КАЗАНСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Кафедра информационных систем

А.Ф. ГАЛИМЯНОВ, Ф.А. ГАЛИМЯНОВ

АРХИТЕКТУРА ИНФОРМАЦИОННЫХ СИСТЕМ

Учебно-методическое пособие

Казань – 2019

УДК 004.032.26

Принято на заседании кафедры информационных систем
Протокол № 8 от 21 марта 2019 года

Рецензенты:
кандидат физико-математических наук,
доцент кафедры теории функций и приближений КФУ **Ю.Р. Агачев;**
кандидат технических наук,
с.н.с. научно-исследовательского института АН РТ «Прикладная семиотика» **А.Р. Гатиятуллин**

Галимянов А.Ф., Галимянов Ф.А.
Архитектура информационных систем / А. Ф. Галимянов, Ф. А. Галимянов. – Казань: Казан. ун-т, 2019. – 117 с.

Учебное пособие посвящено изложению основ архитектуры информационных систем.

Адресовано, в первую очередь, студентам-бакалаврам 2 курса направления 09.03.02 – «Информационные системы и технологии».

Оглавление

1. АРХИТЕКТУРА СИСТЕМЫ

По определению в [18], **архитектура системы** — *принципиальная организация системы, воплощенная в её элементах, их взаимоотношениях друг с другом и со средой, а также принципы, направляющие её проектирование и эволюцию.* Понятие архитектуры в значительной мере субъективно и имеет множество противоречивых толкований; в лучшем случае оно отображает общую точку зрения команды разработчиков на результаты проектирования системы. Для более подробного описания принципов построения архитектуры стандарт ISO/IEC/IEEE 42010-2011 вводит следующие понятия[8]:2.

Архитектурная группа описаний (англ. architectural view) — представление системы в целом с точки зрения связанного набора интересов. Каждая группа описаний относится к одному или более стейкхолдеру. Термин «группа описаний» употребляется для выражения архитектуры системы при некотором методе описания.

Архитектурное описание (англ. architectural description) — рабочий продукт, использующийся для выражения архитектуры.

Архитектурный подход (англ. architectural framework) — соглашения, принципы и практики для описания архитектуры, установленные для конкретной области применения и/или конкретным сообществом стейкхолдеров.

Архитектурный метод описания (англ. architectural viewpoint) — спецификация соглашений для конструирования и применения группы описаний. Шаблон или образец, по которому разрабатываются отдельные группы описаний посредством установления назначений и аудитории для группы описаний, а также приемы их создания и анализа. Метод описания устанавливает соглашения, по которым группа описаний создается, отображается и анализируется. Тем самым метод описания определяет языки (включая нотации, описания или типы продуктов), применяемые для

определения группы описаний, а также все связанные методы моделирования или приемы анализа, применяемые к данным представлениям группы описаний. Данные языки и приемы применяются для получения результатов, имеющих отношение к адресуемым интересам.

Вид модели (англ. model kind) — соглашения по средствам моделирования (например, сети Петри, диаграммы классов, организационные диаграммы и т. д.).

1.1 Виды архитектуры

Свод знаний по системной инженерии (SEBoK) делит архитектуру на логическую и физическую[12]:269.

Логическая архитектура поддерживает функционирование системы на протяжении всего её жизненного цикла на логическом уровне. Она состоит из набора связанных технических концепций и принципов. Логическая архитектура представляется с помощью методов, соответствующих тематическим группам описаний, и как минимум, включает в себя функциональную архитектуру, поведенческую архитектуру и временную архитектуру.

Функциональная архитектура представляет собой набор функций и их подфункций, определяющих преобразования, осуществляемые системой при выполнении своего назначения.

Поведенческая архитектура — соглашение о функциях и их подфункциях, а также интерфейсах (входы и выходы), которые определяют последовательность выполнения, условия для управления или потока данных, уровень производительности, необходимый для удовлетворения системных требований. Поведенческая архитектура может быть описана как совокупность взаимосвязанных сценариев, функций и/или эксплуатационных режимов.

Временная архитектура является классификацией функций системы, которая получена в соответствии с уровнем частоты её исполнения. Временная архитектура включает в себя определение синхронных и асинхронных аспектов

функций. Мониторинг решений, который происходит внутри системы, следует той же временной классификации [12]:287.

Цель проектирования ***физической архитектуры*** заключается в создании физического, конкретного решения, которое согласовано с логической архитектурой и удовлетворяет установленным системным требованиям.

После того, как логическая архитектура определена, должны быть идентифицированы конкретные физические элементы, которые поддерживают функциональные, поведенческие, и временные свойства, а также ожидаемые свойства системы, полученные из нефункциональных требований к системе.

Физическая архитектура является систематизацией физических элементов (элементов системы и физических интерфейсов), которые реализуют спроектированные решения для продукта, услуги или предприятия. Она предназначена для удовлетворения требований к системе и элементам логической архитектуры и реализуется через технологические элементы системы. Системные требования распределяются как на логическую, так и физическую архитектуру. Глобальная архитектура системы оценивается с помощью системного анализа и, после выполнения всех требований, становится основой для реализации системы.

Архитектура может быть зафиксирована с помощью полного архитектурного описания (АО) (см. рисунок). Стандарт ISO/IEC/IEEE 42010-2011 предписывает различать концептуальную архитектуру системы и одно из описаний данной архитектуры, являющееся конкретным продуктом или артефактом.

В сложных системах АО может разрабатываться не только для системы в целом, но и для компонентов системы. Два разных концептуальных АО могут включать группы описаний, которые будут соответствовать одному и тому же методу описания. Хотя системы, описываемые данными двумя группами описаний, будут соотноситься как целое и часть, это не пример множества групп описаний, соответствующих одному методу. Эти АО считаются

отдельными, даже хотя они связаны через системы, которые они описывают[8]:3.

