прецеденты, главные классы, общие механизмы и так далее, а не все элементы, представленные в модели.

Несколько **ключевых сценариев,** показывающих главный поток управления в системе.

Сервис, чтобы зафиксировать модульность, необязательные особенности, аспекты производственной линии.

*2.4.2.2. **Архитектурные решения SEI***

В Институте Программной Инженерии (SEI) разработан подход к формированию архитектуры, исходящий из того, что для конкретной АС, кроме базовых «точек зрения», может потребоваться дополнительный набор видов. А значит, архитектору должны быть предоставлены возможности создания необходимых ему типовых видов, в частности «box and line» средства.

Предлагается комплект средств выражения и методик для такой работы. Но главный акцент предлагается направить на работу с видами с позиций качества (рис.2.15). Значения характеристик качества должны оцениваться и, если значения не соответствуют требованиям, то в построении архитектуры долен происходить «возврат» к предшествующим шагам архитектурного моделирования для внесения коррекций в архитектуру или даже для принятия новых архитектурных решений.

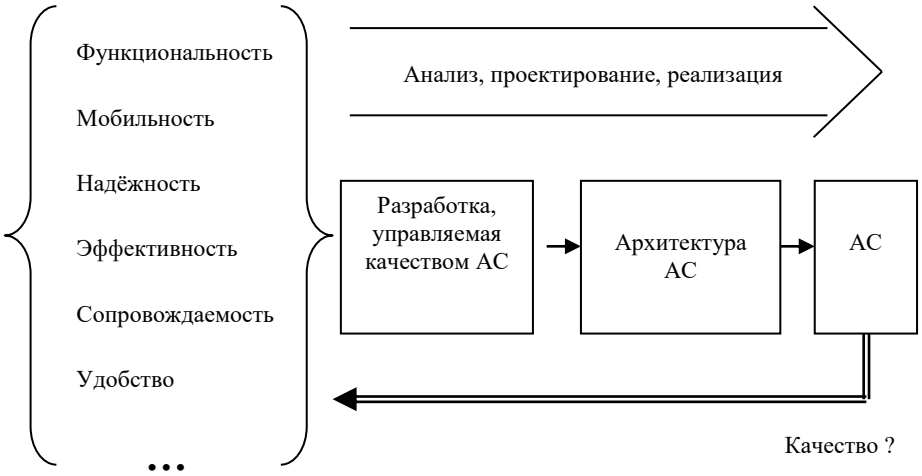


Рис.2.15. Архитектура, управляемая качеством АС

Архитектурные решения SEI согласованы со стандартом ИСО/МЭК 9126. Особое внимание рекомендуется уделять рассуждениям в группе разработчиков и документированию решений.

*2.4.2.3. **Архитектурные решения RM ODP***

Архитектурные решения RM-ODP [88] предоставляют пять точек зрения (рис.2.16) на систему:

– организационная точка зрения описывает постановку цели, области применения, способы и правила применения;

– информационная точка зрения описывает выражение (проявление) и семантику обрабатываемых системой данных, форматы и модели данных;

– компонентная (вычислительная) точка зрения описывает разделение приложения на функциональные модули и определяет их интерфейсы;

– инженерная точка зрения представляет распределение отдельных элементов системы по физическим ресурсам, а также их связи;

– технологическая точка зрения описывает технологии, используемые при реализации системы.

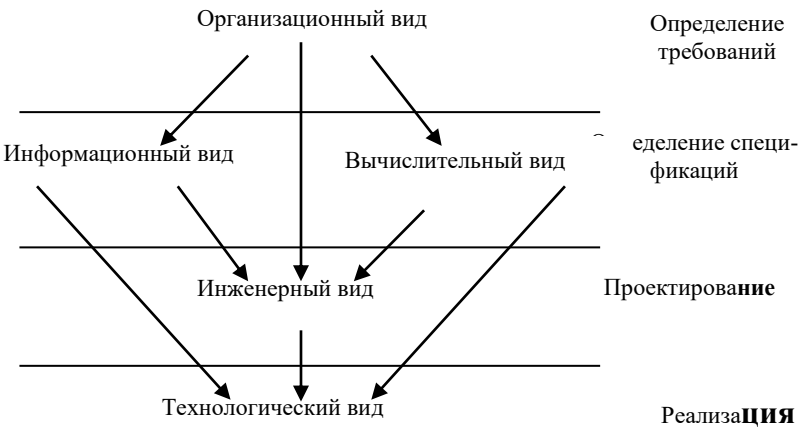


Рис.2.16.Архитектура RM ODP

При помощи этих пяти точек зрения могут быть описаны как существующие системы, так и разработаны новые АС и приложения. Стандарт содержит лингвистические средства для описания видов и систему правил для переходов между видами.

2.4.2.4. Архитектурные решения SIMENS

В архитектурных решениях Siemens (рис.2.17) использование «концептуального вида» в системе «видов» было предложено до создания языка UML, в котором необходимость визуального концептуального моделирования в разработках АС была переведена на уровень стандарта.

Система видов включала:

– концептуальный вид как описание архитектуры системы в понятиях, раскрывающих основные элементы АС и связи между ними;

– модульный вид, раскрывающий функциональную декомпозицию АС и её распределение по уровням детализации;

– вид исполнения, специфицирующий динамическую структуру АС;

– вид с позиций кодов, включающий кодовые компоненты и их библиотечную организацию в среде разработки.

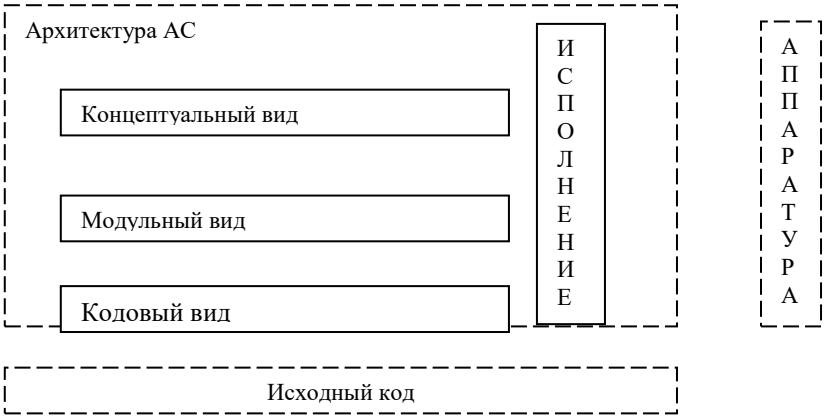


Рис.2.17.Архитектурные решения Siemens

*2.4.2.5.**Архитектурные решения ADS***

Подход «Спецификации рационального описания архитектуры (Rational ADS)» развивает подход «4+1» до его применений, включающих автоматизированные производственные системы, встроенные системы и другие системы, в которых может отсутствовать программное обеспечение. Вариант расширения представлен на рис.2.18.

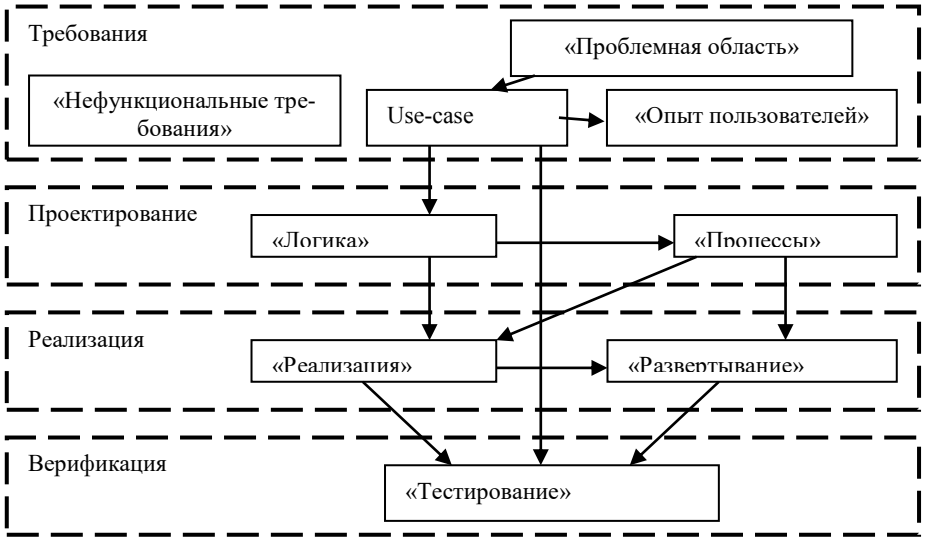


Рис.2.18. Архитектура SAD

В архитектуре виды распределены между четырьмя «точками зрения» (Требования, Проектирование, Реализация и Верификация). Содержание видов соответствует содержанию видов, рассмотренных выше, или понятно из названия. Архитектура нашла своё воплощение в инструментально-технологической среде RUP. Она согласована со стандартами ISO/MEK-12207 и IEEE-1471.

2.4.2.6. Сопоставление образцов архитектур

Перейдём к сопоставлению представленных архитектурных образцов, исходя из возможностей заимствования архитектурных решений для разработок АС. В сопоставлениях ограничимся «возможностью поддержки активности основных ролей, исполняемых членами группы разработчиков» и «наличием средств выражения основных «интересов». Результаты сопоставлений приведены в таблице 2.4 и таблице 2.5.

Таблица 2.4.

Образцы / Роли	**4+1**	**SEI**	**RM-ODP**	**Siemens**	**Rational ADS**
Количество точек зрения			5	4	4 / 9
Архитектор			0	4	3
Проектировщик			0	0	4
Кодировщик			5	0	3
Интегратор			0	0	1
Тестировщик			0	0	1
Специалист по качеству			0	0	0.5
Специалист по управлению			0	0	1
Пользователь			0	0	3
Специалист по руководствам			5	0	3

Таблица 2.5.

Образцы / Роли	«4+1»	SEI	RM-ODP	Siemens	Rational ADS
Количество точек зрения			5	4	4 / 9
Функциональность			3	2	3
Мобильность			1	2	2
Надёжность			2	1	2
Эффективность			1	2	3
Сопровождаемость			1	1	3
Удобство			0	0	2
Безопасность			0	0	1

Сопоставление указывает на то, что общая позиция по архитектурным представлениям АС ещё не устоялась. Системы архитектурных норм при разработках очередных АС, особенно такого типа, как КСА (Комплексы Средств Автоматизации), целесообразно пересматривать, опираясь на текущий опыт архитектурных решений в области программной инженерии.

Выбирая систему архитектурных норм для очередного проекта КСА, следует исходить из взаимодополнительности образцов, доказавших свою полезность на практике. Функции таких образцов способны выполнить образцы, представленные выше.

2.5 .Сопоставление концептуальных схем

Ещё одним примером сопоставления архитектурных концептуальных систем может служить работа [69], в которой (из-за разнородности систем) проводится сопоставление по трём фундаментальным позициям: «цели», «вход» и «выход-результат».

Если конкретная архитектурная система явно поддерживает содержательный элемент строки, то этот факт фиксируется символом «+». Если содержание строки явно не поддерживается или это не отмечается в документации, то в таблице используется символом «-». Частичная явная поддержка отмечается символами «+-». Степень поддержки в таблице не представлена.

Таблица 2.6.

Характеристики	ZF	4+1	FEAF	RM-OD+-	TOGAF	DoDAF
Цели						
Определение архитектуры и её понбимание	+-	+-	+	+	+	+
Процесс архитектурного моделирования	-	-	+	-	+	+
Поддержка эволюции архитектуры	-	-	+	+-	+	+
Анализ архитектуры	+	+	+	+	+	+
Архитектурные модели	+	+	+	+	+	+
Альтернативы проектирования	+-	+-	+-	+-	+-	+
Обоснование проектирования	+-	+-	+-	+	+	+-
Стандартизация	-	-	+-	+	+	+
База знаний архитектуры	-	-	+	+	+	+
Верифицируемость архитектуры	-	+-	-	+-	+	-
Входы						
Методики	+-	+-	+	+-	+	+
Включенность в технологии	-	-	+	+-	+	+
Функциональные требования	+	+	+	+	+	+
Информационная поддержка	+-	+-	+	+	+	+
Текущая архитектура	+-	+	+	+	+	+
Нефункциональные требования	+-	+	+-	+	+	+-
Результаты						
Модель предметной области	+	+-	+	+	+	+
Модель системы	+	+	+	+	+	+
Информационная модель	+	+	+	+	+	+
Вычислительная модель	+	+	+	+	+	+
Модель конфигурации ПО	-	+	-	+-	+	-
Модель процесса	+	+	+	+	+	+
Модель реализации	+-	+-	+-	+	+	+
Платформы	+	+-	+	+	+	+
Проектирование качества	+-	+	+-	+	+	+-
Преобразования в проектировании	-	-	+	-	+	+
Обоснования в проектировании	-	+-	-	+-	+-	+-

2.6. Примеры систем видов

В архитектурной практике находят применение и другие системы видов, содержащие виды, отличающиеся от представленных выше. Одной из таких систем, связанной со специфическим взглядом на качество АС является система видов (Таблица 2.7), предложенная в монографии Rozanski и P.Wood [73].

Таблица 2.7.

Вид	Содержание
Информационный	Фиксирует историю, преобразования, управление и распределение информации. Используются подходящие модели, отражающие статику и потоки данных. Цель анализа – ответить на существенные вопросы о содержании данных, структуре, принадлежности, умолчаниях, ссылках и перемещениях
Согласованности	Описывает согласованность структуры АС и карту функциональных компонентов, что позволяет ясно идентифицировать части системы , управляемо выполняющих согласованные действия. Это влечёт за собой создание моделей, демонстрирующих процессы и трассы их реализации с учётом коммуникативных отношений
Разработка	Раскрывает архитектуру, которая поддерживает процесс разработки АС. Указывает на определённые аспекты разработки, требующие включения в процесс разработки специалистов определённых квалификаций (например, тестирование, сопровождение)
Размещение	Описывает среду, в рамках которой система будет развёрнута, с учётом зависимостей системы от «реального времени» среды. Вид включает аппаратуру не только системы, но и её среды(например, сетевое окружение)
Операционный	Раскрывает как система инсталлируется, функционирует (используется), администрируется и поддерживается. В построении вида целесообразно использовать уровень задач

К этой системе видов в пособии будут дополнения, раскрывающие подходы авторов к учёту характеристик качества и построениям архитектуры.

Ещё один пример системы видов взят из публикаций [59], в которых предлагается целостный взгляд на архитектуру АС, получивший название SPAMMED. Система SPAMMED базируется на следующей системе видов:

1. Вид с позиции системы требований (Requirements over_View).

2. Вид с позиций сервисов (Service View).
3. Вид с позиций инфраструктуры ПО (Software Infrastructure View).
4. Вид процессов (Process View).
5. Вид предметной области АС (Business Domain View).
6. Вид размещения (Deployment View).
7. Вид среды разработки (Development Environment View).
8. Вид с позиций коммерческой и компьютерной безопасности (COMMSEC & COMPUSEC View).
9. Вид с позиций сохранности (Safety View).

Вопросы по второй главе

Q1. Чем по своей сути является стандарт IEEE-1471-2000?
Набор стандартизированных методов разработки
Ряд строго зафиксированных шаблонов построения АС
"Рекомендуемая практика" предоставляющая разработчикам АС рекомендации для описания архитектуры

Q2. Кто выбирает форматы спецификации концептуального каркаса и несёт за них ответственность?
Пользователь
Эксперт-консультант
Администратор
Архитектор

Q3. Чем являются "проекции АС" на определённый интерес или интересы в IEEE-1471-2000?
Архитектурными профилями
Видами
Точками зрения
Моделями АС

Q4. Что формирует точку зрения в IEEE-1471-2000?
Виды
Перспективы
Цели разработки
Интересы

Q5. Что определяет архитектурный вид?
Интересы
Язык описания архитектуры
Точка зрения
Цели разработки

Q6. Как понимаются виды в контексте "Архитектурных концептуальных схем"?

Средства для понимания, взаимопонимания и коммуникативного взаимодействия лиц, заинтересованных в разработке АС

Средства, помогающие гарантировать целостность, точность и завершённость архитектурного представления АС

Средства, обеспечивающие разумное и конструктивное применение методов и обслуживающих их инструментов

Q7. Что определяет вид планировщиков в архитектурной концептуальной схеме Дж. Захмана?

Определяет сущность предприятия (системы), включая структуры, процессы и организационные структуры

Определяет основные цели и границы, в рамках которых должны приниматься архитектурные решения

Определяет как технология должна быть использована для того, чтобы запросы третьего ряда были удовлетворены

Q8. Что определяет вид пользователей в архитектурной концептуальной схеме Дж. Захмана?

Определяет как должны работать системы в рамках предприятия (в контексте окружения)

Определяет основные цели и границы, в рамках которых должны приниматься архитектурные решения

Определяет как и какая технология должна быть использована для того, чтобы запросы третьего ряда были удовлетворены

Q10. Какой дополнительный слой в трёхмерной схеме Дж. Захмана показывает где в процессе разработки будет создан соответствующий артефакт?

Слой ответственности

Слой разработки

Слой метаданных

Q11. Какие виды в архитектурной концептуальной схемы DoDAF описывают текущий стандартный профиль и предсказания по его изменениям?

Системные виды

Операционные виды

Технические виды

Q12. Какие из предложенных видов относиться к архитектуре "4+1"?

Вид с позиции пользователя

Логический вид

Физический вид

ГЛАВА ТРЕТЬЯ. РАЗРАБОТКА АРХИТЕКТУРЫ

3.1. Архитектура как продукт разработки

Архитектура АС является продуктом определённой деятельности, что позволяет ввести для неё и использовать понятие «жизненного цикла архитектуры АС». Работы по созданию архитектуры конкретной АС начинаются в определённый момент процесса разработки АС, когда в достаточном количестве подобраны необходимые «строительные материалы», а также другие необходимые ресурсы и средства.

Построения архитектуры АС целесообразно доверять квалифицированным архитекторам, которые будут выполнять порученную им работу в рамках подходящих технологий с использованием «хороших» инструментов.

Архитектура АС является продуктом, создание которого осуществляется подобно тому, как создают АС. В результате разработки архитектуры появляется дополнительная автоматизированная система (обозначим её $АС^{А}$), применения которой (в процессах разработки и использования соответствующей АС) осуществляются в формах человеко-компьютерной деятельности.

Понимание $АС^{А}$ как автоматизированной системы вполне правомерно, поскольку $АС^{А}$ состоит из системы информационно-образных структур, которые по запросам визуализируются на мониторах рабочих мест в корпоративной сети. А значит, $АС^{А}$ относится к классу информационно-справочных систем. $АС^{А}$ является системой контроля (контроля рассуждений с помощью понимания), причём с богатым информационным содержанием и мощной системой операций.

Поэтому в разработках архитектуры АС следует использовать опыт разработок АС, включая и всё сказанное выше об архитектурах. То есть правомерно

различать, а значит строить и применять «архитектуру» АС^{А} или, что то же самое, «архитектуру» архитектуры АС.

Разумеется, у систем типа АС^{А} есть определённая специфика, которая находит своё выражение в специфике предметной области АС^{А}, а значит и в исходных принципах для деятельности архитекторов, определённых подходах к их работе, полезных образцах и инструментах.

Представленная точка зрения на архитектуру АС является авторской. Она является богатейшим источником аналогий и подсказок (особенно для понимания), что не раз будет использоваться в последующем тексте.

3.2. Архитектурные парадигмы

В публикациях по архитектуре ПО поднимаются вопросы о парадигмах ПО как «модели постановки проблемы и её решения», определяющей основные подходы к проблемам архитектуры.

Особую известность в этом направлении получила публикация [71], в которой раскрываются следующие технические парадигмы:

1. Объектно-ориентированная парадигма, в рамках которой используется Объектно-Ориентированный Подход (ООП) к структуризации АС на всех этапах её жизненного цикла.

2. Компонентно-ориентированная парадигма, исходящая из принципов сборочного проектирования, конструирования и производства АС, в основе которых лежат программные «компоненты» как единицы сборки.

3. Сервисно-ориентированная парадигма, применение которой базируется на идее массового сервисного обслуживания пользователей АС по их запросам.

Представим каждую из названных парадигм и отношения между ними детальнее. Начнём с отношений, поскольку они взаимодополнительны. Реальность такова, что в общем случае при разработке компонентов используют объектно-ориентированную структуризацию и реализацию, а при разработке сервисов используют компонентные сборки. На практике, особенно при разработке архитектуры АС, целесообразно переключаться между объектной, компонентной и сервисной картинами АС.

1. **Объектно-ориентированная парадигма**. В основе этой парадигмы лежат идеи, методы и средства Объектно-Ориентированного Анализа и Проектирования (ООАП). Наиболее последовательно эта парадигма раскрыта в методологии унифицированного процесса разработки [38] и реализована в работе [39] в комплексе инструментально-технологических средств Rational Unified Process (RUP). Если АС разработана в такой среде, то говорят, что она имеет Объектно-Ориентированную Архитектуру (ООА, Object-Oriented Architecture).

К числу основных характеристик ООА относятся:

- реализация основных принципов ООП, в первую очередь инкапсуляции, наследования и полиморфизма;

- классы объектов являются основными единицами моделирования, проектирования и реализации;

- объекты и их взаимодействие находятся в центре интересов ООА;

- интерфейсы реализуются как специальные классы объектов, обслуживающих взаимодействие объектов;

- удалённые объекты используются точно так же, как и локальные объекты.

Наиболее хорошим примером ООА является Common Object Request Broker Architecture (COBRA), разработанная группой OMG. COBRA определяет объектную модель, в соответствии с которой распределённые объекты должны создаваться, использоваться и управляться. COBRA также определяет Reference Architecture (как систему образцов и инструкций), раскрывающую «как достигается взаимодействие между распределёнными объектами». Для этих целей предлагается язык определения интерфейсов (Interface Definition Language, IDL). Объект клиента получает связь с объектом сервера, после чего он в состоянии активизировать его методы тем же самым образом, как и методы локальных объектов.

ООА может быть представлена различными совокупностями видов. Одной из наиболее распространённых таких совокупностей является система видов Ф. Кратчена «4+1 views» [41]. Для описания видов обычно используются средства языка UML.

2. **Компонентно-ориентированная парадигма.** Сложность современных АС и типичная для них форма разработки в распределённой среде коллективом

разработчиков привела к идее сборки АС из независимо разработанных, хорошо оттестированных и повторно-используемых (аппаратных и программных) компонентов (по образцу сборочного производства в других отраслях промышленности). Эта идея применима для любых этапов жизненного цикла АС, в том числе и для этапа разработки архитектуры АС.

Промышленное производство компонентов, а также платформы, подобные J2EE, CORBA Component Model и Microsoft .Net, перевели компонентно-ориентированную парадигму (и основанную на этой парадигме версию разработки Component-based software development (CBSD)) в практическое русло. Говорят, что системы, разработанные на основе CBSD, имеют Компонентно-Ориентированную Архитектуру (КОА,Component-Based Architecture, CBA).