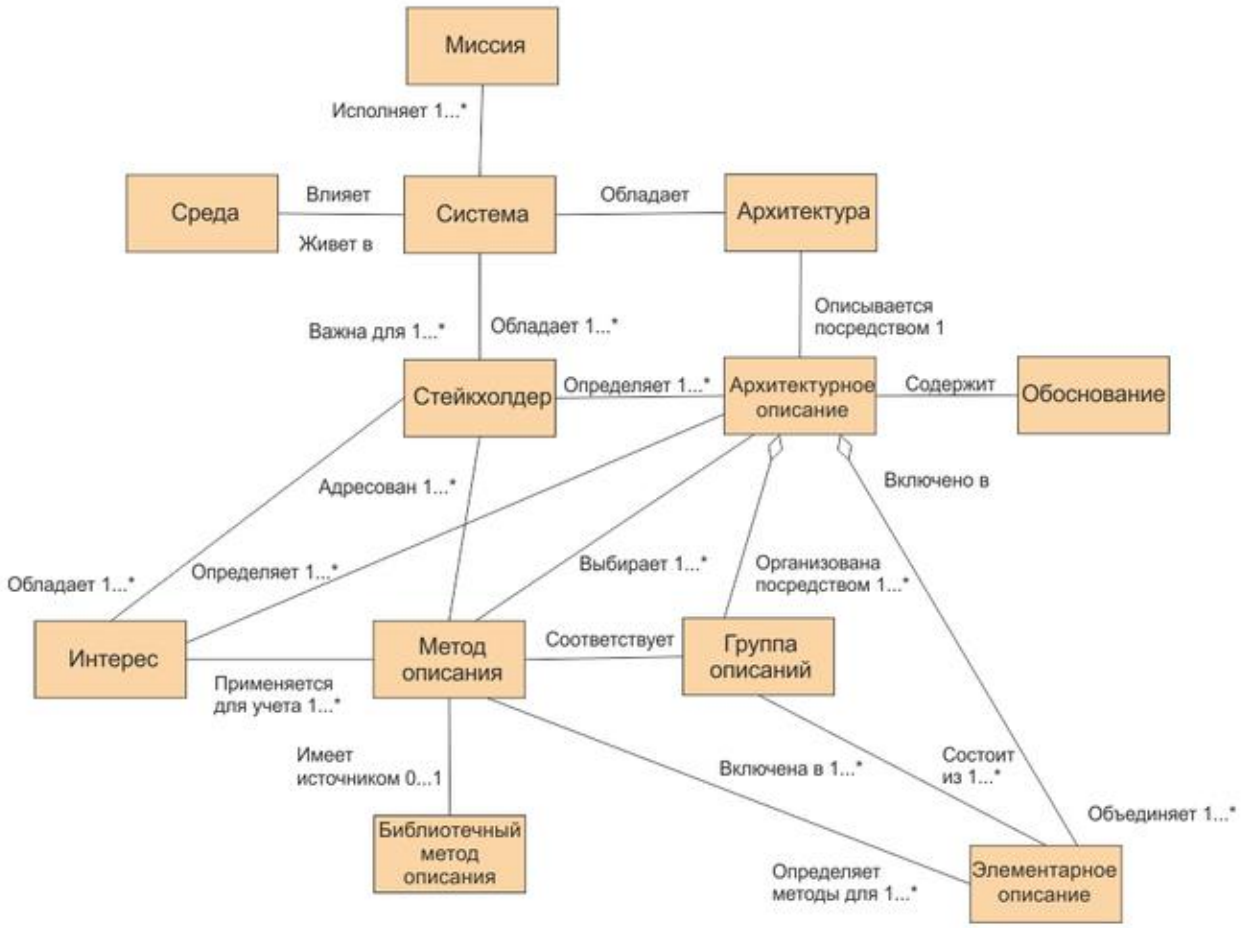


Рис 1. Концептуальная схема архитектурного описания (ISO/IEC 42010)

Концептуальный подход определяет термины и понятия, относящиеся к содержанию и применению АО. На рисунке изображены основные понятия и их взаимосвязи. Все понятия определены в контексте архитектуры определенной системы и соответствующего архитектурного описания. Не нужно предполагать, что у системы существует лишь одна архитектура или что эта архитектура изображается лишь одним архитектурным описанием.

На рисунке прямоугольники изображают классы сущностей.

Линии, соединяющие прямоугольники, изображают связи между сущностями. Связь включает две роли (по одной в каждом направлении). Каждая роль может по желанию быть именована меткой. Роль, направленная от

A к B, [помечена] ближе к B, и наоборот. Например, роли между «системой» и «средой» могут читаться: «„система“ живёт в „среде“» и «„среда“ влияет на „систему“». На рисунке роли обладают арностью 1:1, если не указано иное. Роль может обладать множественной арностью, например, роль, обозначенная как «1..*», применяется для обозначения многих, как в связях «один ко многим» или «многие к одному». Ромб (на конце линии связи) обозначает отношение части целого. Например, «группы описаний» являются частью «архитектурного описания». Эта нотация заимствована из UML.

Рассмотрим каждое составляющее концептуальной схемы подробнее. В контексте рассматриваемой схемы система распространяется на отдельные прикладные программные средства, системы в традиционном смысле, подсистемы, системы систем, продукты, семейства продукции, организации в целом и другие интересующие совокупности.

Система обитает в некоторой среде. Среда некоторой системы может влиять на данную систему. Её среда, или контекст, определяет обстановку и обстоятельства разработки, эксплуатации, политических и иных влияний на данную систему. Такая среда может включать другие системы, взаимодействующие с целевой системой, как напрямую через интерфейсы, так и косвенно иными путями. Такая среда определяет границы, определяющие предмет целевой системы по отношению к другим системам.

У каждой системы есть один или более ***стейкхолдеров***. Каждый стейкхолдер обычно принимает участие в системе, или имеет интересы к данной системе. Интересы предполагают учёт таких аспектов системы как производительность, надежность, безопасность, распределённость и способность к эволюции.

Любая система существует для реализации в своей среде одной или более миссий.

В концептуальном подходе архитектурное описание организовано как одна или более архитектурных групп описаний.

Архитектурное описание выбирает для применения один или более подходящих методов описания. Выбор методов описания обычно основывается на соображениях и интересах заинтересованных сторон, которым адресовано это АО. Определение метода описания может возникать совместно с АО, а может быть определено отдельно. Метод описания, определенный отдельно от АО называется библиотечным методом описания.

Группа описаний может состоять из одного или более архитектурных описаний. Каждое такое архитектурное описание разрабатывается с применением установленных соответствующим ему методов архитектурного описания. Архитектурное описание может входить более чем в одну группу описаний[8]:4-6.

1.2 Типы групп описаний архитектуры

Существует три типа группы описаний: функциональные, логические и физические. Каждая из групп предназначена для описания собственных точек зрения и соответствующего им уровня сложности[10]:224.