К числу основных характеристик КОА (CBA) относятся:

- опора на компонентные модели (подобные CORBA Component Model);

- компоненты являются основными единицами моделирования, проектирования и реализации;

- интерфейсы и взаимодействие находятся в центре интересов разработки архитектуры АС;

- интерфейсы могут обслуживать различные компоненты (интерфейсы поддерживают их расширение и специализацию, в том числе и в формах наследования);

- компонентные модели требуют от компонентов поддержки «действий по самоанализу» так, чтобы их функциональности или свойства могли быть открыты и использованы «во время сборки» или в реальном времени;

- особое внимание уделяется образцам проектирования и принципам разделения интересов для определения роли компонента и его ответственности.

Компонентная модель также включает программную модель, раскрывающую как распределённые компоненты должны быть разработаны, собраны и размещены на физическом оборудовании. Кроме CORBA Component Model (CCM) на практике широко используются JavaBeans и Enterprise JavaBeans в рамках Java 2 Enterprise Edition (J2EE).

Для представления СВА могут быть использованы различные совокупности видов, например совокупность, использованная в стандарте для RM-ODP. Для описания видов часто используют средства UML.

3. **Сервисно-ориентированная парадигма**. Большой класс АС, особенно реализованных в виде WEB-приложений, разрабатывают как системы сервисов, к которым открыт оперативный доступ клиентов. Про систему такого рода говорят, что для её разработки использовалась Сервисно-Ориентированная Архитектура (СОА, Service-Based Architecture, SBA).

К числу основных характеристик СОА (SBA) относятся:

- свободное соединение, заключающееся в том что любой сервис может быть получен как ответ на запрос из любого (разрешённого) источника;

- для сервиса не требуется ничего запоминать от одного вызова до другого (состояния загружаются как часть данных сервиса из базы данных или поступают от запросчика сервиса);

- сервисы являются основными единицами моделирования, проектирования и реализации;

- определение, описание, открытие сервиса, протоколы доступа и аспекты качества являются центральными интересами в архитектурном проектировании АС;

- сервис «выставляет напоказ» свои возможности, включая функциональность, данные и характеристики качества сервисов через описание на специальных языках, таких как язык описания WEB-сервисов (Web Service Description Language, WSDL);

- сервис может быть независимо и динамически обнаружен (открыт) и использован;

- так как сервис не имеет состояний, то обмен данными между распределенными пользователями и провайдерами могут привести к существенным накладным расходам.

На рис. 3.1 приведены основные ключевые элементы, используемые в SBA. Сервис может быть реализован с использованием компонентно-ориентированных технологий или объектно-ориентированных технологий. Описание сервиса обычно регистрируется в базе сервисов в определённом

формате, например, в таком как Universal Description, Discovery and Integration (UDDI).

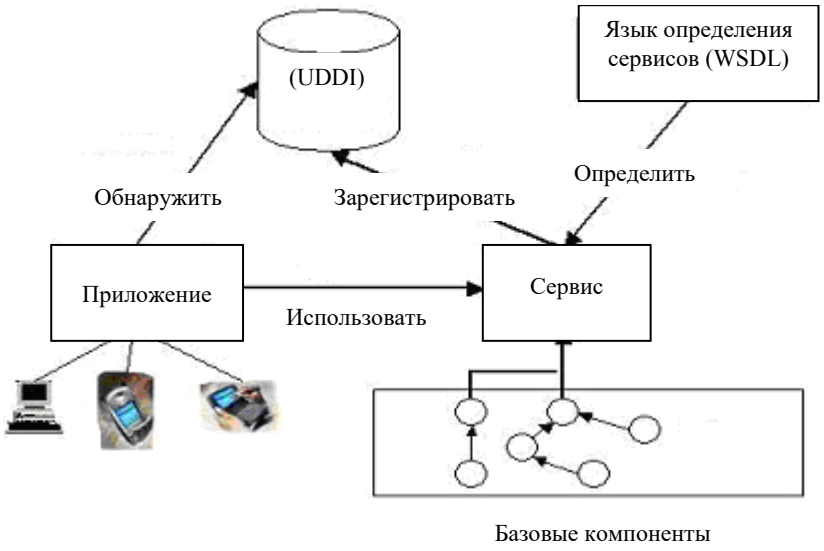


Рис. 3.1. Включение сервисов в приложение

С одной стороны SBA может быть представлена с помощью средств RM-ODP и UML, однако необходимо помнить, что эти средства не совсем адекватны задаче моделирования сервисов. Для описания сервисов более подходят специализированные языковые средства, такие как WSDL.

К месту отметить, что OOA, CBA и SBA предоставляют различные возможности и выгоды и могут быть использованы взаимодополнительно. С точки зрения функциональности сервис может быть реализован с помощью совокупности компонентов с учётом необходимых характеристик качества. В то же время функциональность компонента может быть структурирована в объектно-ориентированном виде и реализована с помощью объектно-ориентированного алгоритмического языка. Однако каждая из парадигм имеет специфику в представлении архитектуры АС. Эта специфика и другие характеристики парадигм в обобщённой форме представлены в таблице 3.1.

Таблица 3.1

Парадигма	OOA	CBA	SOA
Ключевые понятия	Класс, Интерфейс, Объект, ссылка	Компонент, Интерфейс, Контейнер, Сборка	Сервис, Определение сервиса, Регистрация/Обнаружение, Композиция сервисов,
Фокус	Идентификация и определение классов (объектов) и их взаимодействия	Определение компонентов и их интерфейсов, Паттерны (образцы) проектирования	Определение и описание сервисов и их интерфейсов и протоколов доступа
Ключевые особенности	Инкапсуляция операций и состояний, Ссылки между объектами и методы вызова	Промышленное производство компонентов и системных средств их связывания (middleware), Разделение компонентов, Разработка приложений в форме сборок компонентов	Слабое связывание, Самоопределение
Ключевые выгоды	Эффективность, Компактность, Зрелые технологии и языки	Низкая зависимость, Повторное использование, Общность, Легкая интеграция,	Масштабируемость, Расширяемость, Гибкость, Жизнеспособность, Лёгкость связывания, Комплексируемость
Ключевые аспекты	Зависимость, Сильная связность, Низкая гранулированность, Проблема изменений	Различия в middleware платформах, Необходимость поддержки различных стилей взаимодействия, Обслуживание	Качество услуг, Безопасность, Эффективность взаимодействия, Совместимость

Как результат сопоставления парадигм, в [71] предлагается следующий набор практических принципов для применения парадигм в разработке архитектуры АС:

1.Первый принцип нацеливает разработчиков на то, чтобы **Понять приложение** и **Очертить его границы**.

2.Второй принцип нацеливает на выявление **Желаемых атрибутов качества**, выделенный перечень которых открывает доступ к использованию подходящих парадигм по информации из таблицы 1.

3. Третий принцип указывает на необходимость первоочередного построения **Use Case** сценариев для интерфейсов, которые являются ключевыми поня-

тиями в каждой из представленных парадигм. Сценарии помогут прояснить роли, ответственность, характеристики взаимодействия и ожидаемые свойства базовых единиц разработки, а также помогут идентифицировать, какой из трёх типов единиц (объекты, компоненты, сервисы) является наиболее подходящим для разрабатываемой АС.

Применяя принципы в рамках определённой стратегии, необходимо помнить, что парадигмы открыты для их использования в дополнительной манере, разумеется, при необходимости.

Отметим, что совокупность парадигм, используемых в конкретной разработке, не исчерпывается вариантами ООА, CBA и SBA и открыта для модификаций и дополнений.

Отметим, что явное обращение к парадигмам архитектуры ПО в процессе разработки АС вносит в успешность разработки АС свой позитивный вклад.

3.3. Варианты архитектур

3.3.1. Основы архитектурных подходов

В архитектурной практике сложился ряд направлений, в каждом из которых используется определённый акцент на том, что, по мнению последователей направления (в определённых условиях), является центральным или базовым в разработках и использовании архитектуры. К числу таких направлений относятся:

1. объектно-ориентированная архитектура (**object-oriented architecture**) :
2. архитектура, базирующаяся на компонентах (**component-base-architecture**);
3. сервисно-ориентированная архитектура (**service-oriented architecture**);
4. архитектура, базирующаяся на событиях (**event-based architecture**);
5. архитектура, управляемая моделями (**model-driven architecture**);
6. архитектура, центрированная на данных (**data-centered architecture**);
7. архитектура, центрированная на людей (**people-centered architecture**).

Из перечисленных направлений три первых раскрыты выше в связи с вопросами об архитектурных парадигмах. Две последних понятны по названным акцентам, причём акцент на данных указывает на использование архитектурных стилей типа «репозитарий» и «доска сотрудничества». Поэтому представим только два направления с акцентами на события и модели.

3.3.2. Архитектура, ориентированная на события

В основе архитектур, основанных на событиях, лежит событийный архитектурный стиль, то есть приложения, составляющие систему, или их части активизируются при наступлении соответствующих событий. Что это за события и как они приходят в систему (извне или наступают внутри системы) – это вопросы специфики системы, которая, разумеется, должна найти отражение в архитектуре АС. Но главное в том, что процессы в АС управляются (полностью или частично) потоками событий.

К такому типу автоматизированных систем, например, относятся многоагентные системы различного назначения и системы массового обслуживания в распределённых сетях.

3.3.3. Архитектура, управляемая моделями

Архитектурное описание АС не относится к категории исполняемых артефактов. Такие описания не достигли, по крайней мере пока, формы описания, которую можно было бы автоматически оттранслировать в исполняемый код программной части АС. Но разработки исследователей и практиков в этом направлении проводятся интенсивно и уже привели к ряду положительных результатов. Одним из таких результатов является архитектура, управляемая моделями (Model-Driven Architecture, MDA).

Проект MDA был открыт в рамках Object Management Group (OMG), сопровождающей стандарты на версии языка UML [65]. Основная идея заключается в использовании по ходу разработки АС двух классов моделей – класса платформо-независимых моделей (Platform-Independent Models, PIM) и платформо-специфических моделей (Platform-Specific Models, PSM). Классы моде-

лей таковы, что для них разработаны средства преобразования элементов класса PIM в соответствующие им элементы класса SIM, а для моделей класса их преобразования – в программные коды (рис.3.2а). Модели родственны и, например (рис.3.2б), CORBA как специфическая модель PSM, обслуживающая сетевые приложения, для перехода на уровень операционной системы является платформо-независимой PIM и может, например, быть оттранслирована для ОС UNIX. В общем случае, при необходимости, возможны следующие варианты преобразований PIM→PIM, PSM→PSM, PIM→PSM, PSM→КОД, КОД→PSM, PSM→PIM.

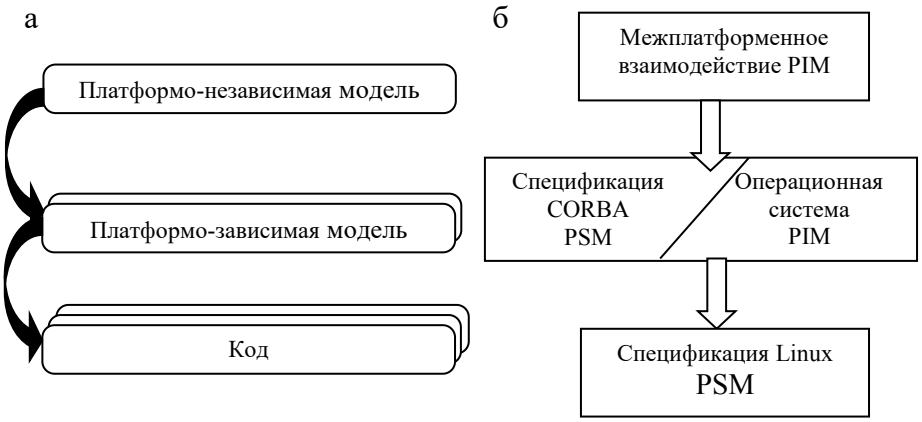


Рис.3.2. Преобразования моделей: а – последовательность преобразования моделей, б –межплатформенные преобразования

Кроме того, для классов моделей PIM и SIM существуют средства их формирования и редактирования в рамках языков со строгой семантикой, то есть языки моделирования являются специализированными формальными языками. Рекомендуемые базовые языки моделирования с указанием их места в MDA названы на рис.3.3. На этом же рисунке раскрыто ядро MDA и названы потенциальные применения подхода, а также отражены типовые функциональности, которые обычно приходится реализовывать в приложениях, используя MDA.

Рис.3.3. Приложения MDA

Завершая представление MDA, отметим, что формирование логики приложения на самом высоком уровне абстракции и автоматическая генерация всех элементов, связанных с платформой реализации приложения, способствуют сокращению сроков проектирования, тестирования и развертывания, полностью исключая затраты времени и сил на кодирование.

3.3.4. Архитектура, ориентированная на шаблоны

Одним из принципиальных подходов к архитектурным представлениям АС является ориентация на шаблоны (паттерны, patterns), что в архитектурах АС привело к направлению «Patten-Oriented Software Architecture», то есть к архитектурам, ориентированным на шаблоны. Следует отметить, что ориентация на образцы (шаблоны) используется в любых версиях построения архитектур. Когда же на «patterns» производится акцент, то имеют в виду направление, основы

которого заложены в работах К. Александра, адаптированные к специфике ПО «бандой четырёх» [30].

Сущность этого направления заключается в следующем понимании шаблона:

«Шаблон проектирования – это способ представления в общем виде как условия проектной задачи, так и правильных подходов к её решению»

Каждый полезный шаблон находит своё место в проекте в результате анализа контекста проектной задачи, что используется для обобщённого определения шаблона в виде

Имя_Шаблона (Контекст, Задача (Проблема), Решение, Результаты).

Ссылка на **Имя_Шаблона** указывает на задачу, её решение и последствия. С помощью словаря имён шаблонов можно вести обсуждение в коллективе, упоминать их в документации.

Для управляемого применения шаблона необходимо сформулировать **Задачу** и её **Контекст.** Также может быть включен перечень условий, при выполнении которых стоит применять шаблон.

С **Решением** связывается абстрактное описание задачи проектирования и того, как она может быть решена с помощью некоторого весьма обобщенного сочетания элементов (классов, объектов и других объектно-ориентированных средств).

Результаты – это следствия применения шаблона. Поскольку в объектно-ориентированном проектировании повторное использование является важным фактором, то к результатам следует отнести и влияние на гибкость, расширяемость и переносимость системы. Перечисление всех последствий помогает понять и оценить их роль.

Более детальное определение шаблона, получившее название GoF-формата приведено в таблице 3.2.

Таблица 3.2.

Имя шаблона	Имя в классификации [30]
Задача	Обобщённая постановка задачи
Также известен как …	Другие хорошо известные имена шаблона …
Мотивация	Сценарий, иллюстрирующий проблему
Применимость	Ситуации, в которых шаблон применим
Структура	Графическая презентация шаблона
Составляющие	Классы, объекты и отношения
Сотрудничество	Распределение ответственности
Последствия	К чему приведёт применение шаблона
Реализация	Как реализовать
Образец кода	Фрагменты кода
Известные реализации	Использовано в «системе»
Родственные шаблоны	Имена родственных шаблонов

Как уже отмечалось, шаблоны используются в любых версиях архитектур АС и их описаний. В то же время интерес к шаблонам и первые результаты были получены для объектно-ориентированной структуризации проектов АС, обеспечивающей структурные представления АС в терминах объектов и их классов.

Для объектно-ориентированной структуризации разработаны специальные библиотеки шаблонов, в основе которых лежат шаблоны, предложенные в [30] и представленные (их именами) в классификационной таблице 3.3.

Библиотеки шаблонов целесообразно строить так, чтобы они включали не только детальное представление каждого из шаблонов таблицы 3.3, но и их модификации, а также версии кодовых представлений на рабочих (для группы или для организации) языках программирования.

Библиотека шаблонов должна быть удобна для оперативного доступа и открыта для включения в её состав новых образцов, их модификаций и кодовых описаний.

Таблица 3.3.

Границы	Цель		
	Генерации	**Структур-ный**	**Поведения**
Класс	Factory Method	Adapter	Interpreter
Объект	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Façade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Momento Observer State Strategy Visitor

В иллюстративных целях представим один из шаблонов. Одним из наиболее распространённых и понятных шаблонов таблицы является шаблон Façade, сущность которого представлена на рис.3.3. Шаблон предназначен для «Предоставления единого интерфейса для набора различных интерфейсов в системе». Другими словами, шаблон Façade определяет интерфейс более высокого уровня, что упрощает работу с системой.

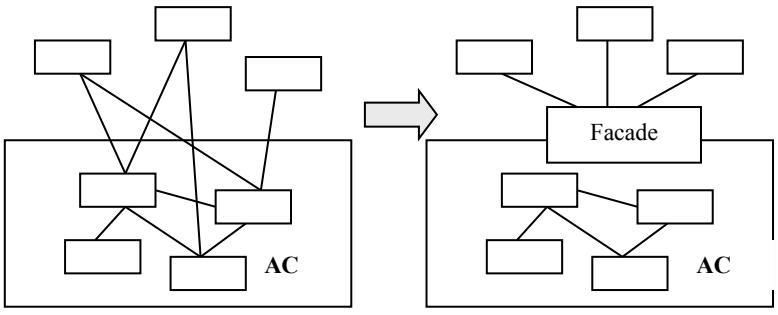


Рис. 3.3. Шаблон Façade: ▭ – элементы структуры

Ориентация на шаблоны приводит к следующей стратегии проектирования:

1. Отыскать шаблоны, присутствующие в проблемной области, и проанализировать полученный набор.

2. Для выделенного набора выполнить следующие действия:

- выбрать шаблон, который в наибольшей степени формирует контекст для всех остальных шаблонов;

- применить этот шаблон к самой высокоуровневой реализации проекта;

- выявить любые дополнительные шаблоны, которые могут появиться в полученной схеме, и добавить их к набору, который был получен в результате предыдущего анализа;

- повторно применить указанный процесс к оставшемуся набору шаблонов, выделенных в результате анализа.

3. Внести в проект все необходимые дополнительные детали.

Такая стратегия в большей мере относиться к «детальному проектированию» АС, чем к построениям её архитектуры. В то же время одна из важнейших позиций построения архитектуры АС связана с разработкой (пусть даже обобщённых) диаграмм классов. Богатым источником образцов для таких построений способны служить библиотеки объектно-ориентированных шаблонов.

Богатым русскоязычным источником учебного и практического материала по шаблонам проектирования является обзор О. Дубиной [84].

3.3.5. Архитектура, ориентированная на предметную область

Существует ряд предметных областей, в которых структуризация статики и динамики АС, используемых в этих областях, имеет устоявшиеся и проверенные на практике формы, приводящие к специфическим архитектурным видам на АС в виде блок-схем (boxes and lines). Например, предметная область «систем управления» с классической схемой управления, представленной на Рис.3.4.

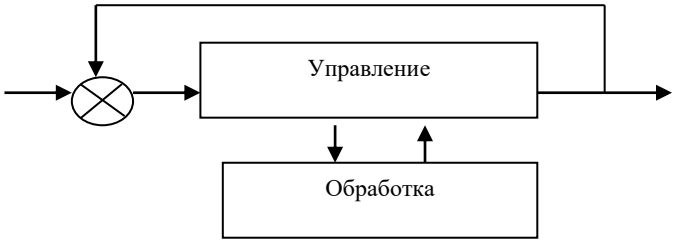


Рис.3.4. Структура системы управления

Ещё одним примером предметной области, устоявшиеся виды на АС в которой игнорировать неразумно, является область «систем массового обслуживания», в блок схемы которых обязательно включают очереди, а значит и обслуживание очередей. Одна из таких схем приведена на рис.3.5.

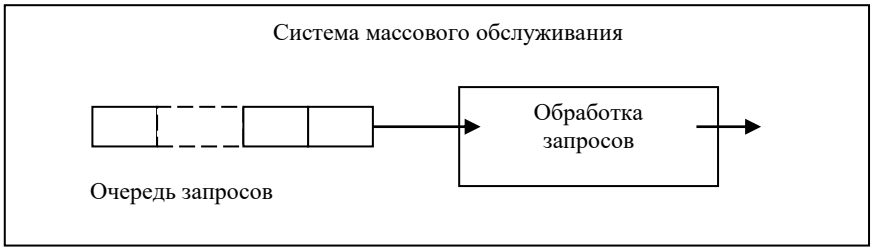


Рис.3.5. Система массового обслуживания

Разумеется, в архитектуру АС, разработанных для специфических предметных областей, следует включать не только традиционные блок-схемы, но и виды, о которых говорилось выше, например, виды RM ODP. Но из-за многократно повторяющихся разработок АС в специфических предметных областях накапливается опыт, в результате которого образуется типовая архитектура, которую целесообразно использовать как образец для подражания при разработках очередных АС.

Такие типовые архитектуры на английском языке называют «Reference Architecture» или «Domain Specific Software Architecture, DSSA». Первое название обычно используют для устоявшихся типовых архитектур в предметной обла-

сти «Архитектура АС». Например, к «Reference Architecture» относят систему архитектурных образцов в RUP. Второе название обычно применяют для специфических предметных областей, в том смысле, о котором говорилось выше.

Завершая пункт, отметим, что к классу специфических предметных областей относится область разработок и применения архитектур АС. Сущность специфики – концептуальное согласование пониманий лиц, вовлечённых в разработку АС, и оперативный контроль рассуждений и понимания в рамках жизненного цикла АС.

3.4. Архитектурные стили

3.4.1. Определение стиля

Как уже отмечено выше, в построениях архитектур АС важны исходные установки:

- в виде парадигмы или согласованной совокупности парадигм, определяющих «модели постановки проблемы и её решения»;

- подхода или согласованной совокупности подходов, выполняющих функции организующего начала для действий, в результате которых создаётся и используется архитектура АС.

Однако, кроме исходных установок, которые носят методологический характер и не привязаны к содержанию АС, лицам, участвующим в разработке и использовании архитектуры, необходимы образцы форм, которые можно было бы заполнять содержанием, раскрывающим конкретную разрабатываемую АС.

В первую очередь нужны образцы форм, которые бы помогли построить эскиз архитектуры АС, или, что то же самое, построить «архитектуру» архитектуры АС. В архитектурной практике функции «образцов форм для построения «архитектуры» архитектуры АС» принято возлагать на **архитектурные стили**.

Понятие архитектурного стиля вошло в архитектуру ПО одним из первых. Его место в разработке архитектуры было уже указано в [52].И все же детальное представление архитектурного стиля, классификация стилей с сопоставлениями

и рекомендациями по применению было проведено только в публикации M.Shaw и P.Clements [63], содержание которой детально раскрывается ниже.

Архитектурный стиль в [63] определялся как «совокупность правил проектирования, идентифицирующих виды компонентов и коннекторов, которые могут быть использованы для образования композиции систем или подсистем, вместе с локальными и глобальными ограничениями, в рамках которых производится построение архитектуры». Причём компоненты, включая их версию в виде вложенных (в систему) подсистем, могут отличаться по их вычислительной природе.