Функциональная группа описаний обеспечивает представление с точки зрения пользователей или операторов, которое включает продукты, относящиеся к фазам, сценариям и потокам задач операционной системы. Информационный поток может быть рассмотрен с пользовательского ракурса, также описываются и пользовательские интерфейсы. Примером продуктов, которые могут быть включены в это описание, будут функциональные данные или графики, сценарное описание (включая использование кейсов), блок-схемы задач, организационные диаграммы и схемы информационных потоков[10]:224.

Логическая группа описаний обеспечивает представление с точки зрения руководителя или заказчика. Логическое представление включает продукты, которые определяют системные границы с её окружением и функциональные интерфейсы с внешними системами, также основные функции и поведение системы, потоки информации, внутренние и внешние наборы данных, внутренних и внешних пользователей, и внутренние функциональные интерфейсы. Примером продуктов могут быть блочные диаграммы

функциональных потоков (FFBD), контекстные диаграммы, N2-диаграммы, IDEF0-диаграммы, данные поточных диаграмм и различных стейкхолдеров — характерные продукты (в том числе бизнес-зависимые продукты)[10]:224.

Физическая группа описаний обеспечивает представление с точки зрения проектировщиков. Включает в себя:

- продукты, которые определяют физические границы системы;
- физические компоненты системы и то, как они взаимодействуют и влияют друг на друга;
- внутренние базы данных и структуры данных;
- инфраструктуру информационных технологий (ИТ) системы;
- внешнюю ИТ-инфраструктуру, с которой система взаимодействует;
- требования, необходимые для развития системы.

Продукт может включать в себя физические блок-схемы на довольно высоком уровне детализации, топологии базы данных, интерфейс управления документами и стандарты. Все из трёх типов групп должны присутствовать в каждом описании архитектуры[10]:224.

1.3. Применение архитектурных описаний

Архитектурные описания в ходе жизненного цикла могут различно применяться всеми стейкхолдерами. Такие применения включают, но не ограничиваются, следующим:

- анализ альтернативных архитектур
- деловое планирование перехода от унаследованной архитектуры к новой;
- коммуникация организаций, участвующих в разработке, производстве, установке, эксплуатации и обслуживании систем;
- коммуникация между заказчиками и разработчиками как часть подготовки соглашения;
- критерии для сертификации соответствия реализации данной архитектуре;

- документирование разработки и обслуживания, включая подготовку материалов для хранилищ с целью повторного использования и учебных материалов;
- исходные данные для последующих мероприятий по системному проектированию и разработке;
- исходные материалы для инструментов создания и анализа системы;
- эксплуатационная и инфраструктурная поддержка; управление конфигурацией и ремонт; перепроектирование и обслуживание систем, подсистем и компонентов;
- поддержка планирования и финансирования[8]:9.

2. ИНФОРМАЦИОННАЯ СИСТЕМА

Информационная система (ИС) — система, предназначенная для хранения, поиска и обработки информации, и соответствующие организационные ресурсы (человеческие, технические, финансовые и т. д.), которые обеспечивают и распространяют информацию (ISO/IEC 2382:2015)[7].

Предназначена для своевременного обеспечения надлежащих людей надлежащей информацией[14], то есть для удовлетворения конкретных информационных потребностей в рамках определённой предметной области, при этом результатом функционирования информационных систем является информационная продукция — документы, информационные массивы, базы данных и информационные услуги[17].

3. ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ АРХИТЕКТУРЫ ИНФОРМАЦИОННЫХ СИСТЕМ

На основе приведенных определений рассмотрим определение архитектуры информационных систем. Определений архитектуры информационных систем много. На сайте SEI (Software Engineering Institute)

имеется специальный раздел, посвященный определениям архитектуры, где их собрано более ста. Различные источники дают следующие оперделения:

- архитектура — организационная структура системы;
- архитектура информационной системы — концепция, определяющая модель, структуру, выполняемые функции и взаимосвязь компонентов информационной системы;
- архитектура — базовая организация системы, воплощенная в ее компонентах, их отношениях между собой и окружением, а также принципы, определяющие проектирование и развитие системы;
- архитектура — набор значимых решений по поводу организации системы программного обеспечения, набор структурных элементов и их интерфейсов, при помощи которых компонуется система вместе с их поведением, определяемым во взаимодействии между этими элементами, компоновка элементов в постепенно укрупняющиеся подсистемы, а также стиль архитектуры, который направляет эту организацию (элементы и их интерфейсы, взаимодействия и компоновку);
- архитектура программы или компьютерной системы — структура или структуры системы, которые включают элементы программы, видимые извне свойства этих элементов и связи между ними;
- архитектура — структура организации и связанное с ней поведение системы. Архитектуру можно рекурсивно разобрать на части, взаимодействующие посредством интерфейсов, связи, которые соединяют части и условия сборки частей. Части, взаимодействующие через интерфейсы, включают классы, компоненты и подсистемы;
- архитектура программного обеспечения системы или набора систем состоит из всех важных проектных решений в отношении структур программы и взаимодействий между этими структурами, составляющих системы. Проектные решения обеспечивают желаемый набор свойств, которые должна поддерживать система, чтобы быть успешной. Проектные решения

предоставляют концептуальную основу для разработки системы, ее поддержки и обслуживания.

Для того чтобы разобраться в этом многообразии определений, выделим наиболее существенные стороны архитектуры ИС.

Основным критерием выбора архитектуры и инфраструктуры ИС в условиях рыночной экономики является минимизация совокупной стоимости владения системой. Из этого следуют два основных тезиса.

1. В проектах построения информационных систем, для которых важен экономический эффект, необходимо выбирать архитектуру системы с минимальной совокупной стоимостью владения.

2. Совокупная стоимость владения ИС состоит из плановых затрат и стоимости рисков. Стоимость рисков определяется стоимостью бизнес-рисков, вероятностями технических рисков и матрицей соответствия между ними. Матрица соответствия определяется архитектурой ИС.

Термин «архитектура информационной системы» обычно довольно согласованно понимается специалистами на уровне подсознания, но при этом и определения этого понятия неоднозначны. Имеются два основных класса определений архитектур — «идеологические» и «конструктивные».

Основные идеологические определения архитектуры ИС таковы:

1. Архитектура ИС — набор решений, наиболее существенным образом влияющих на совокупную стоимость владения системой;

2. Архитектура ИС — набор ключевых решений, неизменных при изменении бизнес-технологии в рамках бизнес-видения.

Эти определения объединяет то, что если ключевое решение приходится изменять при изменении бизнес-технологии в рамках бизнес-видения, то резко возрастает стоимость владения системой. Как следствие возникает понимание важности принятия архитектуры системы как стандарта предприятия ввиду значимости архитектурных решений и их устойчивости по отношению к изменениям бизнес- технологии. Важный вывод из первого определения — архитектура системы должна строиться еще на стадии технико-экономического

обоснования системы. Конструктивно архитектура обычно определяется как набор ответов на следующие вопросы:

- что делает система?;
- на какие части она разделяется?;
- как эти части взаимодействуют?;
- где эти части размещены?.

Таким образом, архитектура ИС является логическим построением, или моделью, и влияет на совокупную стоимость владения через набор связанных с ней решений по выбору средств реализации, СУБД, операционной платформы, телекоммуникационных средств и т. п. — т. е. через то, что мы называем инфраструктурой ИС. При этом инфраструктура включает решения не только по программному обеспечению, но и по аппаратному комплексу и организационному обеспечению.

3.1. Информационная система как объект архитектуры

Выше было приведено одно из определений информационной системы. Применительно к информационным системам термин «архитектура» используется достаточно часто. Данный термин может относиться к организации, программно-аппаратному комплексу, программной системе или центральному процессору.

Применительно к организации обычно используют понятие корпоративная архитектура (enterprise architecture), при этом выделяются следующие типы архитектур: бизнес-архитектура (Business architecture), ИТ-архитектура (Information Technology Architecture), архитектура данных (Data Architecture), архитектура приложения (Application Architecture) или программная архитектура (Software Architecture), техническая архитектура (nardware Architecture). Совокупность данных архитектур и является архитектурой ИС (рис. 2).

Бизнес-архитектура, или архитектура уровня бизнес-процессов, определяет бизнес-стратегии, управление, организацию, ключевые бизнес-процессы в масштабе предприятия, причем не все бизнес- процессы

реализуются средствами ИТ-технологий. Бизнес-архитектура отображается на ИТ-архитектуру.

ИТ-архитектура рассматривается в трех аспектах:

• обеспечивает достижение бизнес-целей посредством использования программной инфраструктуры, ориентированной на реализацию наиболее важных бизнес-приложений;

• среда, обеспечивающая реализацию бизнес-приложений;

• совокупность программных и аппаратных средств, составляющая информационную систему организации и включающая, в частности, базы данных и промежуточное программное обеспечение.

Рис 2. Архитектура информационной системы

Архитектура данных организации включает логические и физические хранилища данных и средства управления данными. Архитектура данных должна быть поддержана ИТ-архитектурой. В современных ИТ-системах, ориентированных на работу со знаниями, иногда выделяют отдельный тип архитектуры — архитектуру знаний (Knowledge Architecture).

Программная архитектура отображает совокупность программных приложений. Программное приложение — это компьютерная программа, ориентированная на решение задач конечного пользователя. Архитектура приложения — это описание отдельного приложения, работающего в составе ИТ-системы, включая его программные интерфейсы. Архитектура приложения

базируется на ИТ-архитектуре и использует сервисы, предоставляемые ИТ-архитектурой.

Техническая архитектура характеризует аппаратные средства и включает такие элементы, как процессор, память, жесткие диски, периферийные устройства, элементы для их соединения, а также сетевые средства.

Часто нельзя провести четкой границы между ИТ-архитектурой и архитектурой отдельного приложения. В частности в том случае, когда некоторую фупкцию требуется реализовать в нескольких приложений, она может быть перенесена на уровень ИТ-архитектуры. Обычно приложения интегрируются средствами ИТ-архитектуры. Элементы архитектуры данных часто интегрируются в приложения.

3.2. Архитектура и проектирование информационных систем

Архитектурное описание самым тесным образом связано с процессом проектирования ИС, причем в ряде определений термина «архитектура» на этот факт указывается в явном виде. Обычно выделяются пять различных подходов к проектированию, которые называют также стилями проектирования и, по существу, определяют группы методологий разработки ПО:

- календарный стиль — основанный на календарном планировании (Calendar-driven);
- стиль, основанный на управлении требованиями (Requirements-driven);
- стиль, в основу которого положен процесс разработки документации (Documental ion-driven);
- стиль, основанный на управлении качеством (Quality-driven);
- архитектурный стиль (Architecture-driven).

Основой календарного стиля является график работ. Команда разработчиков выполняет работы поэтапно. Проектные решения принимаются из целей и задач конкретного этапа. Слабыми местами данного стиля являются то, что основные решения принимаются исходя из локальных целей, при этом мало внимания уделяется процессу разработки, разработке документации,

созданию стабильных архитектур и внесению изменений. Все это приводит к высокой суммарной стоимости владением в долгосрочном плане. Данный стиль считается морально устаревшим, однако многие организации продолжают его использовать.

Стиль, основанный на управлении требованиями, предполагает, что основное внимание уделяется функциональным характеристикам системы, при этом часто недостаточно внимания уделяется нефункциональным характеристикам, таким как масштабируемость, портабельность, подцерживаемость и другим, определенным в ISO 9126. Проектные решения принимаются преимущественно исходя из локальных целей, связанных с реализацией тех или иных функций. Данный подход достаточно эффективен в случае, если требования определены и не изменяются в процессе проектирования. Основные недостатки данного подхода следующие: недостаточное внимание уделяется требованиям стандарта ISO 9126 (управление качеством); разрабатываемые архитектуры, как правило, не являются стабильными, так как каждая из реализуемых функций отображается на один или несколько функциональных компонентов. Это обстоятельство усложняет добавление к системе новых требований. Данный подход позволяет успешно отслеживать требования в плане реализации требуемой функциональности, однако использование его для долгосрочных проектов является неэффективным.

Стиль, в основу которого положен процесс разработки документации, до настоящего времени продолжает использоваться в правительственных организациях и крупных компаниях. Данный стиль может рассматриваться как вырожденный вариант стиля, основанного на управлении качеством, и ориентирован на разработку документации. В качестве основных недостатков данного подхода выделяются следующие: неоправданно много времени и сил затрачивается на разработку документации в ущерб качеству разрабатываемого кода. При этом создаваемая документация часто не используется ни пользователем, ни заказчиком.