Тип компонентов может быть различным, в зависимости от их объединения в пакеты, например по виду их взаимодействия с другими компонентами. Функции компонентов могут выполнять программные модули, комплексы модулей, «клиенты», «серверы», базы данных, библиотеки, «уровни» [64].

Для того чтобы в каждом конкретном случае прояснить уровень абстракции, используемый в представлении компонентов, их взаимодействие выделяется и оформляется в виде соответствующих коннекторов (например, в виде вызовов подпрограмм, средств доступа к разделяемым данным, протоколов взаимодействия, потоков данных и т. д.). Определение коннекторов включает не только указание на средства взаимодействия, но и правила, управляющие таким взаимодействием.

Авторы [64] в публикации преследовали следующие цели:

- установить единый стандарт спецификации стилей, для того чтобы была возможность более точного и единообразного представления стилей архитекторами;

- обеспечить работу архитекторов систематическими средствами, поддерживающими работы по извлечению информации о стилях;

- находить различия между стилями, которые позволят адекватно выбирать более подходящие стили для определённых задач;

- установить последовательность этапов для выбора наиболее подходящего стиля для заданной проблемы.

3.4.2. «Архитектура» архитектуры

В публикациях по архитектурным стилям, в том числе в публикациях, раскрывающих использование архитектурных стилей, автору не встречалась интерпретация стилей как базовых форм для построения «архитектуры» архитектуры АС. В то же время такая интерпретация, во-первых, является богатым источником аналогий (из теории и практики архитектур АС), а, во-вторых, позволяет более адекватно определиться с назначением архитектурных стилей, а значит, и с их местом и ролью в процессах разработки АС.

Как уже отмечалось выше (п. 3.1), вполне правомерно подходить к созданию «архитектуры» $АС^{А}$ точно также, как к созданию архитектуры АС, учитывая специфику $АС^{А}$, в первую очередь специфику использования $АС^{А}$ как средства, обслуживающего контроль (в формах понимания) рассуждений в рамках жизненного цикла АС.

Следовательно, в рассуждениях по созданию $АС^{А}$ целесообразно выделять тему, содержание которой связано с ответами на самые существенные вопросы об $АС^{А}$, то есть об архитектуре АС. Функции основного из этих вопросов целесообразно возложить на следующий вопрос:

> **$Q^{А}$. Какая декларативно-процедурная структуризация будущей АС позволяет архитектору (или группе архитекторов) наиболее рационально обобщить и связать в единое целое требования лиц, заинтересованных в разработке АС?**

Выбор вопроса $Q^{А}$ на роль основного вопроса об «архитектуре» архитектуры АС объясняют следующие основания:

1. Основные проблемы современных АС лежат в создании и использовании их программного обеспечения, построение которого может быть осуществлено в различных декларативно-процедурных версиях, в каждой из которых используется специфическое разделение (как в компьютерной памяти, так и в физическом пространстве) на компоненты данных (декларативные компоненты) и исполняемые коды (процедурные компоненты).

2. Возможности согласования, обобщения и реализации требований лиц, заинтересованных в разработке АС, в существенной мере зависят от версии де-

кларативно-процедурной структуризации программного обеспечения АС. В наиболее подходящей версии декларативно-процедурной структуризации удаётся наиболее рационально интегрировать требования к АС. Выбор такой версии зависит от профессионализма архитектора(ов).

Наличие целостного декларативно-процедурного образца позволяет архитектору(ам) использовать такой образец для контроля (в формах понимания) собственных рассуждений, нацеленных на построение «архитектуры» для архитектуры АС. Кроме этого, конкретный декларативно-процедурный образец выполняет следующие важные функции:

- он позволяет осознанно и контролируемо проверить и обобщить систему требований к АС, построенную на предыдущем этапе «формирования системы требований»;

- образец используется как руководство для построения на его базе определённой альтернативы «архитектуры» $АС^{А}$ с приписанными ей характеристиками для последующего выбора на множестве альтернатив;

- образец выполняет функции ограничений для действий в процессах архитектурного моделирования АС.

Завершая пункт, отметим ещё и следующее:

- то, что принято относить к архитектурным стилям, является декларативно-процедурной структуризацией АС с позиций программного обеспечения;

- если это необходимо для построения «архитектуры» архитектуры АС, то можно объединять в единое целое группу декларативно-процедурных образцов (группу архитектурных стилей), подобно тому, как интегрируют архитектурные виды.

3.4.3. Классификация стилей

На рис.1.13, раскрывающем место архитектуры АС в процессе разработки АС, отражено место архитектурных стилей. Разработка или выбор и адаптация подходящего стиля проводятся на первых этапах построения архитектуры АС. Положенный в основу разработки АС архитектурный стиль входит в состав архитектуры как её «эскиз», как «архитектура» архитектуры АС.

Архитектурный стиль как эскиз раскрывает АС с позиций рациональной декларативно-процедурной структуризации её ПО, учитывающей специфики варианта реализации. К спецификам относятся специфика физического размещения данных и программ в компьютерных сетях, специфика реализации и передачи управления между частями программ (особенно в сетях), а также включение в процессы АС действий, выполняемых человеком.

Классификация архитектурных стилей, которая наиболее часто приводится в публикациях по архитектуре [63, 64], представлена на рис.3.6.

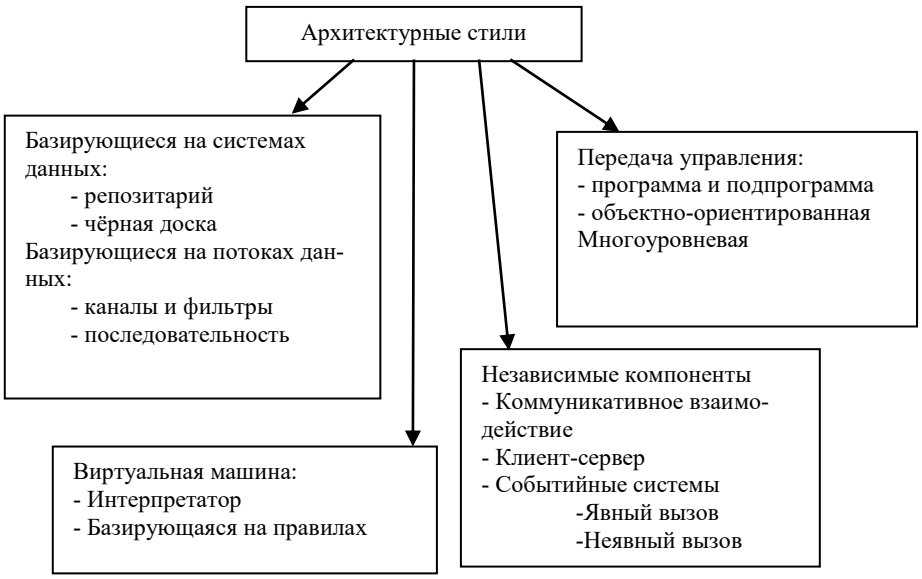


Рис.3.6. Классификация архитектурных стилей

Образцы стилей, раскрывающие содержательную сторону классификации и некоторые детали, приведены в следующем пункте. Заметим, что в определение любого стиля принято включать следующие характеристики :

- словарь элементов проектирования;

- типы компонентов и коннекторов;

- совокупность правил конфигурации, включающих правила:

- для топологических ограничений, определяющих композиции элементов (например, компонент может быть связан более чем с двумя другими компонентами);

- для семантической интерпретации (композиции элементов проектирования должны иметь хорошо понятное значение);

- возможности анализа систем, построенных на основе стиля.

3.4.3. Образцы стилей

Для того чтобы было проще ориентироваться с классификацией стилей, представим формы и обобщённое содержание для наиболее типичных архитектурных стилей.

1.Каналы и фильтры (Pipes and Filters).

Архитектурное описание строится как (или представляет собой) связная совокупность фильтров и каналов (рис.3.7), в которой:

фильтр – независимый блок, получающий на вход поток(и) данных и выдающий поток(и) данных;

возможна достаточно простая переконфигурация системы фильтров (соединение их с помощью pipes);

фильтры работают параллельно.

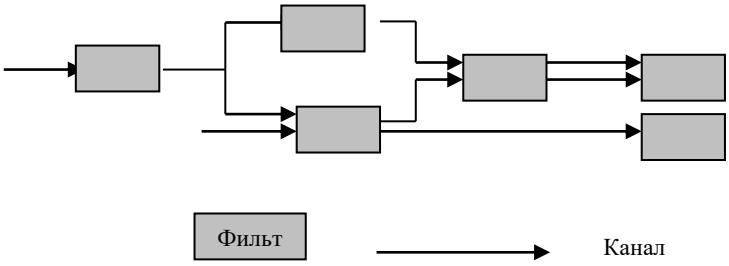


Рис.3.7. Стиль «каналы и фильтры»

2. Стиль, основанный на репозитарии (Repository-based)

В рамках стиля, базирующегося на репозитарии, общие данные разделяет определённая совокупность приложений, каждое из которых в своих обращениях к данным является независимым. Общие данные разделяются и в стиле, получившем название «blackboard».

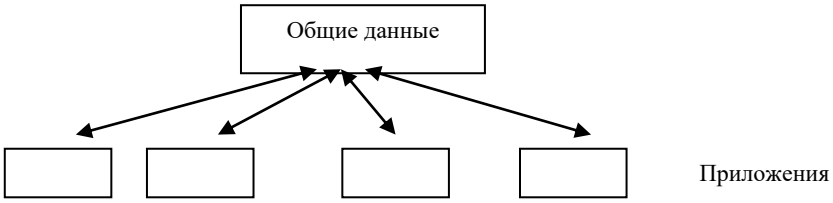


Рис. 3.8. Стиль «репозитарий»

3. Событийно-ориентированный (Event-based) стиль

Событийный стиль (рис.3.9) используется для композиции приложений, которые обращаются (регулярно или случайно) к родственному типу событий (возникающих регулярно или случайно), с которыми взаимодействует система.

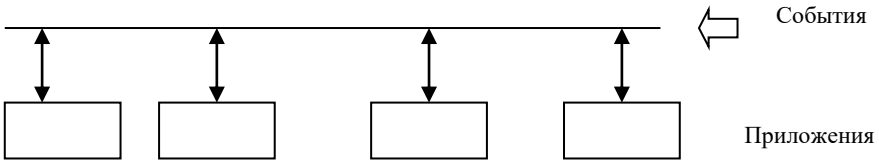


Рис. 3.9. Событийный стиль

4. Одноранговый (Pier-to-Pier) стиль

Одноранговый стиль (рис.3.9) применим в условиях, когда разрабатываемая система состоит из приложений (участников взаимодействия), каждое из которых связано с определённым количеством других. В каждом акте взаимодействия оно устанавливается и происходит только между двумя участниками

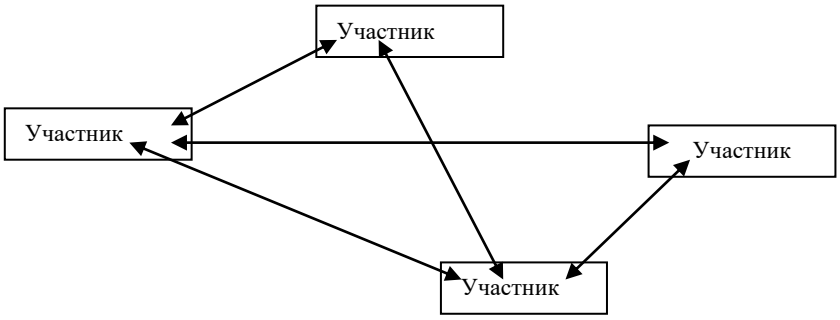


Рис. 3.10. Одноранговый стиль

5. Многоуровневый (Layered) стиль

5.1. Многослойный стиль

Как отмечалось выше, одним из эффективных подходов к разбиению системы на части является использование многослойных структур. Такой подход применим как к приложениям, развёрнутым на одном компьютере, так и для распределённых систем. Типовые разбиения на слои (для таких вариантов) близки по смыслу (рис. 3.11), но имеют различия в терминологиях, используемых для их описаний.

Рис.3.11. Многослойный стиль

5.2. Клиент-серверный стиль

Для распределённых систем типичен случай применения стиля «клиент-сервер», для реализации которого часть системы с определёнными функциональностями размещают на сервере, а другую часть – на клиентских местах пользователей. Два типовых варианта распределения функциональностей с указанием их обощённого содержания представлены на рис.3.12. Один из этих вариантов получил уточнение «тонкий» клиент, а второй – «толстый» клиент. Клиент-серверный стиль получил широкое применение как в корпоративных сетях без использования WEB-возможностей, так и в сетях, использующих такие возможности, в том числе и в сети Интернет.

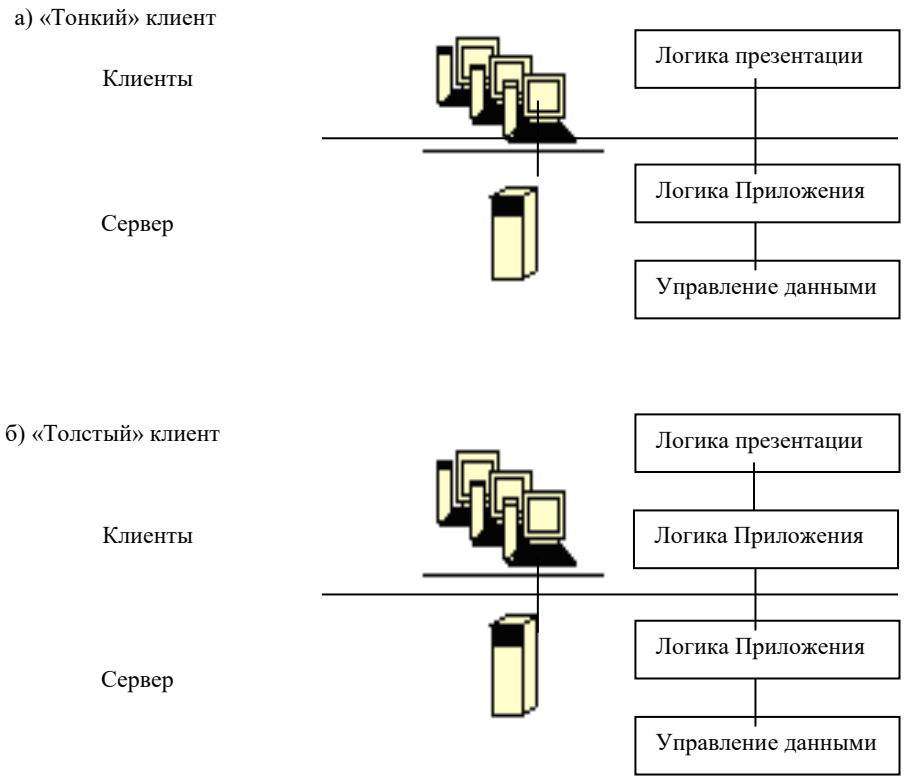


Рис. 3.12. Клиент-серверный стиль: 2 уровня

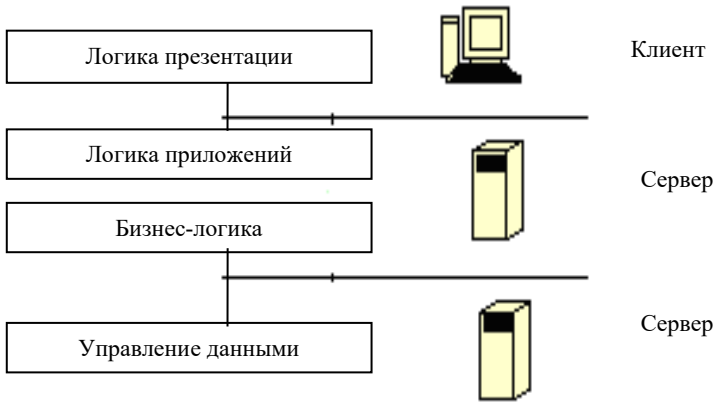


Рис. 3.13. Клиент-серверный стиль: 3 уровня

Представленные на рис.3.12 стили относятся к классу двухуровневых, но специфических двухуровневых, для которых используется не термин «Layer», а «Tier» . Это объясняется тем, что на практике получил широкое распространение архитектурный стиль, представленный на рис.3.13 и получивший название «3-tier». Этот стиль включает уровень бизнес-логики, который разделяет и координирует доступ ряда различных приложений к общим данным.

6. Брокерный (Broker) стиль

Брокерный стиль (рис.3.14) используется для организации доступа определённого количества клиентов к совокупности серверов. Разумеется, для организации такого взаимодействия требуется специальный программный сервис, получивший название «брокер».

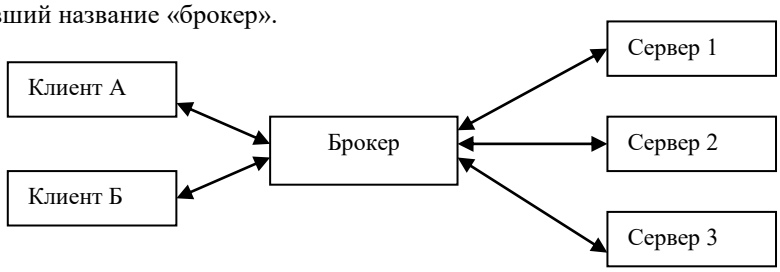


Рис. 3.14. Брокерный стиль

7. Стиль «Модель-представление-управление» (Model-View-Control, MVC)

Стиль MVC согласован с приложениями (рис.3.15), в которых основным видом работ является интерактивное взаимодействие пользователя с визуализированными моделями, представляющими определённые сущности.

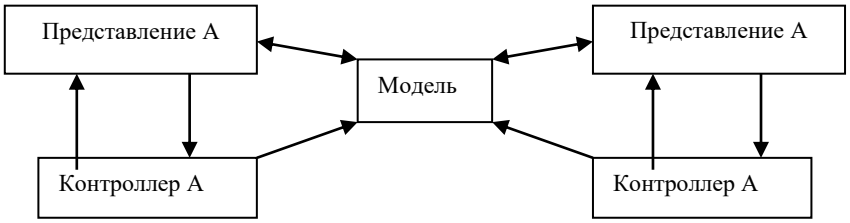


Рис.3.15. Стиль MVC

3.4.4. Сопоставление стилей

Каждый архитектурный стиль является образцом, выбор которого для подражания в конкретной архитектуре проводится на определённом множестве альтернатив. Для того чтобы выбор был возможен, архитектурные стили следует представить в виде, позволяющем выполнить такую работу. Любая попытка создать, например, автоматизированную методику выбора приведёт к «задаче многокритериального принятия решений в условиях неопределённостей».

Наиболее известной, среди используемых на практике, является формулировка отмеченной задачи и методика её решения в рамках **метода анализа иерархий**. Однако в архитектурной практике не распространено представление задачи выбора стиля как решения многокритериальной задачи в условиях неопределённостей. Этому мешает и тот факт, что в конкретной разработке АС могут быть использованы несколько стилей, объединённых во взаимодополнительный комплекс. Взаимодополнительность стилей является типичным архитектурным решением в разработках сложных АС.

Принято упрощать задачу выбора стилей до такого представления ситуации выбора, когда ситуация представлена некоторой совокупностью требований, а множество стилей – табличной структурой, в которой для каждого стиля отмечен тот набор требований, который он (в определённой степени) способен поддержать. Примером такой таблицы является результат сопоставления стилей, проведённый в [29] и пригодный для его использования в WEB-приложениях. Сопоставления архитектурных стилей проведены по характеристикам качества, что согласуется с подходами к разработке архитектуры с позиций качества.

Для практического использования таблицы сопоставлений следует из системы требований к АС выделить набор требований к качеству и использовать этот набор для выбора из таблицы подходящего стиля.

В учебном пособии таблица из работы [29] взята с сокращениями в виде Таблицы 3.3, которая содержит ссылки на стили, представленные выше. В работе [29] представлены детальные характеристики стилей и обоснования для каждой оценки, зафиксированной в Таблице 3.4.

Таблица 3.4

Архитектурный стиль	Исполнимость	Эффективность	Масштабируемость	Простота	Развиваемость	Расширяемость	Привычность	Конфигурируемость	Повторность использования	Визуализируемость	Мобильность	Надёжность
Каналы и фильтры	+ -			+	+	+		+	+			
Репозитарий	+ +		+									+
Клиент-серверный	-	-	+		+				+		+	
Многоуровневый		-	+		+				+		+	
Многоуровневый клиент-серверный	-		+ +	+	+ +				+		+	
Виртуальная машина		+	-	+ -		+	+			-	+	-
Одноранговый	-	-	+		+				+		+	
Мобильный	+	+ +		+ -		+ +	+	+		-	+	

3.4.5. Роли архитектурного стиля в разработке АС

Выше детально раскрыт механизм «точек зрения» и «видов» построении архитектурных описаний. В том случае, когда разработка АС осуществляется на базе композиции архитектурных стилей, каждый стиль выполняет функцию «вида» архитектуры АС. «Эскиз» архитектуры, «архитектура» архитектуры и «вид» на архитектуру – это основные варианты ролей «архитектурных стилей», которые они исполняют в процессе разработки АС.

Исполнение такой роли приводит к тому, что «архитектурный стиль» наследует, а вернее сказать «передает по наследству» архитектуре АС всё то, что отмечалось в п.1.5.2 для архитектуры. То есть, архитектурный стиль (текст приводится повторно специально):

- вносит вклад в извлечение требований 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 (архитектуры) 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.

В отборе, анализе, композиции стилей и их применении важное значение имеют следующие вопросы, ответы на которые управляют действиями архитектора:

1. Какова система понятий, стоящая за определенным стилем?
2. Какие образцы стоят за стилем?
3. Какая вычислительная модель соответствует стилю?
4. Каковы существенные инварианты стиля?
5. Каковы примеры использования стиля?
6. Какие преимущества стиля?
7. Какие недостатки стиля?
8. Какова специализация стиля?

3.5. Архитектура и характеристики качества

3.5.1. Специфика требований к качеству АС

Выше не раз утверждалось, что на начальных этапах разработки АС принципиальная роль возлагается на архитектурные решения по обеспечению требований к АС, которые распределены по реализации АС и отражают определённые «интересы» групп лиц, заинтересованных в разработке АС и её использовании. К числу таких требований, в первую очередь, относятся характеристики качества АС, общепринятая система которых определена стандартом ИСО/ИЭК- 9126.

Необходимо отметить, что проявления характеристик качества при эксплуатации АС создают «слой» вокруг её базовых функциональностей [86], что образно отражено на рис.3.16. У такого слоя два проявления:

- во-первых, большинство характеристик качества воспринимаются пользователем АС со стороны его контактного взаимодействия с АС;

- во-вторых, характеристики качества формулируются, учитываются и оцениваются, начиная с ранних шагов разработки АС.