Стиль, основанный на управлении качеством, предполагает самое широкое использование различных мер для отслеживания критичных с точки зрения функционирования параметров. Например, требуется получить время реакции системы на запрос менее одной секунды или обеспечить среднее время между отказами не менее 10 лет. В этом случае данные параметры отслеживаются на всех этапах проектирования систем и часто это делается в ущерб другим характеристикам, таким как масштабируемость, простота сопровождения и т. п. Типовыми проблемами, возникающими при использовании данного стиля, являются следующие: часто оптимизируются не те параметры, которые должны быть в действительности, когда появляются новые требования, бывает очень трудно изменить функциональность системы. Созданные таким образом системы обычно не отличаются высоким качеством архитектурных решений. В целом данный стиль считается консервативным. Его использование может быть оправдано в случае, когда необходимо создать системы с экстремальными характеристиками.

Архитектурный стиль представляет собой наиболее зрелый подход к проектированию. В рамках данного стиля во главу угла ставится создание фреймворков, которые могут быть легко адаптированы ко всем потенциальным требованиям всех потенциальных заказчиком. Отличительная особенность данного стиля состоит в том, что задача проектирования разбивается на две отдельные подзадачи: создание многократно используемого фреймворка и создание конкретного приложения (системы) на его основе. При этом эти две задачи могут решать разные специалисты. Основная цель создания данного стиля — это устранение недостатков стиля, основанного на управлении требованиями. Использование архитектурного стиля позволяет реализовать инкрементное и итеративное проектирование, т.е. оперативно изменять существующую и добавлять новую функциональность.

В процессе проектирования важное значение приобретают атрибуты качества ИС.

Понятие качества ИС соответствует понятию о том, что система успешно справляется со всеми возлагаемыми на нее задачами, имеет хорошие показатели надежности и приемлемую стоимость, удобна в эксплуатации и обслуживании, легко сочетается с другими системами и в случае необходимости может быть модифицирована.

Разные группы пользователей имеют различные точки зрения на характеристики качества ИС. Например, если задать вопрос о том, какой должна быть хорошая ИС, то от пользователя можно получить следующие варианты ответов:

- система имеет хорошую производительность;
- система имеет широкие функциональные возможности;
- система удобна в эксплуатации;
- система надежна.

Менеджер даст скорее всего другие варианты ответов:

- стоимость системы не должна быть изначально очень высокой;
- система не должна быть очень дорогой в эксплуатации;
- система не должна морально устаревать в течение возможно более длительного периода времени и в случае необходимости может быть легко модифицирована.

Для системного администратора наиболее важными могут оказаться такие характеристики системы, как

- надежность и стабильность работы;
- простота администрирования;
- наличие хорошей эксплуатационной документации;
- хорошая поддержка изготовителем.

Другие заинтересованные лица могут иметь свои точки зрения на то, какой должна быть качественная система.

Поскольку в современных ИС ключевой компонентой является программная компонента, а пользователи, работающие с системой, в большинстве случаев взаимодействуют непосредственно с программной

компонентой, поэтому показатели качества информационных и программных систем в значительной степени совпадают.

Для того чтобы построить правильную и надежную архитектуру и грамотно спроектировать интеграцию программных систем, необходимо четко следовать современным стандартам в этих областях. Без этого велика вероятность создать архитектуру, которая неспособна развиваться и удовлетворять растущие потребности пользователей ИТ. В качестве законодателей стандартов в этой области выступают такие международные организации, как SEI (Software Engineering Institute), WWW (консорциум World Wide Web), OMG (Object Management Group), организация разработчиков Java — JCP (Java Community Process), IEEE (Institute of Electrical and Electronics Engineers) и др.

Качество программного обеспечения определяется стандартом ISO 9126 [6] как вся совокупность его характеристик, относящихся к возможности удовлетворять высказанные или подразумеваемые потребности всех заинтересованных лиц.

Различаются понятия внутреннего качества, связанного с характеристиками программного обеспечения (ПО) самого по себе, без учета его поведения; внешнего качества, характеризующего ПО с точки зрения его поведения; и качества ПО при использовании в различных контекстах — того качества, которое ощущается пользователями при конкретных сценариях работы ПО. Для всех этих аспектов качества введены метрики, позволяющие оценить их. Кроме того, для создания добротного ПО существенное значение имеет качество технологических процессов его разработки.

ISO 9126 — это международный стандарт, определяющий оценочные характеристики качества программного обеспечения. Российский аналог стандарта ГОСТ 28195. Стандарт разделяется на четыре части, описывающие следующие вопросы: модель качества, внешние метрики качества, внутренние метрики качества, метрики качества в использовании.

Вторая и третья части стандарта ISO 9126-2,3 посвящены формализации соответственно внешних и внутренних метрик характеристик качества сложных программных систем. В ней изложены содержание и общие рекомендации по использованию соответствующих метрик и взаимосвязей между типами метрик.

Четвертая часть стандарта ISO 9126-4 предназначена для покупателей, поставщиков, разработчиков, сопровождающих, пользователей и менеджеров качества ПС. В ней повторена концепция трех типов метрик, а также аннотированы рекомендуемые виды измерений характеристик.

Модель качества, установленная в первой части стандарта ISO 9126-1, классифицирует качество ПО в шести структурных наборах характеристик:

- функциональность;
- надежность;
- производительность (эффективность);
- удобство использования (практичность);
- удобство сопровождения;
- переносимость.

Перечисленные характеристики, в свою очередь, детализированы подхарактеристиками (субхарактеристиками). Ниже приведены определения этих характеристик и атрибутов по стандарту ISO 9126:2001.

Функциональность (functionality) определяется как способность ПО в определенных условиях решать задачи, нужные пользователям.

Для данной характеристики выделяются следующие субхарактеристики:

- функциональная пригодность;
- точность;
- способность к взаимодействию;
- защищенность;
- соответствие стандартам и правилам.

Функциональная пригодность (suitability) определяется как способность решать нужный набор задач.

Точность (accuracy) — определяется как способность выдавать нужные результаты.

Способность к взаимодействию (interoperability) — способность взаимодействовать с нужным набором других систем.

Соответствие стандартам и правилам (compliance) — соответствие ПО имеющимся отраслевым стандартам, нормативным и законодательным актам, другим регулирующим нормам.

Защищенность (security) — способность предотвращать неавторизированный, т. е. без указания лица, пытающегося его осуществить, и неразрешенный доступ к данным и программам.

Надежность (reliability) — способность ПО поддерживать определенную работоспособность в заданных условиях.

Для данной характеристики выделяются следующие субхарактеристики:

- зрелость (завершенность);
- устойчивость к отказам;
- способность к восстановлению;
- соответствие стандартам надежности (reliability compliance).