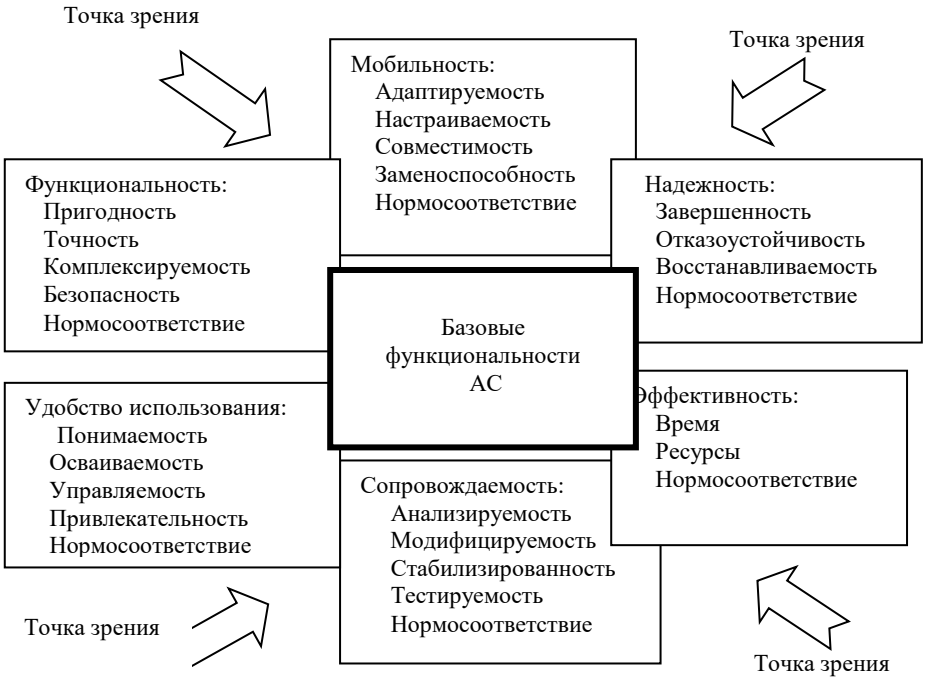


Рис.3.16. Система функциональностей АС

Реальные (достигнутые в процессе разработки) характеристики качества АС материализованы в её реализации, причем элементы материализации качества распределены среди базовых элементов АС или встроены в них. В объектную структуризацию АС «вшиты» (по аспектно-ориентированной терминологии) аспектные решения, обеспечивающие запланированные (с позиций качества) точки зрения на АС.

В процессе разработки и использования АС правомерны и практически полезны, например, «интерес» к АС с позиций «удобства использования» или «сопровождаемости», а также «интерес» с позиции «безопасности» АС. Практика «материализации интересов к качеству АС» показывает, что такие интересы не удаётся представить с помощью подходящего «вида» с точки зрения соответствующего качества. «Следы интереса» (метрики качества) приходится распределять по разным «видам», определённым в соответствии со стандартом IEEE-1471, и даже по разным моделям и документам «видов». «Интересы» кон-

кретного качеств пересекают другие интересы (относятся к типу crosscutting concerns), в том числе и интересы других качеств.

Всё это приводит к тому, что основной задачей построения архитектуры становится не её разработка с позиций функциональности, а разработка с позиций нефункциональных требований, то есть с позиций интегрального качества АС.

По своей сложности и объемам работ создание «слоя качества» часто сопоставимо с созданием системы средств, обеспечивающих реализацию базовых функциональностей АС. Включение «слоя качества» в состав АС существенно повышает её сложность. Однако это не может служить причиной для отказа от создания АС с запланированными и/или заданными требованиями по качеству.

Практика показывает, что обеспечение качества АС – не только запрос конкурентного рынка. Управляемая работа с требованиями качества в рамках АОП, включённая, например, в объектно-ориентированную технологию Rational Unified Process (RUP), способна дополнительно повысить успешность разработок АС [40].

3.5.2. Подход к построению архитектуры с позиций качества

Основной вклад в понимание и построение архитектуры с позиций качества внесли исследования и разработки института SEI Carnegie Mellon [6], в рамках которых разработку АС целесообразно проводить в соответствии со схемой, представленной на рис.3.17.

Схема указывает, что характеристики качества выполняют ключевую роль в построении архитектуры, которая, в свою очередь, управляет процессом разработки АС. Разумеется, функциональность должна быть отражена в архитектуре обязательно, но варианты её реализации должны быть сбалансированы с достижением требуемых характеристик качества, а проблемы такого баланса лежат в области качества.

Ещё одна проблема состоит в том, что баланс приходится осуществлять в условиях, когда АС отсутствует и о её характеристиках качества можно судить

только потенциально, рассуждая о взаимодействии с АС тех групп лиц, каждая из которых заинтересована в определённом качестве АС и его степени.

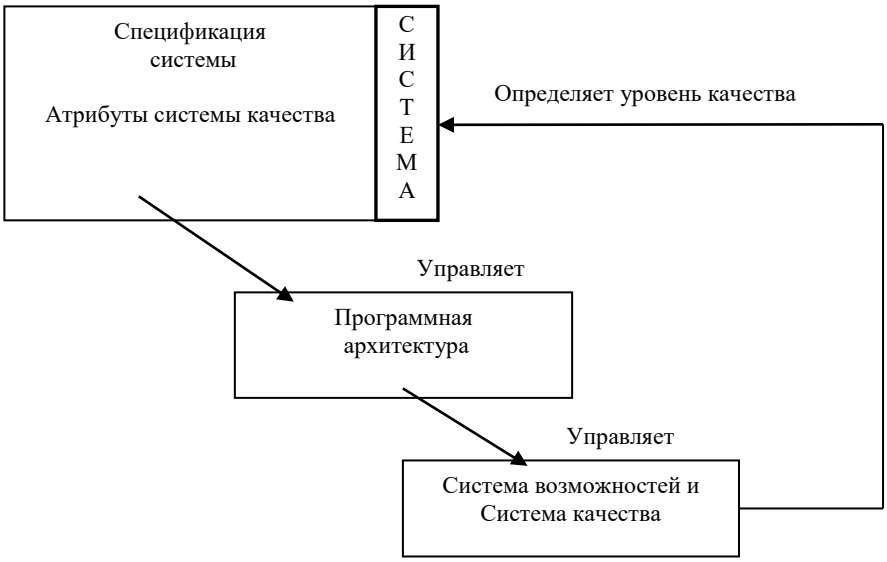


Рис.3.17. Управление качеством АС

Для таких рассуждений (а значит и для построения архитектуры) было предложено использовать сценарии, в каждом из которых отражена определённая «единица» реагирования АС на заданные условия. Была предложена, испытана и внедрена в практику следующая модель сценария:

1. Источник стимула: сущность (актор, человек, компьютер,…), порождающая стимулы.
2. Стимул: предполагаемое воздействие на АС.
3. Среда: условие или состояние, при котором воздействует стимул.
4. Артефакт: то, на что воздействует стимул.
5. Отклик: активность в ответ на стимул.
6. Мера отклика: требование, отражающее определённое проявление качества.

Такая модель сценария подобна схеме условного рефлекса и может быть интерпретирована как условный деятельностный рефлекс, то есть как вполне определённое реагирование актора на условия, которые сложились в опреде-

лённый момент времени и требуют от актора определённой реакции. В этом плане стимул выполняет функции обоснованной (внутренней для актора) причины, материализованной в реагировании, причём определённым образом.

В таблицах 3.5 и 3.6 с иллюстративными целями (для деталей понимания) приведены примеры сценариев и значений каждой из составных частей сценария.

Таблица 3.5

Часть сценария	**Возможное значение**
Источник	Внутренний или внешний для системы
Стимул	Ошибка, упущение, сбой
Артефакт	Процессоры системы, каналы коммуникации
Среда	Нормативное реагирование. Отклонения от запланированных режимов
Отклик	Система должна обнаруживать и демонстрировать определённые события, указание на необходимость «ремонта»
Мера реакции	Допустимое время реагирования, допустимое время на устранение «неполадок»

Таблица 3.6

Часть сценария	Возможное значение
Источник	Конечный пользователь, разработчик, системный администратор
Стимул	Желаемое добавление/модификация/ изменение функциональности, атрибута качества
Артефакт	Интерфейс, платформа, среда; система, взаимодействующая с разрабатываемой системой
Среда	Реальное время, время компиляции, время связывания, время разработки
Отклик	Локальные точки в архитектуре, которая должна быть изменена; модификации, которые не должны влиять на другие функции
Мера реакции	Цена в терминах усилий, финансовых средств и др.

Для разработки архитектуры необходимо извлечь из опыта, а также обнаружить, вообразить и зарегистрировать достаточное количество сценариев.

Этот массив данных следует обработать, сопоставляя значения соответствующих структурных элементов сценариев так, чтобы информация позволила решить основные задачи архитектуры, связанные с балансированием требований. Обработка нацелена на формирование групп сценариев (рис.3.18), позволяющих обнаружить конфликты и непротиворечивые дополнительности в ситеме требований к АС.

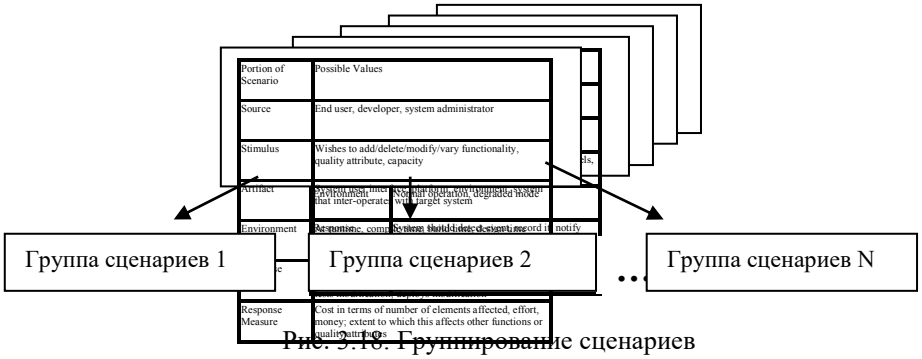


Рис. 3.18. Группирование сценариев

Формирование и обработка сценариев подготавливают последующие действия с этой информацией. Детали таких действий будут представлены в следующем пункте пособия, причём не только с позиций методов, разработанных специалистами SEI.

В настоящее время популярна позиция, предложенная [73], которая исходит из семантического расширения каркаса IEEE-1471, представленного на рис. 3.19.

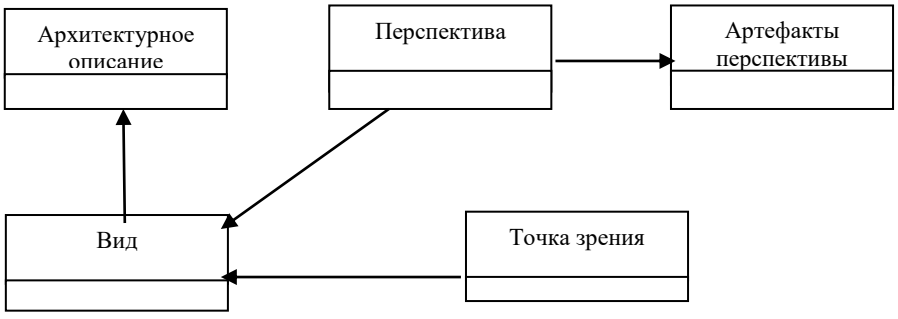


Рис.3.19.Семантическое расширение

В предложенном расширении с важными характеристиками качества связывается понятие «перспектива» со следующим содержанием: архитектурная пер-

спектива – это совокупность действий, схем проверки, тактик и руководств для управления процессом, гарантирующим то, что система будет обладать определённым множеством качественных свойств, согласованных с определёнными архитектурными видами.

«Перспективы» дополнительны к «видам» (рис.3.20), поскольку они позволяют ввести в архитектурное описание интересы, которые невозможно выразить с помощью подходящего вида. В предлагаемой авторами схеме различаются два класса перспектив – основной и дополнительный. Основной включает следующие характеристики качества: производительность, доступность, сопровождаемость, защищённость, размещённость . В дополнительный список перспектив включены

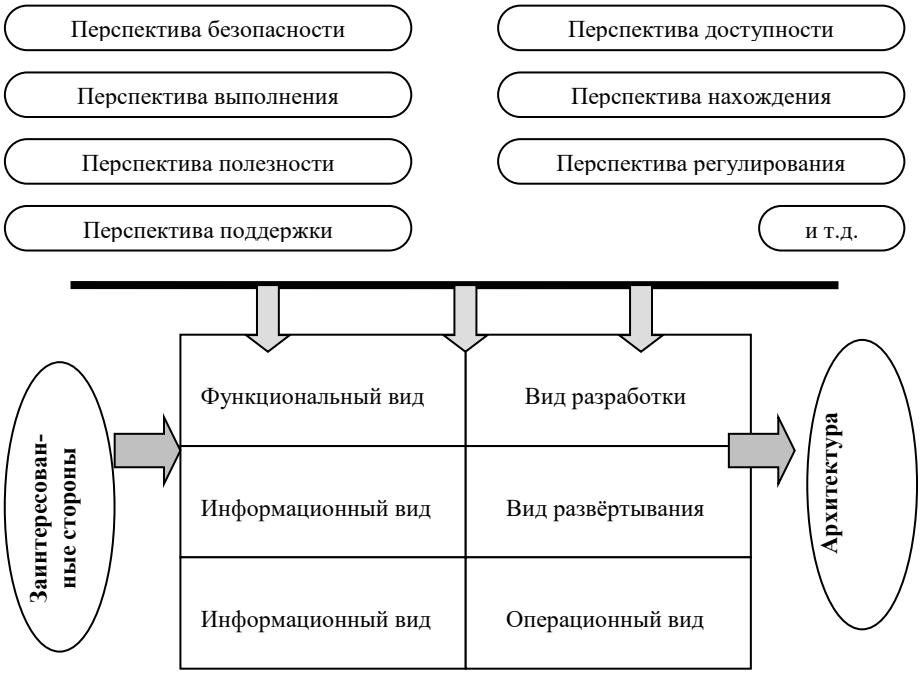


Рис. 3.20.Отношения между видами и перспективами

3.6. Архитектурное проектирование

3.6.1. Основы проектирования архитектур

Архитектура является одной из важнейших форм существования АС, которая до её разработки не существовала или существовала как реализация предшествующей версии или версий. А значит, на определённом этапе разработки АС с помощью определённого процесса и средств архитектура АС должна быть создана, возможно, в новой версии.

Как и любой продукт деятельности, архитектура создаётся для выполнения определённых позитивных функций, конкретнее, для обслуживания процесса разработки АС, а также её использования и сопровождения. Основные задачи, в работе с которыми от наличия АС достигаются определённые позитивы, были названы в п.1.5.2. Разумеется, позитивы достигаются не просто от наличия архитектуры АС, а от того насколько при построении АС были учтен факт и степень их достижения.

На практике широко используются архитектурно-центрированная разработка (Architecture-centric development) систем, интенсивно использующих ПО. Такой тип разработки включает:

- создание бизнес use-case для разрабатываемой системы;
- осознание (понимание) требований;
- создание или выбор архитектуры АС;
- документирование и обсуждение архитектуры;
- анализ и оценка архитектуры;
- материализация системы на базе архитектуры;
- оценка и подтверждение того, что материализация АС соответствует архитектуре.

Такая нагрузка на архитектуру предполагает, что задача разработки архитектуры АС должна быть адекватно (ожиданиям от создания архитектуры) сформулирована и решена. В результате такого решения будет построена концептуальная система «архитектура АС», которая будет выполнять функции «архитектурной модели» АС на последующих этапах её жизненного цикла.

Понимание архитектуры АС как архитектурной модели концептуального типа носит принципиальный характер, поскольку такое понимание предполагает, что «если архитектура АС в форме модели была создана, то архитектурная модель должна быть использована по запланированному назначению». Такая модель (как и любые другие модели) должна исследоваться, когда в этом возникает необходимость, для «получения ответов на появившиеся вопросы». Разумеется, архитектурная модель АС может ответить на вполне определённые типы вопросов, которые были потенциально учтены при создании модели.

Таким образом, для того чтобы архитектурную модель можно было использовать, её сначала нужно построить. Обобщённые детали построения архитектурной модели АС (или, что то же самое, архитектуры АС) представлены на рис.3.21.

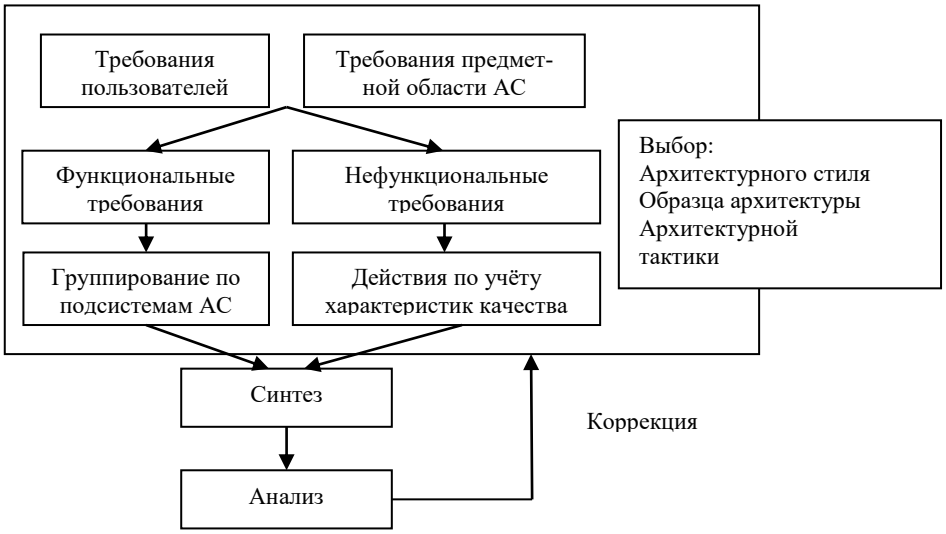


Рис.3.21. Построение архитектуры

Процесс создания «архитектуры АС» состоит из этапов, включающих этап её проектирования. Более точно, процесс создания архитектуры АС, встроенный в процесс разработки АС, носит спиралевидный итеративный характер, что приводит к возвратам к вопросам создания архитектуры АС от «витка спирали к витку».

Архитектурная модель начинает использоваться до её построения в целостном виде. Более того, в итеративном процессе разработки от очередной постро-

енной архитектурной модели переходят к построениям следующей версии модели. А значит, шаг за шагом архитектурная модель становится всё более и более богатой по информационному содержанию, что потенциально увеличивает позитивную отдачу от её использования. Основные позитивы от использования архитектуры (архитектурной модели) АС были представлены в пункте 1.5.2.

3.6.2. Языки архитектурных описаний

Разработка архитектуры АС завершается представлением её проекта, получившего название «Архитектурное Описание». Такой проект носит концептуальный характер и состоит из текстовых единиц (неформального и/или полуформального и/или формального типов), таблиц и графических объектов (в число которых часто включают UML-диаграммы, построенные с использованием формализованного языка UML). Следовательно, при формировании конкретного АО архитектор (или группа архитекторов) стоит перед вопросом «Какие фрагменты АО на каком языке лучше описать?»

За время становления предметной области «Архитектура» был предложен, испытан и внедрён в практику ряд языков описания архитектуры (Architecture Description Languages, ADL).

К числу таких языков относятся языки ACME [90], Rapide [94], Wright [95], Aesop [91], C2 SADL [93] и другие, представленные на рис.3.22.

Рис.3.22.Проблема выбора языка архитектурного описания

В семантике и выразительных возможностях языков архитектурного описания много общего. По этой причине для демонстрации на рис.3.23 представлен фрагмент описания архитектуры для клиент-серверной композиции.

```
System simple_cs  = {
Component client  = {Port send request}
Component server = {Port receive-request}
Connector rpc   = { Roles {caller, callee}}
Attachments : {client.send-request to rpc.caller;
                    Server.receive-request to rpc.callee}}
```

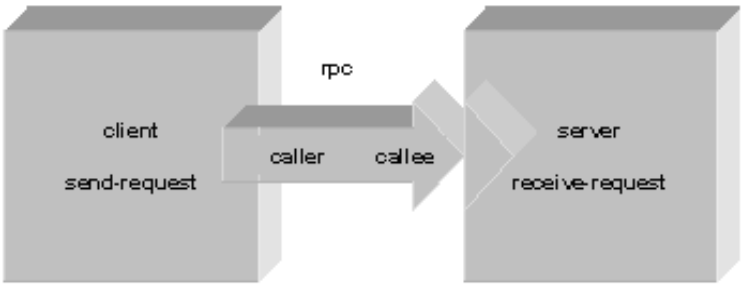


Рис.3.23. Фрагмент описания архитектуры

В применениях языков архитектурного описания различаю как позитивы, так и негативы:

1. Позитивы:

позволяют описывать архитектурные структуры формально;

нацелены на их использование как человеком, так и машиной;

поддерживают описание на более высоком уровне абстракции, чем это было возможно до описания;

позволяют осуществлять анализ архитектуры с позиций соответствия, неопределённостей и реализуемости;

нацелены на автоматическое порождение определённых состояний или форм ПО.

2.Негативы:

- отсутствуют общепринятые соглашения о том, что ADLs должны представлять;

- представления на таких языках трудны для машинного разбора и не поддерживаются коммерческими продуктами;

- большая часть языков создана с академическими, а не коммерческими целями.

В настоящее время широкое использование для построения архитектурных описаний находит язык UML. Детальный анализ языков и причин их трудного внедрения в практику представлен в [75].

3.6.3. Методы проектирования

В практике создания архитектур АС накоплен достаточный опыт проектирования архитектуры, оформленный в виде методов. Так, например, с каждой из приведённых выше концептуальных архитектурных схем связана определённая система построения и использования архитектуры АС. Легко доступны через Интернет, например, системы методов DoDAF [77], TOGAF [80] и FEAF [78]. Для небольших проектов интересна и полезна система методов SPAMMED [59].

В то же время особого внимания, из-за их пионерского вклада в теорию и практику архитектурного моделирования АС, заслуживают методы, созданные в Институте программной инженерии Carnegie Mellon [6,17-20]. По этой причине раскроем эти методы в деталях.

Учёными и специалистами SEI создана система методов разработки архитектуры, обслуживающих действия архитектора или архитекторов по основным этапам жизненного цикла. К числу этих методов относятся: Architecture Tradeoff Analysis Method (ATAM), Cost-Benefit Analysis Method (CBAM), Quality Attribute Workshop (QAW), Active Reviews for Intermediate Designs (ARID), Attribute-Driven Design (ADD). Место использования методов в рамках жизненного цикла отражено в таблице.3.7.

Таблица 3.7

Уровень жизненного цикла	**QAW**	**ADD**	**ATAM**	**CBAM**	**ARID**
Бизнес необходимости и принуждения	Ввод	Ввод	Ввод	Ввод	
Требования	Ввод Вывод	Ввод	Ввод Вывод	Ввод Вывод	
План архитектуры		Вывод	Ввод Вывод	Ввод Вывод	Ввод
Детальный план					Ввод Вывод
Реализация					
Тестирование					
Развёртывание					
Поддержка				Ввод Вывод	

Разумеется, совокупность методов, разработанных в SEI, не перекрывает те виды активностей, которые приходится использовать при построении архитектуры. Место этих методов в составе совокупности активностей отражают таблицы 3.8 и 3.9.