Зрелость, завершенность (maturity) — величина, обратная частоте отказов ПО. Обычно измеряется средним временем работы без сбоев и величиной, обратной вероятности возникновения отказа за данный период времени.

Устойчивость к отказам (fault tolerance) — способность поддерживать заданный уровень работоспособности при отказах и нарушениях правил взаимодействия с окружением.

Способность к восстановлению (recoverability) определяется как способность восстанавливать определенный уровень работоспособности и целостность данных после отказа, необходимые для этого время и ресурсы.

Производительность (efficiency), или эффективность, — способность ПО при заданных условиях обеспечивать необходимую работоспособность по отношению к выделяемым для этого ресурсам. Можно определить ее и как

отношение получаемых с помощью ПО результатов к затрачиваемым на это ресурсам всех типов.

Для данной характеристики выделяются следующие субхарактеристики:

- временная эффективность;
- эффективность использования ресурсов;
- соответствие стандартам производительности (efficiency compliance).

Временная эффективность (time behaviour) — способность ПО выдавать ожидаемые результаты, а также обеспечивать передачу необходимого объема данных за отведенное время.

Эффективность использования ресурсов (resource utilisation) — способность решать нужные задачи с использованием определенных объемов ресурсов определенных видов. Имеются в виду такие ресурсы, как оперативная и долговременная память, сетевые соединения, устройства ввода и вывода и пр.

Удобство использования (usability), или практичность, определяется как способность ПО быть удобным в обучении и использовании, а также привлекательным для пользователей.

Для данной характеристики выделяются следующие субхарактеристики:

- понятность;
- удобство работы;
- удобство обучения;
- привлекательность;
- соответствие стандартам удобства использования (usability compliance).

Понятность (understandability) — это показатель, обратный усилиям, которые затрачиваются пользователями на восприятие основ- них понятий ПО и осознание их применимости для решения своих задач.

Удобство работы (operability) — это показатель, обратный усилиям, предпринимаемым пользователями для решения своих задач с помощью ПО.

Удобство обучения (learnability) — показатель, обратный усилиям, затрачиваемым пользователями на обучение работе с ПО.

Привлекательность (attractiveness) — это способность ПО быть привлекательным для пользователей. Этот атрибут добавлен в 2001 г.

Удобство сопровождения (maintainability) определяется как удобство проведения всех видов деятельности, связанных с сопровождением программ.

Для данной характеристики выделяются следующие субхарактеристики:

- анализируемость;
- удобство внесения изменений;
- стабильность;
- удобство проверки;
- соответствие стандартам удобства сопровождения (maintainability compliance).

Анализируемость (analyzability), или удобство проведения анализа, определяется как удобство проведения анализа ошибок, дефектов и недостатков, а также удобство анализа необходимости изменений и их возможных последствий.

Удобство внесения изменений (changeability) — показатель, обратный трудозатратам на выполнение необходимых изменений.

Стабильность (stability) — показатель, обратный риску возникновения неожиданных эффектов при внесении необходимых изменений.

Удобство проверки (testability) — показатель, обратный трудозатратам на проведение тестирования и других видов проверки того, что внесенные изменения привели к нужным результатам.

Переносимость (portability) определяется как способность ПО сохранять работоспособность при переносе из одного окружения в другое, включая организационные, аппаратные и программные аспекты окружения.

Иногда эта характеристика называется в русскоязычной литературе мобильностью. Однако термин «мобильность» стоит зарезервировать для

перевода «mobility» — способности ПО и компьютерной системы в целом сохранять работоспособность при ее физическом перемещении в пространстве.

Для данной характеристики выделяются следующие субхарактеристики:

- адаптируемость;
- удобство установки;
- способность к сосуществованию;
- удобство замены;
- соответствие стандартам переносимости (portability compliance).

Адаптируемость (adaptability) — способность ПО приспосабливаться к различным окружениям без проведения для этого действий помимо заранее предусмотренных.

Удобство установки (installability) — способность ПО быть установленным или развернутым в определенном окружении.

Способность к сосуществованию (coexistence) — способность ПО сосуществовать с другими программами в общем окружении, деля с ними ресурсы.

Удобство замены (replaceability) другого ПО данным определяется как возможность применения данного ПО вместо других программных систем для решения тех же задач в определенном окружении.

Перечисленные атрибуты относятся к внутреннему и внешнему качеству ПО согласно ISO 9126. Для описания качества ПО при использовании стандарт ISO 9126-4 предлагает другой, более узкий набор характеристик:

- эффективность;
- продуктивность;
- безопасность;
- удовлетворение пользователей.

Эффективность (effectiveness) — способность ПО предоставлять пользователям возможность решать их задачи с необходимой точностью при использовании в заданном контексте.

Продуктивность (productivity) — способность ПО предоставлять пользователям определенные результаты в рамках ожидаемых затрат ресурсов.

Безопасность (safety) — способность ПО обеспечивать необходимо низкий уровень риска нанесения ущерба жизни и здоровью людей, бизнесу, собственности или окружающей среде.

Удовлетворение пользователей (satisfaction) — способность ПО приносить удовлетворение пользователям при использовании в заданном контексте.

Одним из широко применяемых методологией проектирования и описания логической архитектуры является методология функционального моделирования.

4. МЕТОДОЛОГИЯ МОДЕЛИРОВАНИЯ ПРОЦЕССОВ

В настоящее время для описания бизнес-процессов существует множество методологий (IDEF0, IDEF3, DFD, WORKFLOW, UML, ARIS и другие) и инструментальных средств (BPWin, ERWin, PowerDesigner и другие).

В структурном и объектно-ориентированном анализах используются средства, моделирующие в форме диаграмм определенного вида деловые процессы и отношения между данными в системе. Этим средствам соответствуют определенные виды системных моделей, наиболее распространены среди них следующие:

- IDEF (Integrated Definition) - семейство структурных моделей и соответствующих им диаграмм;
- DFD (Data Flow Diagrams) - диаграммы потоков данных;
- ERD (Entity-Relationship Diagrams) - диаграммы «сущность-связь»;
- Workflow - технология управления потоками работ;
- BPMN (Business Process Modeling Notation);

- средства имитационного моделирования, основанные на математическом аппарате раскрашенных сетей Петри (Color Petri Nets, CPN);
- объектно-ориентированные методологии на основе унифицированного языка моделирования UML;
- интегрированные средства и методологии широкого назначения, например ARIS.