Таблица 3.8

Стадии жизненного цикла	**Архитектурно-центрированная разработка**
Деловое моделирование	Создание системы документов, фиксирующих бизнес - цели: среда, возможности, обоснования и ограничения, презентация образца
Требования	Извлечение, проверка, систематизация и документирование требований (на основе сценариев, раскрывающих атрибутику качества)
Архитектурное проектирование	Проектирование архитектуры с использованием ADD; Документирование архитек5турны с использованием совокупности видов; Анализ архитектуры с использованием ATAM, ARID и CBAM
Детальное проектирование	Проверки с использованием ARID.
Реализация	
Тестирование	
Развёртывание	
Сопровождение	Использование ADD и CBAM.

Таблица 3.9

Метод	Роль	Строка описания	Детали трудового процесса	Продукты
QAW	Системный аналитик	Требования	Понимание необходимостей организатора	Бизнес выбор Добавочная спецификация
ADD	Архитектор ПО	Анализ и замысел	Определение кандидатов архитектур Исполнение архитектурного синтеза	Архитектурные документы ПО
ATAM/CBAM	Технический обозреватель	Анализ и замысел	Усовершенствование архитектуры	Обзор записей Архитектурные документы ПО
ARID	Технический обозреватель	Анализ и замысел	Усовершенствование архитектуры Анализ режимов работы	Обзор записей

Представим сущность этих методов, используя их имена в виде аббревиатуры на английском языке. Первым с позиций его включения в общий поток работ по разработке АС является метод QAW. Его назначение связано с формированием подсистемы требований (как части общей системы требований), на основе которых будет проводиться разработка архитектурной модели. Метод «стоит» на границе между этапами «Анализ требований» и «Проектирование архитектуры». Его применение позволяет внести в общую систему требований вклад (в виде подсистемы этой системы), который будет использован при проектировании архитектуры АС.

Метод QAW базируется на коллективной работе отобранной группы лиц и включает следующую совокупность действий:

1. Презентация группе метода QAW с позиций его сущности и методик.
2. Представление группе архитектурного плана.
3. Выбор и идентификация архитектурных драйверов (характеристик качества).
4. Выявление и формирование множества сценариев.
5. Отбор сводного списка сценариев.
6. Определение приоритетов в списке сценариев.
7. Коррекция списка и его систематизация.

Роль входной информации для такой работы выполняют бизнес use-case и документ «Концепция (Vision)» как архитектурный план. Результатами реализации метода являются откорректированный бизнес use-case и библиотека сценариев. Основная работы при исполнении метода осуществляется в виде рассуждений разной формы.

Следующим, с позиций последовательности применения методов SEI, идёт метод проектирования ADD, использование которого базируется на следующей циклической совокупности действий:

1. Отобрать модуль для декомпозиции.

2. Усовершенствовать представление модуля в соответствии со следующими шагами:

2.1. Выбрать архитектурные драйверы.

2.2. Выбрать архитектурные образцы, удовлетворяющие драйверам.

2.3. Проиллюстрировать модуль примерами и распределить функциональности на основе use-case. Представить модуль с помощью подходящей системы «видов».

2.4. Определить интерфейсы с дочерними модулями

2.5. Проверить и откорректировать use-case и сценарии, выражающие качества, и сформировать на их базе ограничения для дочерних модулей.

3. Повторить шаги для следующего модуля.

Роль входной информации для метода выполняют «концепция» (как ограничения), use-case модель (как источник функциональных и нефункциональных требований) и библиотека сценариев. Выходом служит комплект документов, раскрывающих декомпозицию, распараллеливание и вид, фиксирующий «размещение». И при исполнении этого метода основным видом работ являются рассуждения разной формы.

Методы QAW и ADD представлены обобщённо по публикации [6] авторов методов, которая раскрывает возможность их включения в потоки работ RUP. Главным при исполнении методов является их опора на сценарии, при работе с которыми необходимо придерживаться следующих рекомендаций:

1. При отборе для модуля (или компонента) подходящих сценариев из библиотеки сценариев необходимо отдавать предпочтение тем, которые наиболее существенны и выполняются чаще других.

2. Исходя из определённого архитектурного стиля, проводится определение места модуля в процессе исполнения каждого отобранного сценария и дополнительных (с позиции сценария) модулей, обеспечивающих целостное исполнение сценария.

3. Особо внимание уделяется интерфейсам и обменам сообщениям между модулями для каждого сценария.

3.6.4. Подходы к оцениванию архитектуры

Для анализа характеристик архитектуры АС и оценки её пригодности, а также для сравнительного анализа альтернативного набора архитектур в SEI был разработан ряд методов [18], из которых первым был метод анализа *архитектуры ПО* (Software Architecture Analysis Method, SAAM), идея которого представлена на рис.3.24.

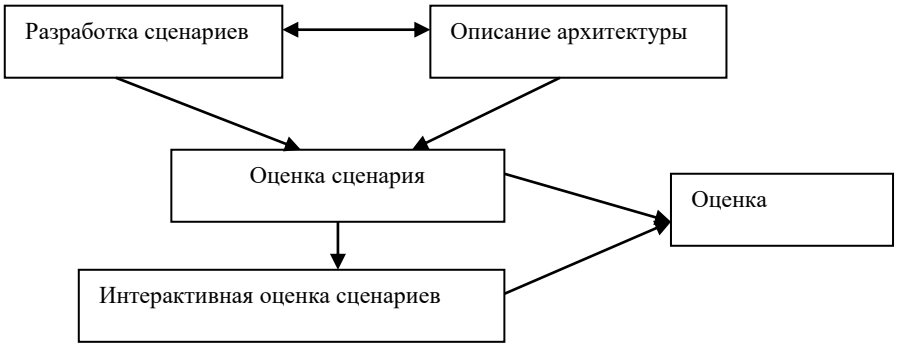


Рис. 3.24.Схема анализа архитектуры

В основе использования метода SAAM лежит работа со сценариями, в содержании которых отражается необходимая для построения архитектуры и её оценивания система функциональных и нефункциональных требований к АС.

Применение метода предполагает выполнение следующих шагов:

1. Определить архитектуру (или несколько сравниваемых архитектур). Описание архитектуры должно быть нормативным для используемой технологии разработки и, возможно, подготовленным для целей анализа и используемых методов оценки.

2. Для оцениваемой архитектуры отобрать из библиотеки сценариев (если она есть) и из других источников представительный набор сценариев. Чем представительней и адекватней набор сценариев, тем выше будет качество анализа. Со сценариями можно связать ожидаемые риски.

3. Провести классификацию сценариев с позиций их потенциальной реализуемости в рамках архитектуры (в том числе и с учётом возможных изменений архитектуры).

4. Для каждого сценария с учётом их классификации провести оценку степени реализуемости в рамках оцениваемой архитектуры. Определиться с необходимыми изменениями архитектуры с достаточной степенью детализации.

5. Выявить взаимодействие сценариев с учётом смысловых связей между взаимодействующими сценариями. Степень связности несёт информацию о потенциальной декомпозиции структуры АС, зарегистрированной в описании архитектуры.

6. Оценить архитектуру в целом (или сравнить несколько заданных архитектур). Для этого целесообразно использовать подходящие методы оценки, учитывающие важности сценариев и степень их поддержки архитектурой.

Кроме метода SAAM, в SEI для оценки архитектур были разработаны, испытаны и внедрены в практику методы ATAM, CBAM и ARID [6].

В основе метода ATAM лежит следующая совокупность шагов:

1. Презентация группе метода ATAM с позиций его сущности и методик.

2. Представление группе совокупности бизнес-драйверов.

3. Представление группе архитектуры.

4. Идентификация архитектурных подходов.

5. Формирование дерева качества.

6. Анализ архитектурных подходов.

7. Мозговой штурм и приписывание приоритетов сценариям.

8. Анализ архитектурных подходов.

9. Представление результата.

Входными данными для метода являются бизнес use-case, документация на архитектуру и содержимое библиотеки сценариев. В результате исполнения ATAM формируются архитектурные документы, включающие результаты анализа и предложения по коррекции. Основным видом работ при исполнении метода являются рассуждений (индивидуальные и коллективные).

В основе метода CBAM лежит следующая совокупность шагов:

1. Проверить и сопоставить сценарии.

2. Усовершенствовать сценарии.

3. Приписать сценариям приоритеты.

4. Определиться с межсценарными характеристиками качества и построить их дерево.

5. Разработать архитектурные стратегии и определить уровни реакций атрибутов качества.

6. Определить выгоды от ожидаемых значений характеристик качества.

7. Вычислить общую выгоду, полученную от архитектурной стратегии.

8. Выбрать архитектурные стратегии на основе возврата от инвестиций.

9. Найти интуитивные подтверждения полученных результатов.

Входными данными для метода являются бизнес use-case, документация на архитектуру и содержимое библиотеки сценариев. В результате исполнения CBAM формируются архитектурные документы, включающие результаты оценок выгод, и предложения по развитию (обогащению) библиотеки сценариев. Основным видом работ при исполнении метода являются рассуждений (индивидуальные и коллективные).

В основе метода ARID лежит следующая совокупность шагов:

1. Определиться с группой рецензентов.

2. Подготовить справку по проектированию.

3. Подготовить группу родственных сценариев.

4. Подготовить другие необходимые материалы.

5. Провести презентацию (в группе рецензентов) сущности и деталей ARID.

6. Провести презентацию проекта.

7. Провести мозговой штурм и приписать приоритеты сценариям.

8. Применить сценарии.

9. Провести обобщение.

Функции входа в ARID выполняют документы архитектурного описания, в которых представлены элементы проектирования доступных сервисов, и выборки из библиотеки сценариев. Применения метода оформляется в виде отчёта

по ревизии. Основным видом работ при исполнении метода являются рассуждений (индивидуальные и коллективные).

На практике применяют и другие методы проектирования архитектуры [25, 59, 80], но в любом случае исполнение этих методов базируется на рассуждениях групп специально отобранных лиц и оформлении результатов их работы в виде определённой совокупности документов.

3.6.5. Документирование архитектурных решений

В построении архитектуры АС рекомендуется (целесообразно и полезно) использовать стандарт IEEE-1471,опираясь и на другие стандарты и нормативы. Результат таких построений должен быть оформлен документально. В состав документов целесообразно включать не только окончательный результат, но и промежуточные результаты, то есть документирование должно сопровождать разработку архитектуры на всех этапах её жизненного цикла. Хорошим руководством по документированию архитектуры способна служить монография [17].

Вопрос о подходящей системе документов, фиксирующих процесс и результата разработки архитектуры, должен решаться с учётом специфики организации, заказавшей разработку АС или выполняющей заказ. Однозначный ответ о требуемом наборе документов отсутствует. Существуют только конкретные инструментально-технологические среды и практика авторитетных организаций (например, Microsoft, IBM, Siemens).

Разработка документации на архитектуру является работой, в процессе исполнения которой приходится решать определённые задачи. Сущность задач необходимо обязательно привязывать к тем нагрузкам, которые возлагаются на «документацию архитектурного описания». К основным применениям документации архитектуры относят:

1. Использование документации для целей обучения, в первую очередь тех лиц, которые будут воплощать архитектуру в реализацию АС. К числу важных групп лиц, которым способна помочь документация на архитектуру, относятся пользователи АС и лица, ответственные за сопровождение АС.

2. Использование документации на архитектуру в виде средства коммуникации между лицами, вовлечёнными в разработку АС на всех этапах жизненного цикла, особенно на ранних этапах.

3. Использование документации для целей анализа состояния разработки и принимаемых решений. И в этой задаче документы по архитектуре полезны для их применения на всех этапах жизненного цикла, особенно на этапах сопровождения системы и создания её новых версий.

В каждой конкретной разработке совокупность задач следует сформулировать и решать с учётом специфики предметной области и используемых инструментально технологических средств. Только в этом случае будет достигнут баланс между требованиями, предъявляемыми к документам на архитектуру АС.

В основе системы документов на архитектуру АС лежат документы (рис.3.25), фиксирующие:

- «виды», вложенные в архитектуру;
- информацию, раскрывающую, как «виды» интегрируются в единое целое.

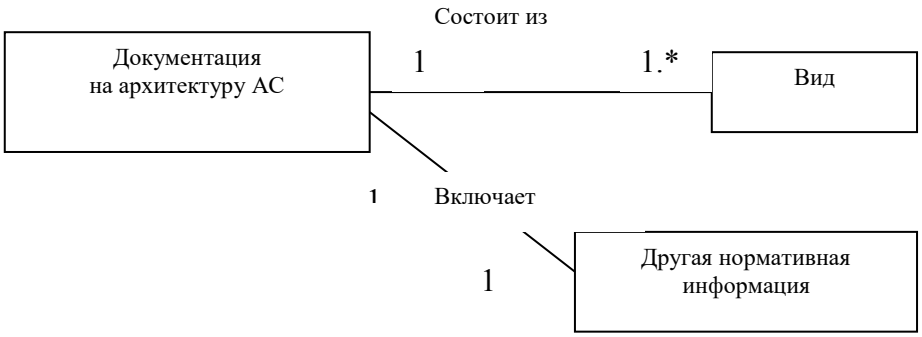


Рис. 3.25. Схема документирования

Виды являются средством, вводящим в документацию базовую структуру не только АС как целого, но и на уровне вида. В практике документирования зарекомендовала себя структура документов на вид, представленная на рис.3.26 .

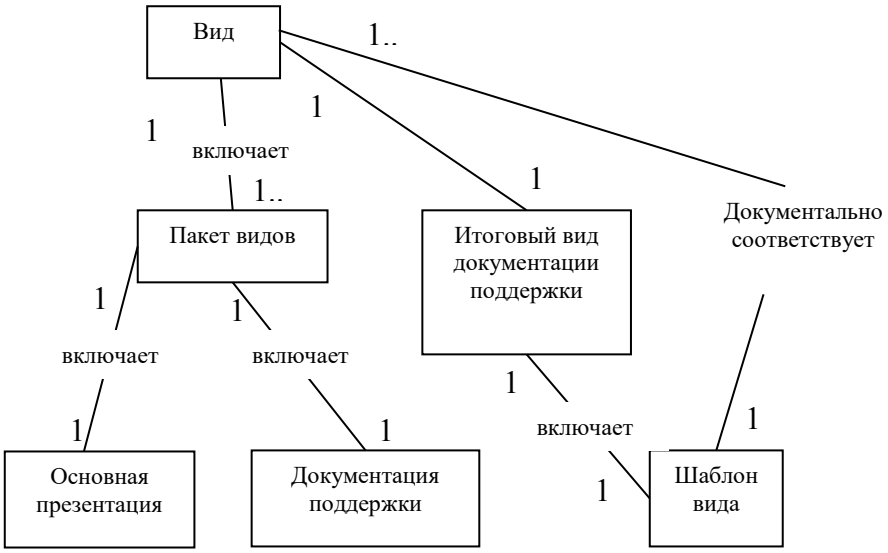


Рис.3.26. Структура документов

Структура документов на вид включает следующее:

- в общем случае (из-за большого количества моделей и документов) документальное представление вида можно разложить по пакетам;

- содержимое каждого пакета может включать графические представления (например, моделей) и сопровождающую (поддерживающую) документацию;

- за рамками пакетов документируется их интеграция и другая поддерживающая информация, для создания целостного документального представления вида;

- если имеется шаблон (template) для документирования вида, то все отмеченные позиции заполняются в соответствии с шаблоном;

- включение документов и их частей, не предусмотренных шаблоном, если есть основания внести изменения в шаблон.

Документы на каждый вид и другие документы должны быть включены в архитектурное представление АС так, чтобы структура общего описания и доступ к частям этого описания были удобны для решения задач, возложенных на

систему документов на архитектуру. Для подобных структур документов также разрабатывают и используют шаблоны.

3.6.6. Рассуждения в разработке и использовании архитектуры

В пособии не раз отмечалось, что в основе построений и использований архитектуры АС лежат рассуждения лиц, заинтересованных в разработке АС. Такие рассуждения в одних определённых условиях выполняются одним лицом, а в других определённых условиях выполняются коллективно (например, мозговой штурм в формировании или анализе системы сценариев).

Одним из средств инициирующих и направляющих рассуждения в архитектурном моделировании являются образцы (например, образцы архитектурных стилей, семантическая сеть IEEE-1471, образцы архитектурных описаний). Каждый образец неявно «задаёт вопросы» о составляющих, из которых он состоит, их свойствах и отношениях между составляющими. На такие вопросы необходимо ответить для того, чтобы попытаться адаптировать образец к исследуемому случаю, возможно, внеся в него изменения.

Для работы с определённым образцом можно использовать различные варианты рассуждений. От того, как организовывать и осуществлять подобные рассуждения, существенно зависит результат работы архитектора(ов).

К сожалению, в практике разработок АС, использующей богатейшие коллекции образцов, в том числе и для архитектурного моделирования, практически отсутствуют образцы для работы с рассуждениями. К числу немногих результатов в области «автоматизированной поддержки рассуждений в процессе разработки АС» относится ряд результатов, полученных в SEI [5], в которых явно называются «рассуждения, reasoning» и предлагаются схемы их реализации. Интерес к рассуждениям в SEI появился около пяти лет назад и связан, в основном, с их базовыми идеями по характеристикам качества АС при построениях и оценках архитектурных моделей.

3.6.7.Аспектно-ориентированный подход к структуризации и интеграции архитектуры АС

Для практики разработок современных АС типичны случаи, когда «интересы», необходимые для учёта, должны оставлять следы, материализующие факт их учёта в совокупности мест, распределённых по физической реализации АС. К такого рода интересам относятся, например, интересы, связанные с требованиями к АС по качеству (раскрыто в п.3.5.1).

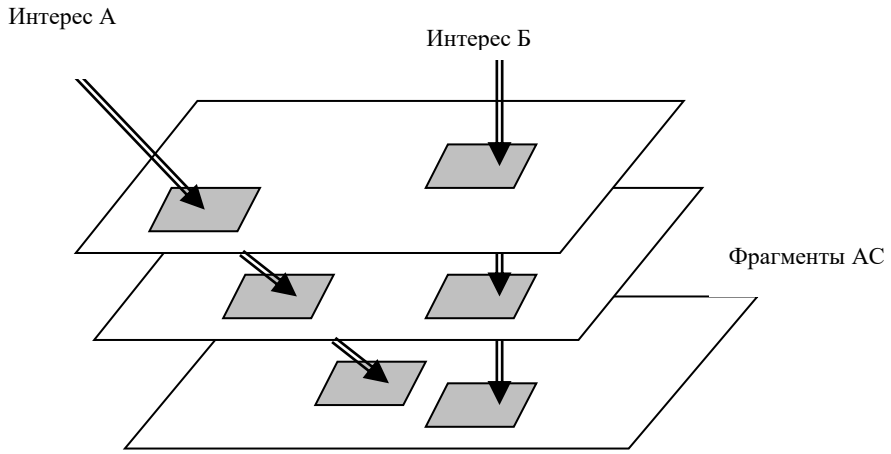


Рис.3.27. Пересечение «аспектов»

Материализованные «следы интересов» в их концептуальном представлении принято называть «аспектами». Реальность «интересов», приводящих к их представлению в виде распределённой совокупности «материализованных аспектов» такова, что они пересекаются (рис.3.27):

- либо разделяя некоторую совокупность фрагментов ПО;
- либо оставляя «следы» одного «интереса» на следах другого «интереса».

Для включения кодов следов в состав фрагментов ПО разработаны специальные средства программирования, получившие название средства «аспектно-ориентированного программирования» [2]. Но для того, чтобы знать, в какие фрагменты ПО и в какие места фрагментов «следы» включить, следует в процессе проектирования АС использовать средства «аспектно-ориентированного

проектирования». Именно такое распределение аспектов по коду программных составляющих АС составляет сущность «аспектно-ориентированного проектирования» [2]. Аспектно-ориентированное проектирование считают очередным шагом в эволюции подходов к проектированию, который следует за «объектно-ориентированном анализом и проектированием» и дополняет его.

Основные проектные решения, относящиеся к задачам аспектно-ориентированного проектирования АС, принимаются при построении архитектуры АС.

Вопросы по третьей главе

Q1. Какая из архитектурных парадигм является наиболее низкоуровневой?

Компонентно-ориентированная парадигма.
Объектно-ориентированная парадигма.
Сервисно-ориентированная парадигма.

Q2. В каком стиле общие данные разделяет определённая совокупность приложений?

Стиль, основанный на репозитарии.
Событийно-ориентированный стиль.
Одноранговый стиль.
Мнгоуровневый стиль.

Q3. Какой стиль используется для композиции приложений, которые обращаются к родственному типу событий?

Стиль, основанный на репозитарии.
Событийно-ориентированный стиль.
Одноранговый стиль.
Мнгоуровневый стиль.

Q4. Чем характеризуется клиент-серверный стиль тонкого клиента?

На сервере компоненты управления данными и логика представления.
На сервере компоненты управления данными и логика приложений.
На сервере компоненты логики приложений и представления.
На сервере компонент управления данными.

Q5. Чем характеризуется клиент-серверный стиль толстого клиента?

На клиенте компоненты управления данными и логика приложений.
На клиенте компоненты логики приложений.
На клиенте компонент логики представления.
На клиенте компоненты логики приложений и представления.

Q6. Чем характеризуется трёхуровневый серверный стиль?
Включает уровень базовых объектов.
Включает уровень интерфейса.
Включает уровень представления.
Включает уровень бизнес-логики.

Q7. Какая отличительная черта у брокерного стиля?
Организован доступ клиентов к совокупности серверов.
Организован доступ ряда различных приложений к общим данным.
Бизнес-логика вынесена в отдельный уровень.
Приложения напрямую обращаются к серверу.

Q8 К какому стилю можно отнести архитектуру WWW?
Тонкий клиент.
Толстый клиент.
Брокерный стиль.
Одноранговый стиль.

Q9. Какова функция уровня бизнес-логики?
Разделяет доступ клиентов к общим приложениям.
Представляет данные от разных серверов в унифицированном для системы виде.
Разделяет и координирует доступ ряда различных приложений к общим данным.
Организует доступ определённого количества клиентов к совокупности серверов.

Q10. Какие языки относятся к языкам описания архитектуры?
Язык UML
Язык ADL.
SDL.
Язык C++.

Q11. Что включает в себя вид с позиции разработки в архитектуре "4+1"?
Прецеденты и сценарии, охватывающие архитектурно существенное поведение, классы или технические риски.
Описание задач (процессов и нитей), их взаимодействия и конфигурации и распределения объектов и классов по задачам.
Краткий обзор модели выполнения в терминах модулей пакетов и уровней.

ГЛАВА 4. СОВЕРШЕНСТВОВАНИЕ НОРМАТИВНОГО АРХИТЕКТУРНОГО ПРЕДСТАВЛЕНИЯ СИСТЕМ

4.1. Основы совершенствования стандарта IEEE-1471

Текст предшествующих трёх глав идентичен пособию, опубликованному автором около 10 лет назад **[**91**]** где в списке литературы использовались два источника [42] и [77], представляющих прошлое, настоящее и будущее предметной области «Архитектурное моделирование систем с программным обеспечением». В этой главе пособия раскрывается как на самом деле происходило совершенствование архитектурного моделирования (реализация «будущего») за прошедшие десять лет на примере адаптации стандарта **IEEE 1471** к международной нормативной базе **ISO/IEK**.

Началось с того, что в 2007 году Международная организация по стандартизации (ISO) опубликовала стандарт **ISO / IEC 42010: 2007**, ***Systems and software engineering — Recommended practice for architectural description of software-intensive systems***. (*Системы и программная инженерия — Рекомендуемая практика архитектурного описания систем, интенсивно использующих программное обеспечение*). Текст этого стандарта **ISO / IEC 42010: 2007** был идентичен стандарту **IEEE 1471: 2000** и был взят за основу для совместной ревизии ISO и IEEE.

Основные направления, цели и составляющие ревизии представлены в [92] и в этой их версии рассматривались на международной конференции **WICSA'2007** (**Working IIIE/IFIP Conference on Software architecture**).

Было отмечено, что стандарт **IEEE-1471** выполнил несколько своих первоначальных целей:

1. Установить систему отсчета терминов и понятий для архитектурного описания.
2. Кодифицировать лучшие практики для архитектурного описания систем, интенсивно использующих программное обеспечения».
3. А также служить в качестве основы эволюции мышления в предметной области «**Архитектуры программное обеспечения**».

Среди основных целей усовершенствования стандарта были названы:

1. Расширение сферы применения приложений стандарта от программных систем до **общей архитектуры с**истем (включая архитектуру предприятия);
2. Согласование с процессами жизненного цикла **ISO** (**ISO 15288**) и разработки программного обеспечения (**ISO 12207**);
3. А также создание общего словарь для согласования терминов и концепций с другими проектами архитектуры **ISO**, включая **RM-OD**P (**ISO 10746**) и **GERAM** (**ISO 15704**).

В результате четырёхлетней ревизии и согласований был принят и в настоящее время активно используется стандарт **ISO/IEC/IEEE 42010:2011**, ***Systems and software engineering — Architecture description (Системная и программная инженерия — Описание архитектуры)***, который доступен по ссылке [93]. Стандарт русифицирован и включён в российскую нормативную базу как **ГОСТ Р 57100—2016 "Системная и программная инженерия. Описание архитектуры"** [94].

Содержание стандарта раскроем следующим образом. Его разработка будет интерпретироваться как **проект нормативной системы** с архитектурой, представленной **совокупностью видов**, каждый из которых фокусируется на определённой особенности стандарта или их совокупности. Такое решение выводит на возможность использование терминологии и моделей стандарта **IEEE 147**1, представленных в предыдущих главах пособия.

Начнём с обобщённо-онтологического вида, приведённого на рисунке 4.1, для понимания которого приведём следующие комментарии:

1. **Систем**у, всегда следует рассматривать в **контексте её ок**ружения (среды) с учётом **интересов** (concerns) **заинтересованных сторон** (stakeholders).

Рис. 4.1. Архитектурное моделирование в разработке систем

2. Каждая система имеет **архитектуру**.
3. Архитектура описывается с помощью **описания архитектуры (architectural description, AD).**
4. **Заинтересованные стороны** используют **AD**, в первую очередь, для **понимания** архитектуры и, посредством этого, **понимания** системы.
5. **AD** создается с использованием **архитектурных каркасов** (architecture frameworks, **AF**) и **языков описания архитектуры** (architecture description languages, обозначено выше как **AD**L)
6. AD включает в себя следующее:
   - Спецификации **заинтересованных сторон**, вовлеченных в разработку и эксплуатацию системы и, в первую очередь, их **интересы**, которые учтены;
   - **Точки зрения** на архитектуру и соответствующие им **виды**;
   - Использованные **модели** выбранных **архитектурных типов**;
   - **Обоснование архитектуры** и **архитектурные решения**, соответствующие обоснованиям;
   - **Правила соответствия** и **экземпляры** таких правил.
7. Архитектурные **точки зрения** обслуживают построение видов и их овеществление в системе.
8. Архитектурные **виды** строятся на базе подходящих **архитектурных моделей**.
9. **Виды** порождаются и регулируются на основе **точек зрения**.
10. Построение **моделей** управляются на основе **типов моделей**.
11. **Обоснование архитектуры** оправдывает **архитектурные решения**.
12. **Правила соответствия** определяют достижение **соответствия**.

Таким образом стандарт определяет совокупность требований к структуре и содержанию следующих артефактов:

• Архитектурное описания системы;

• Выбранные для описания архитектурные каркасы;

• Языки описания архитектуры;

• Выбранные для **AD точки зрения** на архитектуру.

И всё это в контексте следующего определения архитектуры:

> ***Архитектура - это фундаментальные концепции или свойства системы в ее среде, воплощенные в ее элементах, отношениях и принципах ее проектирования и эволюции***

Частично повторяясь с уже сказанным выше, отметим ряд деталей:

1. В онтологии стандарта различают **элементы AD**, к числу которых относятся составляющие набора учитываемых интересов, модели их источников (каждого из различных **заинтересованных сторон**), представления точек зрения, видов, типы моделей, архитектурные решения, обоснования и другие.
2. К числу **заинтересованных сторон** относятся как определённые лица (например, проектировщики, пользователи, владельцы, поставщики и другие лица), а также (возможно) другие системы, **интересы** (требования) которых приходится учитывать в создании АД.
3. С каждым **интересом** к системе связано определённое отношение определённой **заинтересованной стороны** или группы сторон, например заинтересованность в том, чтобы система обладала определённой характеристикой качества.
4. Каждая **точка зрения** устанавливает **соглашения** для построения, интерпретации и использования в архитектурном представлении материализуемого воплощения соответствующего **интереса**. В общем случае, определённый **интерес** может исходить от ряда точек зрения. Соглашения обслуживаются лингвистическим средствами (например, **ADL**), обозначениями, типами моделей, правилами проектирования, методами моделирования и другими операциями над архитектурными элементами и их композициями.
5. **Вид** выражает архитектурную составляющую, представляющую определённый **интерес** или их совокупность согласованно с соответствующей **точкой зрения**.
6. **Тип модели** определяет типовой вариант block-and-line схемы изображающей **вид** или его составную часть.
7. **Архитектурная модель** является результатом спецификации соответствующего **типа модели** в определённых условиях, например, определённая UML-диаграмма.

8. **Обоснование** обычно включает объяснение, обоснование или аргументацию в отношении принимаемых архитектурных решений основу для принятия решения, альтернативы и компромиссы, возможные последствия решения и ссылки на источники дополнительной информации.
9. **Решение** определённый результат выбора, оказывающий воздействие на архитектуры системы или воплощённый в элементах архитектуры или их композициях.
10. **Правило с**оответствия определяющее согласование совокупности видов и **соответствие** как реализация правила или их группы для совокупности видов.
11. **Архитектурный карк**ас (например, каркас Захмана, **TOGAF** или **RM-ODP**), интегрирующий в единое целое определённую совокупность видов. Включает в себя соглашения, принципы и практики для описания архитектур, установленных в определенной области применения и / или сообществе **заинтересованных сторон**.
12. **Язык архитектурного опис**ания (например, Rapide, Райт, SysML или ArchiMate), обслуживающий формирование AD и его составляющих.

Архитектурное описание для конкретной системы создается архитектором или их группой, на которых в коллективе разработчиков лежит принципиальная ответственность. В существующих технологиях разработки систем с программным обеспечением такой фронт работ связывают с ролью «архитектор», для исполнения которой в технологию встраивается необходимый инструментарий.

4.2. Системная ориентация архитектурного моделирования

Принципиальное отличие стандарта **ИСО/ИЕК 42010** от предшественников в том, что он указывает на необходимость конструктивного использования системного подхода, в соответствии с которым:

1. Любую систему **S** как совокупность компонент и связей между ними следует рассматривать в контексте её связей с окружающей её среды.
2. С системной **точки зрения**, любая компонента системы **S** может рассматриваться как подсистема (тоже как система) в согласованной с ней окружающей среде.

Более того, с каждой системой связан её жизненный цикл по ходу которого происходит её становление и существование в определённых состояниях до момента времени, когда существование системы прекратится.

Именно такое понимание системы конструктивно детализировано в стандартах **ISO 15288** и **ISO 12207**, ориентация на которые рекомендована в стандарте **ИСО/ИЕК 42010**. Но такая рекомендация не категорична и разрешает лицам, использующим стандарт, вкладывать его нормы в различные жизненные циклы, различные модели процессов и различные методы архитектуры.

В то же время, в любом случае применения стандарта к некоторой системе S, её окружение (компоненты окружения со связями между ними) на интервалах жизненного цикла следует конструктивно представлять и специфицировать. Такая работа лежит в сфере ответственности архитектора (или группы архитекторов) системы S.

С позиций архитектуры, конструктивное представление среды системы в обязательном порядке должно включать:

1. Модели того, что стоит за объектами, процессами и факторами, оказывающими воздействие на систему по ходу её жизненного цикла.
2. Модели тех лиц и их групп, которые проявляют заинтересованность в системе на этапах её жизненного цикла.

А значит составляющие среды, связанные с воздействиями и заинтересованностями, следует выявлять, моделировать с учётом связей между ними и учитывать в процессах архитектурного моделирования. Такой вид активности отражает концептуальная модель, приведённая на рисунке 4.2, на котором составляющие среды воздействующие на жизненный цикл системы, представлены как «сущности». В реальности «сущность» может быть субъектом или их группой, объектом или их совокупностью, не инкапсулируемым в объект образованием (полем ил подобием ему), системой, процессом или их совокупностью.

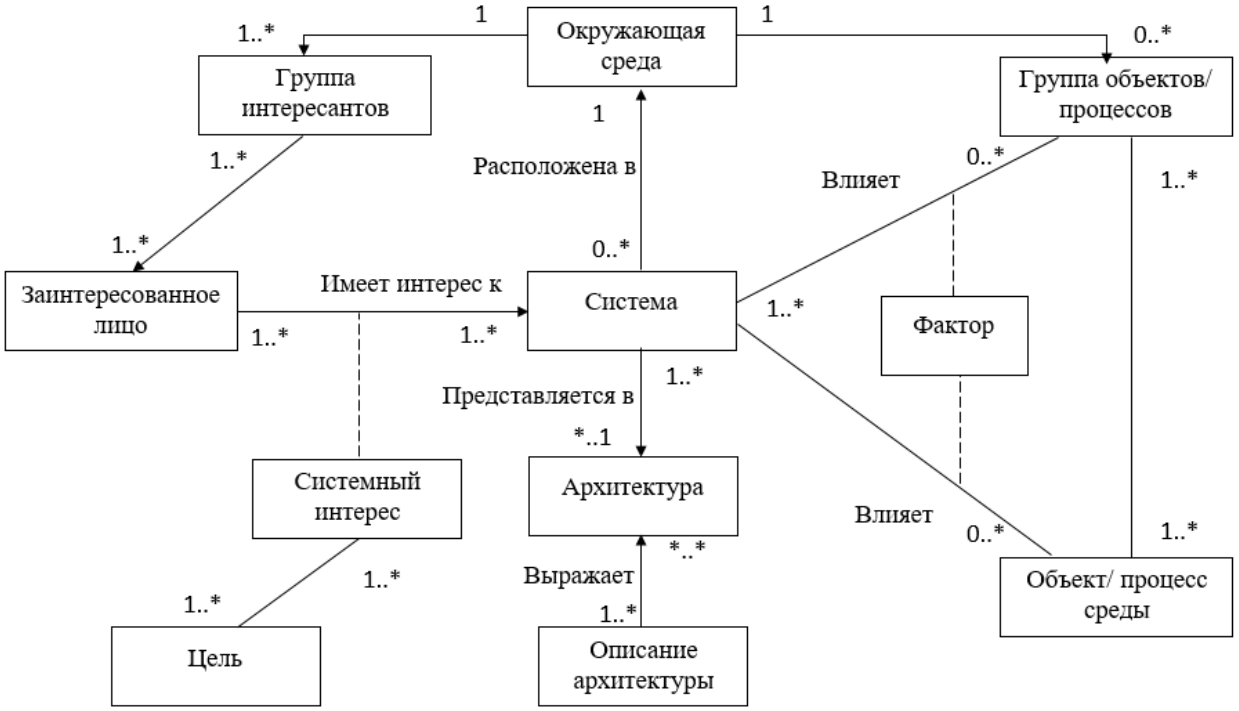


Рис. 4.2. Детали среды архитектурного моделирования

На рисунке 4.2 также показано, что с каждым прямым или опосредованным влиянием на архитектуру определённой составляющей среды связано определённое обстоятельство, которое следует представлять и учитывать, как соответствующий фактор. Ряд таких факторов был представлен в п. 1.4, раскрывающем сложность процесса разработки **АС** и «силовое давление» на этот процесс.

Отметим, что в архитектурном моделировании приходится оценивать широкий спектр контекстуальных факторов, прежде чем принимать решение о наиболее подходящем процессе разработки и его результатах. Практически в любом проекте по ходу его разработки возникают непредсказуемые заранее ситуационные обстоятельства, что является одним из важнейших источником причин уникальности проектов.

Очень полезный обзор ситуативных факторов приведён в публикации [95], в которой исследовано около 400 ситуационных факторов, аналитическая обработка которых привела авторов к справочнику, содержащему 8 классификаций 44 факторов, выводящих на 170 подчинённых подфакторов. Все единица справочника содержательно определены.

Такое влияние среды на разработку в её рамках системы подсказывает целесообразность создания полезных видов на эту сущность, удовлетворяющих рекомендациям стандарта **ISO/IEC/IEEE 42010:2011**. Полезный анализ такого применения стандарта приведён в публикации [96], в которой её авторы предложили различать в среде (в контексте разработки систем) её составляющие, представленные на рисунке 4.3.

Версия контекстной структуризации построена исходя из того, что разрабатываемые приложения будут работать в постоянно меняющихся условиях с **точки зрения** их технических, эксплуатационных и социально-экономических характеристик и должны учитывать этические, социальные, юридические, безопасность и экономические проблемы.

Однако, обычно программные системы концептуализируются и моделируются на основе функциональных и нефункциональных требований, в то время как внешняя среда исполнения, зависимости и контекст неизвестны, воспринимаются как должное или предполагаются или ведут себя определенным образом, что может привести к необоснованным предположения

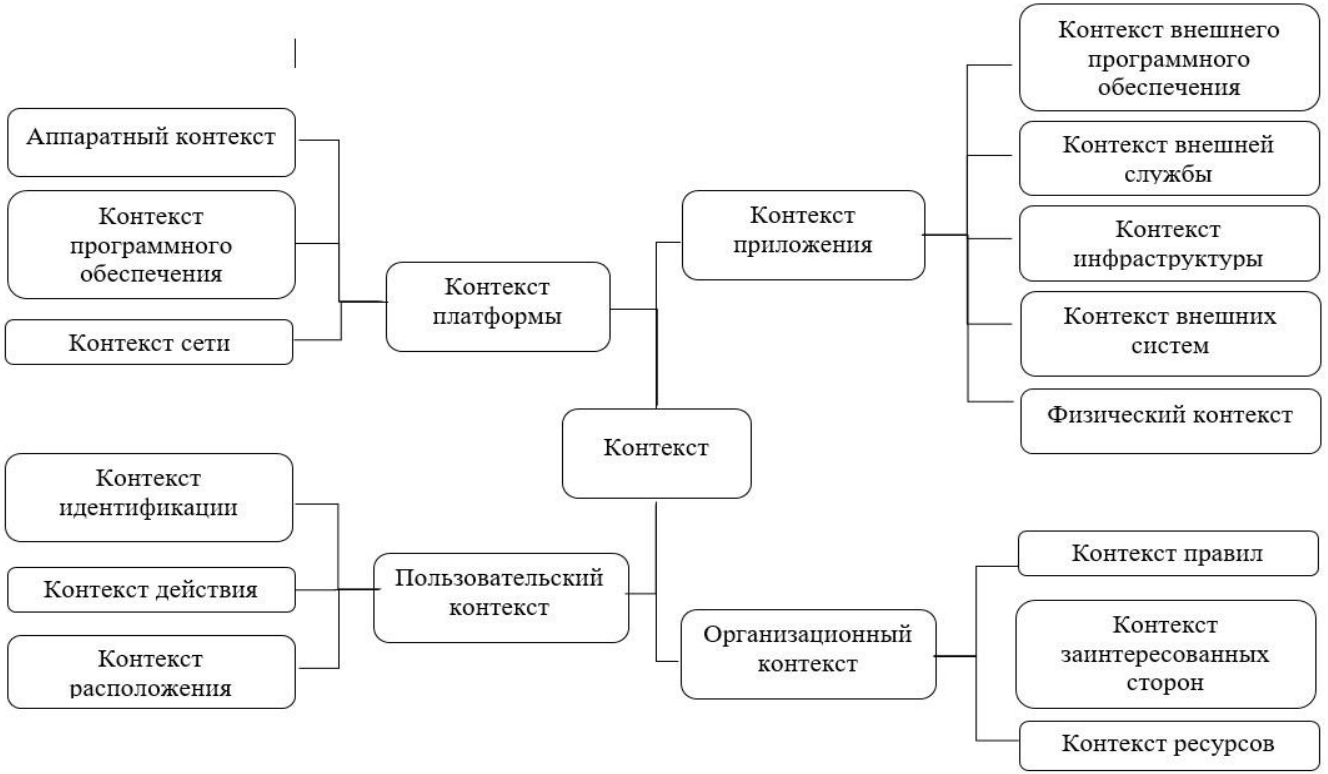


Рис.4.3. Типовые составляющие окружающей среды

Содержание составляющих схемы обобщённо понятно по их названиям. Разумеется в разработке конкретной системы для этих составляющих придётся проводить анализ их содержания в тех деталях, которые придётся учитывать в процессе проектирования. Авторы публикации [96] считают, что в такой работе следует ориентироваться на следующие вопросы:

Какие составляющие находятся в диапазоне управления системой (системная область), а какие нет (область контекста)?

Где граница между системой и ее контекстом и какие взаимодействия между системой и ее контекстом пересекают эту границу?

Кто является пользователями системы; каковы их типы, роли и характеристики; и как и где они получают доступ и используют систему?

Какие внешние службы и / или приложения имеют отношение к системе, включая их свойства и поставщиков?

Какова ожидаемая или желаемая среда технического исполнения, в которой система будет работать?

Какие **заинтересованные стороны**, включая организации и их ресурсы, влияют на систему и каким образом?

Какое влияние оказывает система на организации и **заинтересованные стороны**?

Ещё одной версией учета составляющих среды и их представления является явный учёт пространства проектирования (design space, **S**), документирование его проявлений, а также построение и использование полезных моделей S приводят к позитивам как для процесса проектирования **АС**, так и его результатов.

В наиболее общем плане под пространством проектирования понимают открытое динамическое образование, состоящее из связных объектов разной природы, ограничивающих активность проектировщиков, вовлечённых в согласованную работу. Такое понимание **S** предполагает явное или опосредованное взаимодействие проектировщиков с его составляющими. Более того, оно подсказывает целесообразность отображения **S** на его концептуальное представление - концептуальное пространство проектирования [97].

.

4.3. Заинтересованные лица и их интересы

Разнообразие **заинтересованных сторон** и их **интересов** создает богатство окружающей среды разрабатываемой системы, что, в свою очередь, определяет сложности, в условиях которых архитекторы обязаны осуществить сбалансированный учёт **интересов** и их адекватную материализацию в архитектурных решениях конкретного проекта.

А значит выявление **заинтересованных сторон** и понимание их **интересов** является основой успешной архитектуры. Такая работа должна найти свое явное выражение в её результате, например, в виде представленном (для примера) на рисунке 4.4, где отражено, что отобранные для учёта **интересы** должны быть согласованно систематизированы. По сути дела, за отбором и учётом **интересов** стоит формирование совокупности требований к системе, определяющих ту архитектуру, которую и будет иметь система после её разработки. А значит, эта совокупность определяет класс архитектурных требований к системе.

Рис. 4.4. Системазизация интересов

Следует отметить, что для выявления **заинтересованных сторон** и их **интересов** целесообразно использовать подходящие информационные средства (например, справочники) и инструменты, помогающие архитектуру и повышающие результативность его действий.

Обое место среди таких информационных средств занимают перечни типовых **интересов**, напрмер, следующий их список, выбранный из стадарта

ИСО/ИЕК 42010 и из публикации [98]: приемлемость, доступность, отчетность, точность, адаптивность, администрирование, доступность, гибкость, уверенность, аудитодоступность, аутентификация, автономность, доступность, резервное копирование, бизнес-цели, бизнес-стратегии, сертификация, совместимость, полнота , сложность, соответствие нормативным требованиям, концептуальная целостность, параллельность, конфиденциальность, конфигурируемость, согласованность, контроль, правильность, стоимость, доверие, опыт работы с клиентами, настраиваемость, доступность данных, целостность данных, конфиденциальность данных, надежность, развертывание, аварийное восстановление, документация, долговечность, легкость обучения, простота использования, экономичность, эффективность, защита окружающей среды, обработка ошибок, расширяемость, отказоустойчивость, осуществимость, гибкость, функциональность, общность, внедрение взаимозаменяемость, интернационализация, интероперабельность, обучаемость, правовые, лицензирование, локализуемость, логистика, ремонтопригодность, управляемость, мобильность, изменяемость, модульность, мониторинг, топология сети, открытость, оперативность, эксплуатационные расходы, оптимизация, организация, производительность, постоянство, совместимость с платформой, переносимость, предсказуемость, цена, конфиденциальность, качество обслуживания, возможность восстановления, соответствие нормативным требованиям, надежность, повторяемость, отчетность, воспроизводимость, время отклика, отзывчивость, повторное использование, надежность, безопасность, масштабируемость, расписание, удобство обслуживания, простота, стабильность, изменение состояния, поддержка, живучесть, устойчивость, технологические ограничения, тестируемость, пропускная способность, своевременность, надежность, понятность, удобство использования, универсальность, управляемость.

Отметим, что ориентируясь на типовые **интересы**, следует помнить, что за материальным воплощением каждого **интереса** в проекте стоит определённая цена, а значит, выявляя **интересы**, релевантные проекту, следует учитывать вклад их цены в общую стоимость проекта.

Типовые **интересы** принято структурировать. Наиболее обобщённо, виды **интересов** представлены на рисунке 4.5, где в отдельный вид выделены мотивы, цели, намерения и стремления [99].