4.1. Семейство стандартов структурного моделирования IDEF

Одними из самых известных и широко используемых методологий в области моделирования бизнес-процессов являются методологии семейства IDEF. Семейство IDEF появилось в конце 60-х гг. XX в. под названием SADT (Structured Analysis and Design Technique) В настоящее время оно включает следующие стандарты.

1) IDEF0 - методология функционального моделирования. Используется для создания функциональной модели, отображающей структуру и функции системы, а также потоки информации и материальных объектов, связывающих эти функции.

2) IDEF1 - методология моделирования информационных потоков внутри систем, позволяющая отображать их структуру и взаимосвязи. Методология применяется для построения информационной модели, отображающей структуру и содержание информационных потоков, необходимых для поддержки функций системы.

3) IDEF1X (IDEF1X Extended) - методология построения реляционных информационных структур. IDEF1X относится к типу методологий «сущность—связь» и, как правило, используется для моделирования реляционных баз данных, имеющих отношение к рассматриваемой системе.

4) IDEF2 - методология динамического моделирования развития систем, которая позволяет построить динамическую модель меняющихся во времени поведения функций, информации и ресурсов системы. В настоящее время известны алгоритмы и их компьютерные реализации, позволяющие

превращать набор статических диаграмм IDEF0 в динамические модели, построенные на базе «раскрашенных сетей Петри» (CPN- Color Petri Nets).

5) IDEF3 — методология документирования процессов, происходящих в системе. С помощью IDEF3 описываются сценарий и последовательность операций для каждого процесса. Функция в диаграмме IDEF3 может быть представлена в виде отдельного процесса средствами IDEF3.

6) IDEF4 — методология построения объектно-ориентированных систем. Средства IDEF4 позволяют наглядно отображать структуру объектов и заложенные принципы их взаимодействия, позволяя тем самым анализировать и оптимизировать сложные объектно-ориентированные системы.

7) IDEF5 — методология онтологического исследования сложных систем. С помощью этой методологии онтология системы описывается при помощи определенного словаря терминов и правил, на основе которых могут быть сформированы достоверные утверждения о состоянии рассматриваемой системы в некоторый момент времени. На основе этих утверждений формируются выводы о дальнейшем развитии системы и производится ее оптимизация.

8) IDEF6 (Design Rational Capture - метод рационального представления процесса проектирования информационных систем, позволяющий обосновать необходимость проектируемых моделей, выявить причинно-следственные связи и отразить это в итоговой документации системы.

9) IDEF8 (User Interface Modeling) - Human - System Interaction Design Method - метод проектирования взаимодействия пользователей с системами различной природы (не обязательно информационно - вычислительными).

10) IDEF9 (Business Constraint Discovery Method) - метод изучения и анализа бизнес-систем в терминах «ограничений». Ограничения инициируют результат, руководят и ограничивают поведение объектов и агентов (автономных программных модулей) для выполнения целей или намерений системы.

11) IDEF14 (Network Design Method) - метод проектирования вычислительных сетей, позволяющий устанавливать требования, определять сетевые компоненты, анализировать существующие сетевые конфигурации и формулировать желаемые характеристики сети.

Анонсированные корпорацией KBSI (Knowledge Based System Inc.) методы IDEF7 (Information System Audit Method), IDEF10 (Information Artifact Modeling) и IDEF12 (Organization Design) не получили дальнейшего развития.

С помощью методологий семейства IDEF можно эффективно отображать и анализировать модели деятельности широкого спектра сложных, в том числе и информационных систем. К настоящему времени наибольшее распространение и применение имеют методологии IDEF0 и IDEF1 (IDEF1X), получившие в США статус федеральных стандартов.

4.2. Функциональное моделирование бизнес-процессов в IDEF0

IDEF0 может быть использована для моделирования широкого класса систем. Для новых систем применение IDEF0 направлено на определение требований и указание функций для последующей разработки системы, отвечающей поставленным требованиями и реализующей выделенные функции. Применительно к уже существующим системам IDEF0 может быть использована для анализа функций, выполняемых системой, и отображения механизмов, посредством которых эти функции выполняются. Результатом применения IDEF0 к некоторой системе является модель этой системы, состоящая из иерархически упорядоченного набора диаграмм, текста документации и словарей, связанных друг с другом с помощью перекрестных ссылок. Двумя наиболее важными компонентами, из которых строятся диаграммы IDEF0, являются бизнес-функции, или работы (блоки), и данные, или объекты (дуги), связывающие между собой работы и отображающие взаимодействия и взаимосвязи между ними.

4.3. Синтаксис графического языка IDEF0

Набор структурных компонентов языка, их характеристики и правила, определяющие связи между компонентами, представляют собой синтаксис языка. Компоненты синтаксиса IDEF0 - блоки, стрелки, диаграммы и правила.

Блоки представляют функции, определяемые как деятельность, процесс, операция, действие или преобразование. Блок описывает функцию. Внутри каждого блока помещается его имя и номер. Имя должно быть активным глаголом или глагольным оборотом, описывающим функцию. Номер блока размещается в правом нижнем углу. Номера блоков используются для их идентификации на диаграмме и в соответствующем тексте.

Требования к блокам в IDEF0 следующие:

1) размеры блоков должны быть достаточными для того, чтобы включить имя блока;

2) блоки должны быть прямоугольными, с прямыми углами;

3) блоки должны быть нарисованы сплошными линиями.

Стрелки представляют данные или материальные объекты, связанные с функциями.

Стрелка формируется из одного или более отрезков прямых и наконечника на одном конце. Сегменты стрелок могут быть прямыми или ломаными. Стрелки не представляют поток или последовательность событий, как в традиционных блок-схемах потоков или процессов. Они лишь показывают, какие данные или материальные объекты должны поступить на вход функции для того, чтобы эта функция могла выполняться.

Требования к стрелкам в IDEF0 следующие:

1) ломаные стрелки изменяют направление только под углом 90 градусов;

2) стрелки должны быть нарисованы сплошными линиями различной толщины;

3) стрелки могут состоять только из вертикальных или горизонтальных отрезков, отрезки, направленные по диагонали, не допускаются;

4) концы стрелок должны касаться внешней границы функционального блока, но не должны пересекать ее;

5) стрелки должны присоединяться к блоку на его сторонах, присоединение в углах не допускается;

6) каждая стрелка должна быть помечена существительным или боротом существительного (например, «отчет об испытаниях», «конструкция детали», «бюджет», «конструкторские требования» и др.).