Рис. 4.5. Разновидности интересов

Для типовых **интересов** и их подмножеств, отобранных для разработки конкретного проекта, используют не только их имена и виды, а также их определения (спецификации), включающие область значений для каждого атрибута каждого определения. В формулировке определений **интересов** принято ориентироваться на концептуальную модель их окружения, представленную на рисунке 4.6.

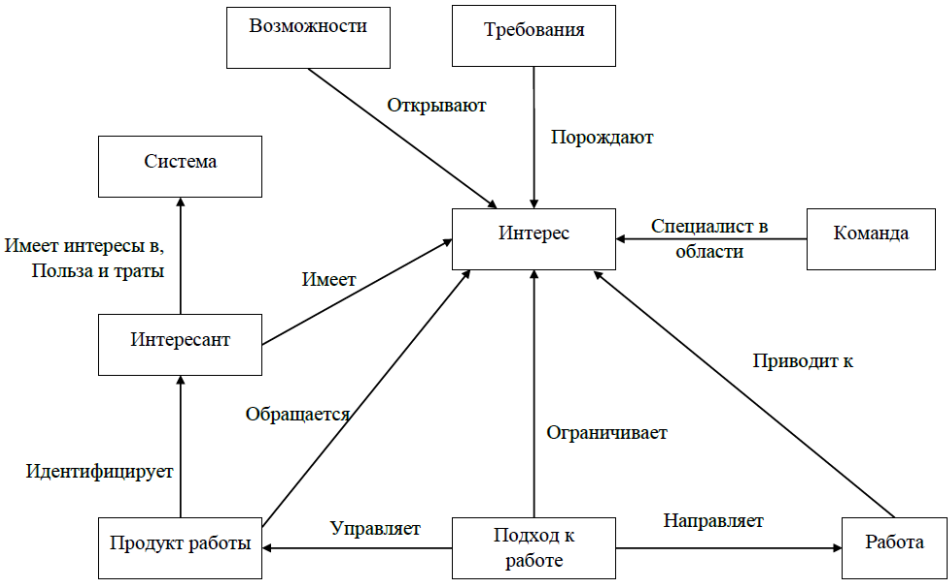


Рис. 4.6. Концептуальная модель «интереса»

Отметим, что для многих **интересов** существуют их определения. Так, например, для **интересов**, связанныъ с качеством, полезны стандарты **ISO9126: 2001** и его усовершенствование **ISO 25010:2010**.

Определения (спецификации) **интересов** должны в обязательном порядке учитывать их согласованную объективацию (материальное воплощение) в реализованном проекте. Более того, по ходу проекта следует целессообразно отслеживать состояния объективации каждого **интереса** и согласованности их объективации.

Обобщая содержание пункта, отметим, что в разработке архитектурного описания, принципиальными видами работ являются:

1. Выявление **интересов** лиц, прямо или опосредованно имеющих отношения к проекту.
2. Согласованная спецификация отобранных **интересов** и, тем самым, формирование системы **интересов**.
3. Мониторинг состояния объективации каждого из **интер**есов (степень достижения запланированнх значений материализации) и их системы по ходу реализации проекта.

4.4. Концептуальная модель

Центральное место в стандарте ИСО/ИЕК 42010 занимает вид, модель которого (рисунок 4.7) расширяет концептуальный каркас стандарта IEEE1471, представленный во второй главе пособия на рисунке 2.1. Как и для предыдущего стандарта, по ходу применения каркаса его составляющие следует наполнить содержанием, соответствующим разрабатываемому проекту. В процессе спецификации схемы, представленной на рисунке 4.7, основное внимание архитектора направлено на формирование «архитектурного описания», в котором центральное место занимают конструкты типов «вид моделей» и «**точка зрения**».

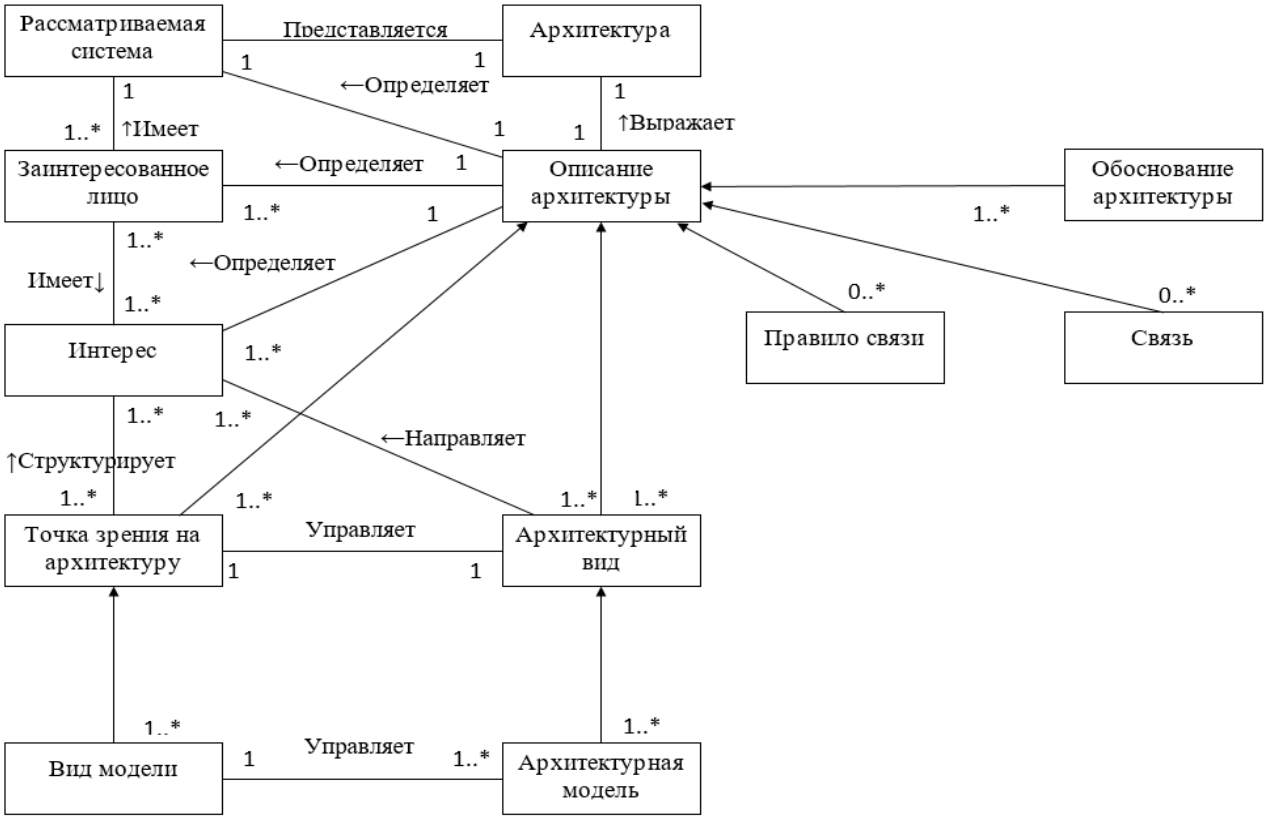


Рис. 4.8. Концептуальная модель стандарта

При создании каждого вида, архитектору приходится, учитывая его физическую и концептуально-алгоритмическую реализуемость, выполнять следующие работы:

1. Согласованно и обоснованно выбирать и/или строить составляющие его модели так, чтобы их композиция составляла системное образование в форме, способствующей пониманию структуры и содержания вида.
2. Порождая вид, обеспечивать его согласованность с остальными видами архитектурного описания.
3. При обнаружении несогласованностей, вводить необходимые изменения в уже построенные виды, используя подходящие механизмы управления изменениями.

Несложно заметить, что за обобщённо названными работами стоит целесообразность использования лучших практик, освоенных в постоянно расширяющейся предметной области «Архитектурного моделирования систем». Например, в известнейшей технологии Rational Unified Process [100], специальный набор лучших практик включён в состав инструментального сопровождения роли «Архитектор» [101].

Необходимость использования архитектором (или лицом, исполняющим роль архитектора, или их группой) расширяемого набора лучших практик приводит к их оценкам с позиции профессиональной зрелости осуществляемого архитектурного моделирования. Для таких оценок принято использовать так называемые модели профессиональной зрелости (Maturity Models) [102].

С позиций профессионально зрелого архитектурного моделирования и оценок его реализации следует различать:

1. Профессиональную зрелость построения архитектурного описания разрабатываемой системы.
2. Профессиональную зрелость овеществления архитектурного описания в реализации системы.

Именно по этой причине, архитектурное описание (изменяясь, когда в этом возникает необходимость) востребовано на всех этапах жизненного цикла соответствующей системы.

Полезная версия модели профессионально зрелого построения **AD** приведена в публикации 2, в которой специфика уровней зрелости отражена в их названиях.

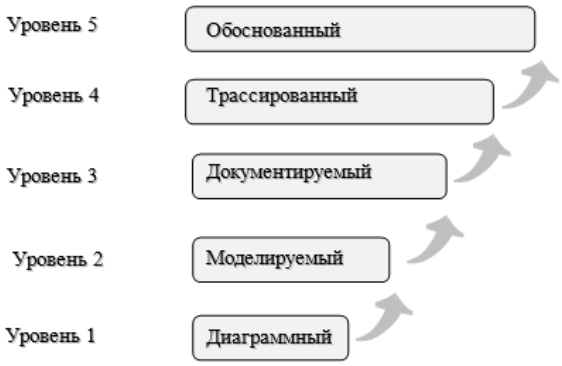


Рис. 4.8. Уровни профессиональной зрелости

Раскроем детальнее содержание уровней.

На «Диаграммном» уровне зрелости для описания архитектуры используются средства графики, например, средства псевдографики Microsoft PowerPoint. Использование block-and-line схем облегчает понимание архитектуры. Основным недостатком таких схем является их семантическая бедность, из-за чего содержание диаграмм может быть неверно истолковано, что может приводить в проекте к семантическим ошибкам.

Использование средств уровня «**Моделирование**» нацелено на уменьшение риска неправильного толкования за счёт формального описания архитектуры. Именно такая нагрузка возложена на языки типа **ADL**. Другим видом формального описания является использование мета моделирования. Таким образом, семантика определяется в метамодели, которая выражает, как структурированы действительные экземпляры модели. В реальной практике архитектурного моделирования, наиболее популярной метамоделью для описания программных систем является система метамоделей **Unified Modeling Language**.

На «**Документированном**» уровне в дополнение к документации уровня моделирования, в которой основное внимание уделяется описанию архитектуры, соответственно структуре системы, документирование раскрывает **проект-**

ные решения. Это означает, что в документации необходимо указывать использование шаблонов проектирования и их связь с элементами архитектуры. Этот уровень зрелости требует указания использования шаблонов проектирования, но не требует объяснений использования соответствующего шаблона проектирования.

Уровень «**Трансированный**» раскрывает **причинно-следственные связи проектных решений** с **требованиями** к разрабатываемой системе. Кроме того, при изменении требований зависимые проектные решения могут быть легко **идентифицированы** и надлежащим образом изменены. Основным преимуществом уровня 4 является сокращение использования несоответствующих шаблонов проектирования и **управление изменениями**.

Пятый уровень зрелости требует раскрыть **обоснования проектных решений**, исходя из **причин** и учитывая **альтернативы**. Такое обоснование решений способствует их пониманию лицами, которые не участвуют в разработке системы, но должны ее поддерживать. На этом уровне раскрываются **зависимости между решениями** и проводятся **обоснования** их **согласованности**.

Включение в процессы архитектурного моделирования интерпретации его процессов и результата с позиций профессиональной зрелости приводит:

1. К конструктивному представлению (на каждом уровне) «точек зрения» для соответствующих «видов» соответствующих **интересов**.
2. К возможности оценивания каждого **интереса** учитываемого в **AD** как лингвистической переменной с нечёткими значениями, привязанными к шкале уровней зрелости.

По первой из этих позиций напомним какое содержание вносится в понятие «**точка зрения**». Это набор образцов, шаблонов и соглашений для построения одного типа «архитектурных видов». В нем определяются **заинтересованные стороны**, чьи проблемы отражены в **точке зрения**, а также руководства, принципы и шаблонные модели для построения соответствующих видов.

Таким образом за определённым употреблением понятия «**точка зрения**» стоит построение соответствующего вида, возможно по шаблону из ресурсов «**точки зрения**», причём в диаграмматической форме с дополнениями, выполняющими интегрально роль совокупности требований к системе, реализация

которых приведёт к овеществлению соответствующего **интереса** с запланированной в архитектурном описании оценкой профессиональной зрелости.

В приведённом толковании «**точки зрения**» специально выделены два этапа действий – формирование совокупности требований к воплощению определённого **интереса** (например, характеристики качества) в разрабатываемую систему и материализация этих требований в разработанной системе.

Представленная на рисунке 4.8 структура уровней и их содержание относятся к оценкам первого этапа. Сохранит или нет (вложенная в **AD**) оценка зрелости **интереса** своё значение в разработанной **AC** зависит от уровня профессиональной зрелости «лучших практик», материализующих требования к воплощению **интереса** на этапе реализации, то есть на втором этапе.

По этой причине с каждым **интересом**, выбранным для воплощения в **AC**, целесообразно связывать две оценки его интерпретации как лингвистической переменной – первую оценку для представления **интереса** в **AD**, а вторую оценку для материализации в разработанной **AC**.

Отметим, что использование интерпретации **интересов** как лингвистических переменных с нечёткими значениями открывает возможность для вычислений оценок совокупностей **интересов**, учитываемых в **AD**, а также **AD** как целого. Более того, появляется возможность отслеживания значений оценок для текущего состояния процесса разработки.

4.4. Специфика архитектурных решений

В предыдущем пункте был раскрыт ряд деталей, связанных с действиями архитектора или их группы, нацеленными на формирование и использование «архитектурных видов». В результате таких действий каждый **интерес**, выбранный для его учёта в разрабатываемой **AC**, получает определённое диаграммматическое представление, дополненное деталями, соответствующими тому, каким образом **интерес** будет овеществлён в реализованной **AC**. А это означает, что каждый «архитектурный вид» - это форма выражения определённого требования или требований, которые должны быть материализованы в **AC**

Чуть выше было отмечено, что четвертый уровень профессиональной зрелости построений **AD** исходит из того, что в **AD** регистрируются связи с требо-

ваниями к АС , которые были выявлены за рамками архитектурного моделирования, в первую очередь с теми требованиями, на основании которых архитектор приступил к построениям **AD**, пытаясь их (согласовав) интегрировать в целостности в формах, способствующих пониманию.

Каждую из таких целостностей, то есть определённый «вид» *V(t*) архитектор начинает формировать, в общем случае, в рамках следующей импликации

$$??U(t) \xrightarrow{???W(t)} ?V(t),$$

составляющие которой имеют следующее содержание:

1. Для вида V(t), который решено вложить в **АС** известно намерение – его реализация в **АС** должна овеществить соответствующий **интерес**, представленный (семантически нечетко) лингвистической переменной. На этот факт в импликации у составляющей V(t) указывает знак вопроса «?».
2. Формирование вида начнётся и будет осуществляться в условиях U(t), в которых архитектор обязан учитывать сложившуюся до начала его работы совокупность требованиям, но он, если это полезно для проекта, может вносить в них изменения, а также расширять эту совокупность, в том числе за счёт требований, интегрируемых в V(t). На такой вид неопределённостей, с которыми сталкивается архитектор в совей работе, в составляющей U(t) указывает два знака вопроса «??».
3. Ещё больше неопределённостей, которые придётся преодолеть архитектору в той работе W(t), которую ему придётся выполнить, специфицируя импликацию для последующей материализации вида V(t). По этой причине W(t) в импликации помечена тремя знаками вопроса «???».

Специфика работы архитектора, обусловленная достаточной свободой в поиске ответов на вопросы, обусловленные отмеченными неопределённостями, определяет специфику архитектурных решений и их включение в нормативы стандарта **ISO 25010:2010**. В частности, в этот стандарт включена концептуальная модель, представленная на рисунке 4.9 и фокусирующая внимание на архитектурном решении.

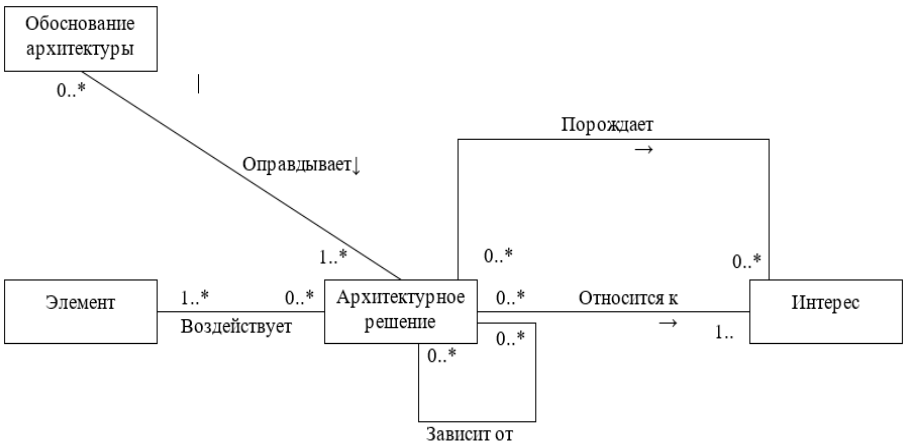


Рис. 4.9. Контекст архитектурного решения

Особой составляющей схемы архитектурного решения является то, что обозначено как «элемент». В наиболее общем плане «элемент» - это любая конструкция в описании архитектуры (**заинтересованная сторона**, **интерес**, **точка зрения**, вид модели, архитектурная модель, архитектурное решение, обоснование и другие составляющие **AD**). Важным является то, что многое включается в **AD** в результате решения, которое должно быть обосновано.

А значит, «архитектурное решение» можно и целесообразно рассматривать с позиций полезных **точек зрения**. В этом плане отметим исследования «архитектурных решений», представленные в публикации [103], авторы которой конструктивно определили и проверили на практике следующий **точки зрения**:

- «Детали принятия решений»;
- «Связь с требованиями»;
- «Хронология принятия решений»;
- «Участие **заинтересованных сторон** решения».

Исследования и спецификации предложенных точек зрения проведены в [103] с использованием следующего набора вопросов:

Q1 Какие решения были приняты?

Q2 Каков текущий набор соответствующих решений?

Q3 Каково обоснование решения D?

Q4 Какие **интересы** C_i имеют отношения к решению D?

Q5 Какие силы воздействуют / повлияли на каждое решение?

Q6 Какие решения затрагивают **интерес** Q?

Q7 Какие решения имеют находятся в конфликте с **интереса** C?

Q8 Какие решения затребованы в решении D?

Q9 Какие решения противоречат решению D?

Q10 Какие решения зависят от решения D?

Q11 Какие решения связаны с решением D?

Q12 Какие решения влияют на решение D (или элемент E архитектуры)?

Q13. На какие решения допускают изменение?

Q14 Какие решения будут затронуты при интеграции набора решения S?

Q15 Как применить набор решений из другого проекта?

Q16 Какие **заинтересованные стороны** затронуты решением D?

Q17 Какие решения затрагивают **заинтересованные стороны** S?

Q18 Какие **заинтересованные стороны** были вовлечены в решение D?

Q19 На какие решения влияют **заинтересованные стороны** S?

Q20 Каков порядок принятия в совокупности решений?

Q21 Какие решения были изменены с момента времени T?

Q22 Какие решения стали устаревшими после изменения CH?

Q23 Какие решения D или подпункты решения SG могут быть повторно использованы в других проекты?

В ответах на эти вопросы, которые были распределены между названными **точками зрений**, они были конструктивно специфицированы, и, что особо важно, авторами [103] был построен концептуальный каркас, представленный на рисунке 4.10.

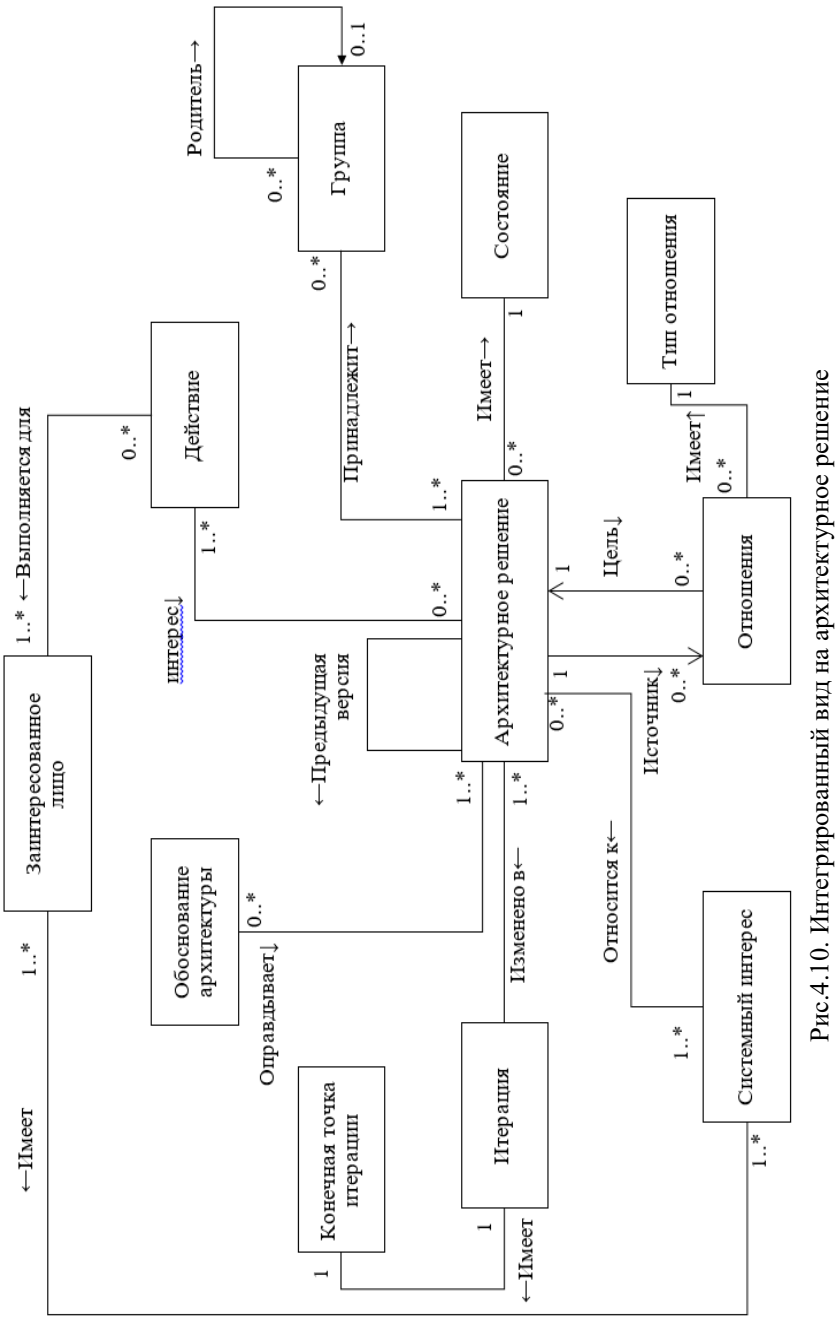


Рис.4.10. Интегрированный вид на архитектурное решение

.

Интегрированный вид подчёркивает, что, в общем случае, архитектурное решение носит итерационный характер, обусловленный необходимостью согласовать совокупность решений в непротиворечивое целое.

Отметим, что в следующей за [103] публикации [104] авторы расширили набор точек зрения на архитектурное решение, добавив в него **точку зрения** «Силовое воздействия на архитектурное решение» (Force Decision Viewpoint), модель которой приведена на рисунке 4.11.

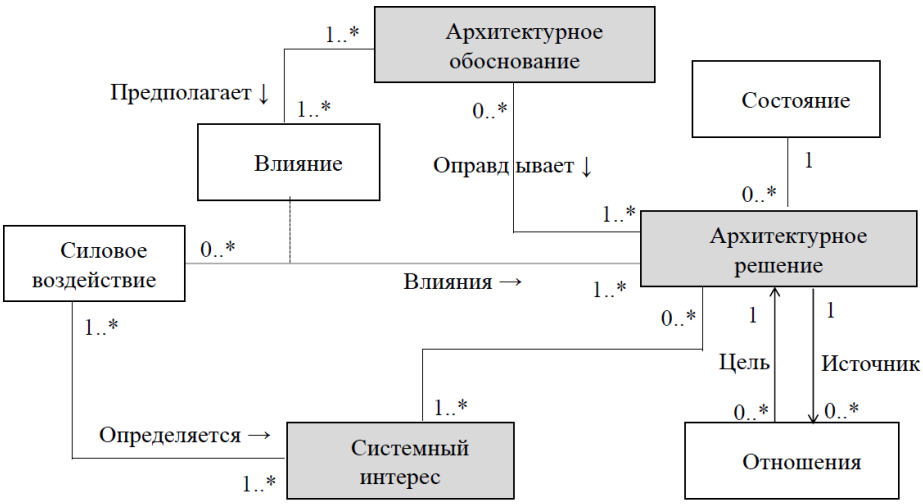


Рис.4.11. Силовое воздействие на архитектурное решение

Представления, использующие **точку зрения** «Силовое воздействия на архитектурное решение», четко определяют отношения между архитектурными решениями и силами, которые влияли на архитектора при принятии решений При наличии множества альтернатив. Термин «сила» берется из множества образцов, которые использует силы для разработки описания проблемы, которая должна быть решена с помощью подходящего шаблона. «Силу» можно определять, как «любой аспект проблемы, который следует учитывать при ее решении». [105]. Аналогичным образом, при рассмотрении архитектурных решений сила - это любой аспект архитектурной проблемы, возникающей в системе или ее окружении (операционная, развивающая, деловая, организационная, политическая, экономическая, правовая, нормативная, экологическая, социальная и т. Д.), учитывается при выборе среди доступных альтернатив решения. Напомним,

что силовая интерпретация воздействий разработку **АС** рассматривалась выше в п. 1.4 пособия.

Силы возникают из многих источников:

- Чаще всего из требований, а также из ограничений, принципов архитектуры и других «намерений», налагаемых на систему;
- Личных предпочтения или опыта архитектора(ов) и команды разработчиков;
- Бизнес-цели, в частности таких как ограничения по времени на разработку проекта, ограничений на финансирование;
- Стратегической ориентации на конкретные технологии.

Перед принятием решений архитектор рассматривает и оценивает все «силы», имеющие значение в контексте разрабатываемой системы. Это может быть хорошей практикой для формирования и использования списка типовых «сил», специфичных для предметной области **АС**.

Различные силы могут быть ортогональны друг другу, они могут поддерживать, противодействовать или противоречить друг другу. Поэтому архитектор должен балансировать силы для принятия наилучших возможных решений.

Так что, конструктивная спецификация **точки зрения** «Силовое воздействия на архитектурное решение» открывает возможность её использование по важному направлению согласования архитектурных решений.

Завершая пункт, отметим, что именно практика согласованного принятия архитектурных решений вывела на аспектно-ориентированное распределение **интересов** по архитектурным видам. Другими словами, в общем случае, конкретный **интерес**, например определённая характеристика качества не инкапсулируется в одном «архитектурном виде» (раскрывалось выше в п. 1.4). То есть, определённое качество распределяется по некоторой совокупности видов, в которых они могут «пересекаться» с другими характеристиками качества [106].

Вопросы по четвёртой главе

Q1. К какому типу интересов относится то, что называют «конфигурируемость»?

Архитектурные цели.

Качественные требования.

Функциональные требования.

Q2. Чем отличаются такие составляющие архитектурного описания системы как «точка зрения» и «архитектурный вид»?

Средства «точки зрения предназначены для построения «архитектурного вида».

Средства «архитектурного вида» предназначены для построения «точки зрения».

Не отличаются.

Q3. Какой формальный язык не используют при построении архитектурного описания?

SysML.

C#.

Rapid.

Q4. Что из ниже перечисленного может выполнять роль «интереса»?

Удобство пользователя.

Намерение.

Производительность.

Q5. Может ли интерес распределятся по совокупности видов?

Да, может.

Нет.

В особых случаях.

Q6. Для какого класса систем можно применять стандарт ISO/IEC/IEEE 42010:2011?

Информационные системы.

Системы теплоэнергетики.

Автоматизированные системы.

Q7. Что из ниже перечисленного оказывает воздействие на архитектуру системы?

Операционная среда.

Нормативные ограничения.

Экологическая обстановка применения системы

Q8 Спецификация каких составляющих AD остается за рамками стандарта IEEE-1471?

Правила соответствия.
Архитектурные решения.
Виды моделей.

Q9. На каком этапе разработки АС применим стандарт ISO/IEC/IEEE 42010:2011?
Концептуальный этап
Этап сопровождения.
Этап тестирования.

Q10. Какие языки относятся к языкам описания архитектуры?
Язык UML
Язык ADL.
SDL.
Язык C++.

Q11. Содержит ли стандарт ISO/IEC/IEEE 42010:2011 требования по аспектно-ориентированному распределению интересов по архитектурным видам?
Да, содержит.
Содержит рекомендации, а не требования.
Нет, не содержит.

Заключение

Во втором издании учебного пособия раскрывается специфика предметной области «Архитектура автоматизированных систем» по зарубежным (в основном) и российским источникам, включающим стандарты, монографии, статьи и отчёты, представленным в Интернете. Учебный материал пособия распределён по четырём главам. Содержание трёх первых глав идентично содержанию первого издания, а четвёртая глава содержит информацию о совершенствовании нормативной базы архитектурного моделирования за последнее десятилетие.

За время прошедшее после первого издания основная суть архитектурного моделирования не изменилась, а именно, в архитектурном моделировании должно осуществляться формирование архитектурного описания на основе «визуальных моделей», интегрированных в систему «архитектурных видов» исходя из совокупности «точек зрения» «заинтересованных сторон», у каждой из которых имеются определённые «интересы» и их (эти интересы) необходимо объективировать (овеществить) в разрабатываемой системе.

В четвертой главе, расширяющей первое издание пособия, особое внимание уделяется нормативному совершенствованию стандарта IEEE-1471, которое закреплено в стандарте **ISO/IEC/IEEE 42010:2011** и его русифицированной версии **ГОСТ Р 57100—2016. С достаточной детальностью** раскрыты причины: расширения ответственности нормативов архитектурного моделирования **(от систем, интенсивно использующих программное обеспечение до систем любого типа**); введения дополнительных структур в концептуальную модель (фреймворк) стандарта IEEE-1471; дополнения новой версии концептуальной модели уточнениями в виде совокупности подчинённых моделей; уточнения терминологии. Ещё одним расширением материалов первого издания является включение Приложения, которое содержит шаблон архитектурного описания системы.

Особое внимание уделяется вопросам качества АС, языкам описания архитектур и методам их проектирования, а также вопросам оценки и документирования результатов архитектурного моделирования. Обобщённо представляются

идеи аспектно-ориентированного анализа и проектирования АС. Каждая из глав заканчивается списком контрольных вопросов, на каждый из которых приведён ряд потенциальных ответов.

Список использованных источников

[1] Allen R. and D. Garlan. A Formal Approach to Software Architectures. // Proceeding, IFIP 92, Elsevier, 1992, pp. 134–141.

[2] Aspectj home page. Xerox PARC, USA. http://aspectj.org/.

[3] Architecture and Architecture Modeling Techniques. hpttp://www.agiledata.org/ essays/enterpriseArchitectureTechniques.html

[4] Agile Architectural Modeling. http://www.agilemodeling.com/essays/ agileArchitecture.htm

[5] Bass L. et al. Reasoning Frameworks. (CMU/SEI-2005-TR-007), Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005.

[6] Bass L. et al. Software Architecture in Practice. Addison Wesley, Boston, MA, USA, 2003.

[7] Bittner K.and I. Spence. Use-Case modeling, Addison-Wesley, 2002.

[8] Building Core Ontologies. White paper of the DELOS Working Group on Ontologies Harmonization, 2003.

[9] Buschmann E et al. Pattern-Oriented Software Architecture: A System of Patterns. John Wiley & Sons, 1996.

[10] Booch G. Software Architecture: The Next Step. // Proceeding, 1st European Workshop Software Architecture (EWSA 04), Springer, 2004.

[11] Booch G. The Limits of Software. http://www.booch.com/ architecture/ blog.jsp? part=Papers

[12] Booch G. The Complexity of Programming Models. http://www.booch.com/ architecture/ blog.jsp? part=Papers

[13] Booch G. Collaborative Development Environments. http://www.booch.com/ architecture / blog.jsp?part=Papers

[14] Booch G. Quantitative Observation and Theoretical Construction in Software Architecture. http://www.booch.com/ architecture/blog.jsp?part=Papers

[15] Charette R.N. Why software falls. IEEE Spectrum,vol 42,#9, 2005, pp. 36-43.

[16] Clarke S. and R. J. Walker. Composition patterns: An approach to designing reusable aspects. // Proceeding, InternationalConferenceonSoftwareEngineering, 2001, pp. 5–14.

[17] Clements P. et al., Documenting Software Architectures: Views and Beyond, Addison-Wesley, 2002.

[18] P. Clements P., R. Kazman and M. Klein, Evaluating Software Architecture. Addison-Wesley, 2002.

[19] Clements P. and L. Northrop. Software Product Lines: Practice and Patterns. Addison-Wesley, 2002.

[20] P.Clements et al. A practical method for documenting software architectures. http: //www-2.cs.cmu.edu/afs/cs/project/ able/ftp/icse03-dsa/submitted.pdf

[21] Clements P. et al. Tutorial F3: Documenting Software Architectures: Views and Beyond.// Proceeding, International Conference on Software Engineering, 2003.

[22] Cockburn A. Agile Software Development, Addison-Wesley, 2002,

[23] Dikel D.M., D. Kane and J.R. Wilson. Software Architecture: Organizational Principles and Patterns.— Prentice Hall.— 2001.

[24] Eeles P. Layering strategies. http://www/ibm.com/developerworks/rational/ library/4699.html

[25] Eeles P. Capturing Architectural Requirements. http://www/ibm.com/developerworks/ rational/library/4706.html

[26] Eeles P. What is a software architecture? http://www/ibm.com/developerworks/ rational/library/feb06/eeles

[27] Eeles P. Characteristics of a software architect. http://www/ibm.com/developerworks/ rational/library/mar06/eeles

[28] Eeles P. The benefits of software architecting. http://www/ibm.com/developerworks/ rational/library/may06/eeles

[29] Fielding R. T. Architectural styles and the design of network-based software architecture.// Dissertation, university of California, Irvin, 2000. www.ics.uci.edu/~fielding/ pubs / dissertation/fielding_cv_2000.htm

[30] Gamma, E., R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

[31] Garlan D. and M. Shaw. An Introduction to Software Architecture. Advances in Software Engineering and Knowledge Engineering, vol. 1, World Scientific, 1993, pp. 1–39.

[32] Garlan D. Software architecture: a Roadmap The Future of Software Engineering. A. Finkekstein (Ed), ACM Press, 2000

[34] Hofmeister C., R. Nord and D. Soni, Applied Software Architecture. Addison-Wesley, 1999.

[35] Jacobson I., K. Palmkvist and S. Dyrhage, Systems of Interconnected Systems. // Report on Object-Oriented Analysis and Design (ROAD), May-June 1995, vol. 2, no. 1.

[36] Jucan G. A 3D Software Architecture Framework www.opendatasys.com/ 3D_Frmwk.html

[37] Katera M. and S. Katz. Architectural views of aspects. // Proceedin, International Conference on Aspect-oriented Software Development,2003, pp. 1–10.

[38] Kazman R. et al. An Architectural Analysis Case Study: Internet Information Systems

[39] Kazman R. et al. SAAM: A Method for Analyzing the Properties of Software Architectures // Proceeding, 16th Int'l Conf. Software Eng. (ICSE 94), IEEE CS Press, 1994, pp. 81–90.

[40] Kroll P. The Spirit of the RUP. The Rational Edge, 2001.

[41] Kroll P. and Ph. Kruchten. The Rational Unified Process Made Easy: A Practitioners Guide to the RUP. Addison-Wesley, 2003.

[42] Kruchten Ph. The 4+1 View Model of Software Architecture. IEEE Software, vol. 12, no. 6, 1995, pp. 42–50.

[43] Kruchten Ph. The Rational Unified Process-An Introduction. Addison-Wesley, 1998.

[44] Kruchten Ph., Henk Obbink, Judith Stafford. The Past, Present, and Future for Software Architecture. IEEE Software, IEEE Computer Society, 2006

[45] Leffingwell D. and D. Widrig, Management Software Requirements: A Unified Approach. Addison-Wesley, 1999.

[46] Luders F. Architectural Styles in Component-Based Software Engineering. www.idt.mdh.se/kurser/phd/CBSE/reports/Final_reports/report_05.pdf

[47] May N. A survey of software architecture viewpoint models. // Proceeding, The Sixth Australasian Workshop on Software and System Architectures (AWSA

2005), Swinburne University of Technology, Melbourne, Australia., 2005, pp. 13–24.

[48] Monin B. Practical Software Architecture For the Enterprise. www.safe-house.org/ SH-Book/ safe-house-book.html

[49] Norris D. Communicating Complex Architectures with UML and the Rational ADS. // Proceeding, IBM Rational Software Development User Conference, 2004

[50] Obbink H. et al. Report on Software Architecture Review and Assessment (SARA). V1.0, Feb. 2002; www. philippe.kruchten.com/architecture/SARAv1.pdf.

[51] Obbink H. et al. COPA: A Component-Oriented Platform Architecting Method for Families of Software-Intensive Electronic Products (Tutorial). // . Proceeding, 1st Software Product Line Conf. (SPLC1), 2000.

[52] Ommering R. et al. The Koala Component Model for Consumer Electronics. IEEE Trans. Computers, 2000, vol. 33, no. 3.

[53] Perry D.E. and A.L. Wolf. Foundations for the Study of Software Architecture. ACM SIGSOFT Software Eng. Notes, Oct. 1992, pp. 40–52.

[54] Putman J. Architecting with RM-ODP. Prentice Hall, 2000.

[55] Ran A. ARES Conceptual Framework for Software Architecture. Software Architecture for Product Families: Principles and Practice, M, Addison-Wesley, 2000.

[56] Rechtin E. Systems Architecting: Creating and Building Complex Systems. Prentice Hall, 1991.

[57] Rechtin E. and M. Maier, The Art of Systems Architecting. CRC Books, 1997.

[58] W.E. Royce, W. Royce, Software Architecture: Integrating Process and Technology. TRW Quest., 1991, vol. 14, no. 1.

[59] Rotem-Gal-Oz A. Service Oriented Architecture. http://www.rgoarchitects.com/blog/ default.aspx

[60] Rotem-Gal-Oz A. Software Architecture. http://www.rgoarchitects.com/blog/ default.aspx

[61] Selic B. The Pragmatics of Model-Driven Development. IEEE Software, 2004, vol. 20,

[62] Shaw M. and P. Clements, The Golden Age of Software Architecture: A Comprehensive Survey, tech. report CMU-ISRI-06-101, Inst. for Software Research Int'l, Carnegie Mellon Univ., Feb. 2006.

[63] Shaw M. The Coming-of-Age of Software Architecture Research. // Proeeding,. 23rd Int'l Conf. Software Eng., IEEE CS Press, 2001, pp. 656–664.

[64] Shaw M. and P. Clements. A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems. // Proceeding, COMPSAC 97: Int'l Computer Software and Applications Conf., IEEE CS Press, 1997, pp. 6–13.

[65] Shaw M. and D. Garlan, Software Architecture: Perspectives on an Emerging Discipline, Prentice Hall, 1996.

[66] Soley R. Model-Driven Architecture. Object Management Group, 2000.

[67] Sommerville I.Software Engineering. Addison Wesley, Boston, MA, USA, 6th edition, 2000.

[68] Soni D. et al. Software architecture in industrial applications.// Ptroceedi8ng, International Conference on Software Engineering, pages 196–207, 1995.

[69] Sosnin P. Question-Answer Processor for Cooperative Work in Human-Computer Environment. // Proceeding, the 2 International IEEE conference Intelligent System 2004 p.452-456/.

[70] Tang A, J. Han and P.Chen, A Comparative Analysis of Architecture Frameworks // Technical Report: SUTIT-TR2004.01,CeCSES Centre Report: SUT.CeCSES-TR001/ 2004.

[71] The Standish group, Charting the Seas of Information Technology-Chaos, The Standish Group International, 1994.

[72] Wang G. and C. K. Fung Architecture Paradigms and Their Influences and Impacts on Component-Based Software Systems // Preceeding, 37 Hawaii International Conference on Systems Sciences, 2004, 10pp.

[73] Witt F., Baker and E. Merritt, Software Architecture and Design: Principles, Models and Methods. Van Nostrand Reinhold, 1994.

[74] Wood E. Using Architectural Perspectives. http://www.eoinwoods.info/index.php?page=articles

[75] Wood E. Software Architecture: Stakeholders, Viewpoints, Perspectives. http://www. eoinwoods .info/index.php?page=articles

[76] Wood E. The Past, Present and Future of Software Architecture. http://www. eoinwoods .info/ index.php?page=articles

[77] Wood E. Architectural Evaluation for Fun and Profit! http://www.eoinwoods.info/ index.php? page=articles

[78] Ссылки:

[79] DoD Architecture Framework and Software Architecture Workshop Report – March 2003 http://www.ichnet.org/DODAF%20SEI%20report.pdf

[80] FEAF: FEA Frameworks // www.eaframeworks.com/FEAF/index.html

[81] MODAF: Ministry of Defence Architecture Framework// www.telelogic.com/standards/modaf.cfm

[82] TOGAF: The Open Group Architectural Framework// www.togaf.org/

[83] G. Booch http://www.booch.com/ architecture

[84] I. Jacobson http://www. Ivarjacobson.com

[85] Дубина О. Обзор паттернов проектирования . http://www.citforum.ru /SE/ project/pattern

[86] Аналитический отчет. Анализ международного опыта стандартизации архитектуры программного обеспечения государственных информационных систем . http://projects.economy.gov.ru/pms/public/ PublicWorkProducts.aspx?projectid=695269fc-eaeb-474c-b0e3-99b905fa17d1

Стандарты:

[87] International Standard ISO/IEC 9126 – 1:2001 Software engineering - - Product quality // www.iso.org/iso/en

[88] International Standard ISO/IEC 12207 – Software Life Cycle Process// www.abelia/com/docs/12207cpt.pdf

[89] ISO/IEC 10746:1995, Reference Model of Open Distributed Processing (RM-ODP). ITU Rec. X901, 1995.

[90] IEEE--**1471**. Recommended Practice for Architectural Description of Software-Intensive Systems. Institute of Electrical and Electronics Engineers, Sept. 2000. IEEE Std 1471-2000.

Список использованных источников (Глава 4)

[91] Соснин, П. И. Архитектурное моделирование автоматизированных систем: учебное пособие / П. И. Соснин. – Ульяновск : УлГТУ, 2007. – 146 с.

[92] Updating IEEE 1471: Architecture frameworks and other topics with David Emery. Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008). preprint, slides

[93] ISO/IEC/IEEE 42010:2011 Systems and software engineering - Architecture description, pp. 1-46, Dec. 2011

[94] ГОСТ Р 57100-2016/ISO/IEC/IEEE 42010:2011 Системная и программная инженерия. Описание архитектуры, http://docs.cntd.ru/document/1200139542

[95] Clarke P. and O'Connor R.V. The situational factors that affect the software development process: Towards a comprehensive reference framework, Journal of Information Software and Technology, № 54(5), pp. 433-447, 2012

[96] Bedjeti A., Lago P., Lewis G. A., Boer R. D. D. and Hilliard R., Modeling Context with an Architecture Viewpoint, 2017 IEEE International Conference on Software Architecture (ICSA), pp. 117-120, 2017.

[97] M. A. Boden M. A. Conceptual Spaces, P. Meusburger et al., Eds., Milieus of Creativity, Knowledge and Space 2, Springer Science + Business Media B.V., pp. 235-243, 2009

[98] Hilliard R. Lessons from the fundamental unity of architecting In Software Engineering in the Systems Context, editors: Ivar Jacobson and Harold 'Bud' Lawson, 2015

[99] J. Schenkhuizen. Consistent Inconsistency Management: a Concern-Driven Approachhttps://dspace.library.uu.nl/bitstream/handle/1874/334223/thesisv1_digitaal.pdf

[100] Полис Г., Огастин Л., Мадхар Д. Разработка программных проектов: на основе Rational Unified Process (RUP). – М.: ООО «Бином-Пресс», 2009

[101] Кватрани Т., Палистрант Д. Визуальное моделирование с помощью IBM Rational Software Architect и UML. Пер. с англ. – М.: КУДИЦ-ПРЕСС. – 2007

[102] Rathfelder C. and Groenda H. Towards an Architecture Maintainability Maturity Model (Softwaretechnik-Trends vol 28 4) pp 3-7, 2008

[103] Van Heescha U., Avgerioua P. and R. Hilliard. A documentation framework for architecture decisions. The Journal of Systems & Software, **85**(4), pp. 795-820. 2012

[104] Van Heesch U, Avgeriou P. and Hilliard R. 2012. Forces on Architecture Decisions - A Viewpoint. In Proceedings of the 2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture (WICSA-ECSA '12). IEEE Computer Society, Washington, DC, USA, 101-110)

[105] G. Mustapic, A. Wall, C. Norstrom, I. Crnkovic, K. Sandstrom,J. Froberg, and J. Andersson, "Real world influences on software architecture-interviews with industrial system experts," in Fourth Working IEEE/IFIP Conference on Software Architecture, pp. 101–111, 2004.

[106] Hilliard R. Using aspects in architectural description,h LNCS, volume 4765, pp-. 139-154, 2007.