Стрелки, представляя множества объектов, в зависимости от того, в какую грань блока (прямоугольника работы) они входят или из какой грани выходят, делятся на пять видов:

- входа (входят в левую грань работы) - изображают данные или объекты, изменяемые в ходе выполнения работы;
- управления (входят в верхнюю грань работы) - изображают правила и ограничения, согласно которым выполняется работа;
- выхода (выходят из правой грани работы) - изображают данные или объекты, появляющиеся в результате выполнения работы;
- механизма (входят в нижнюю грань работы) - изображают ресурсы, необходимые для выполнения работы, но не изменяющиеся в процессе работы (например, оборудование, людские ресурсы и т.д.);
- вызова (выходят из нижней грани работы) - изображают связи между разными диаграммами или моделями, указывая на некоторую диаграмму, где данная работа рассмотрена более подробно.

Входные дуги изображают объекты, используемые и преобразуемые функциями. Управленческие дуги представляют информацию, управляющую действиями функций. Выходные дуги изображают объекты, в которые преобразуются входы. Дуги механизмов IDEF0 изображают физические аспекты функции (склады, людей, организации, приборы). Таким образом,

стороны блока графически сортируют объекты, изображаемые касающимися блока дугами.

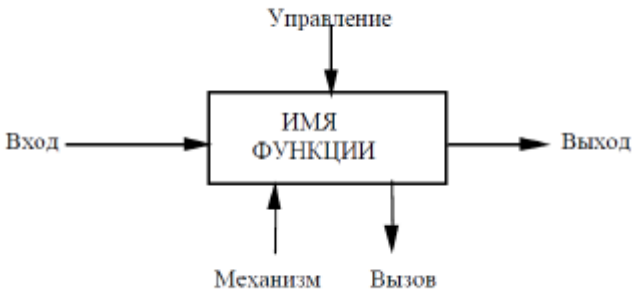


Рис. 3 Блок IDEF0

4.4. Семантика языка IDEF0

Семантика определяет содержание (значение) синтаксических компонентов языка и способствует правильности их интерпретации. Интерпретация устанавливает соответствие между блоками и стрелками с одной стороны и функциями и их интерфейсами - с другой.

Семантические правила блоков и стрелок:

1) имя блока должно быть активным глаголом или глагольным оборотом;

2) каждая сторона функционального блока должна иметь стандартное отношение блок/стрелки:

- входные стрелки должны связываться с левой стороной блока;
- управляющие стрелки должны связываться с верхней стороной блока;
- выходные стрелки должны связываться с правой стороной блока;
- стрелки механизма (кроме стрелок вызова) должны указывать вверх и подключаться к нижней стороне блока;
- стрелки вызова механизма должны указывать вниз, подключаться к нижней стороне блока, и помечаться ссылкой на вызываемый блок;

3) сегменты стрелок, за исключением стрелок вызова, должны помечаться существительным или оборотом существительного, если только единственная метка стрелки несомненно не относится к стрелке в целом;

4) чтобы связать стрелку с меткой, следует использовать "тильду";

5) в метках стрелок не должны использоваться следующие термины: функция, вход, управление, выход, механизм, вызов.

Первая диаграмма в иерархии диаграмм IDEF0 всегда изображает функционирование системы в целом. Такие диаграммы называются контекстными. В контекст входят описание цели моделирования, области (описание того, что будет рассматриваться как компонент системы, а что - как внешнее воздействие) и точка зрения (позиция, с которой будет строиться модель). Обычно в качестве точки зрения выбирается точка зрения лица или объекта, ответственного за работу моделируемой системы в целом. После описания контекста строят следующие диаграммы в иерархии. Каждая последующая диаграмма является более подробным описанием (декомпозицией) одной из работ на вышестоящей диаграмме (рис. 4.). Описание каждой подсистемы проводится аналитиком совместно с экспертом предметной области. Обычно экспертом является человек, отвечающий за эту подсистему, и поэтому дос конально знающий все ее функции. Таким образом, вся система разбивается на подсистемы до нужного уровня детализации, и получается модель, аппроксимирующая систему с заданным уровнем точности. Получив модель, адекватно отображающую текущие бизнес-процессы (так называемую модель AS-IS), аналитик может увидеть все наиболее уязвимые места системы. После этого с учетом выявленных недостатков можно строить модель новой организации бизнес-процессов (модель TO-BE).

IDEF0 модель имеет единственную цель и единственный субъект. Цель модели - получение ответов на определенную совокупность вопросов. Субъект — это сама система. Методология IDEF0 требует, чтобы создаваемая модель системы рассматривалась всегда

с одной и той же позиции, или точки зрения. После определения точки зрения, с которой описывается модель, создается список данных, а потом список функций.

Каждый блок диаграммы IDEF0-модели может быть детализирован на другой диаграмме. Поскольку каждый блок понимается как отдельный, полностью определенный объект, разделение такого объекта на его структурные части (блоки и дуги, составляющие диаграмму) называется декомпозицией. Декомпозиция формирует границы, и каждый блок в IDEF0 рассматривается как формальная граница некоторой части описываемой системы, т.е. блок и касающиеся его дуги определяют точную границу диаграммы, представляющей декомпозицию этого блока. Эта диаграмма, называемая диаграммой- потомком, описывает все, связанное с этим блоком и его дугами, и не описывает ничего вне этой границы.

Декомпозируемый блок называется родительским блоком, а содержащая его диаграмма - родительской диаграммой.

IDEF0 требует, чтобы все внешние дуги (ведущие к краю страницы) диаграммы были согласованы с дугами, образующими границу этой диаграммы, т.е. диаграмма должна быть «состыкована» со своей родительской диаграммой посредством согласования по числу и наименованию дуг.

В методологии принята схема кодирования дуг «ICOM», которая получила название по первым буквам английских эквивалентов слов: вход (Input), управление (Control), выход (Output), механизм (Mechanism). При построении диаграммы следующего уровня дуги, касающиеся декомпозируемого блока, используются в качестве источников и приемников для дуг, которые создаются на новой диаграмме. После завершения диаграммы ее внешние дуги стыкуются с родительской диаграммой для обеспечения согласованности. Стыковка осуществляется посредством присваивания кодов ICOM внешним дугам новой диаграммы. Таким образом, IDEF0-диаграмма составлена из блоков, связанных дугами, которые определяют, как блоки влияют друг на друга. Дуги на диаграммах изображают интерфейсы между функциями системы, а также между системой и ее окружающей средой.

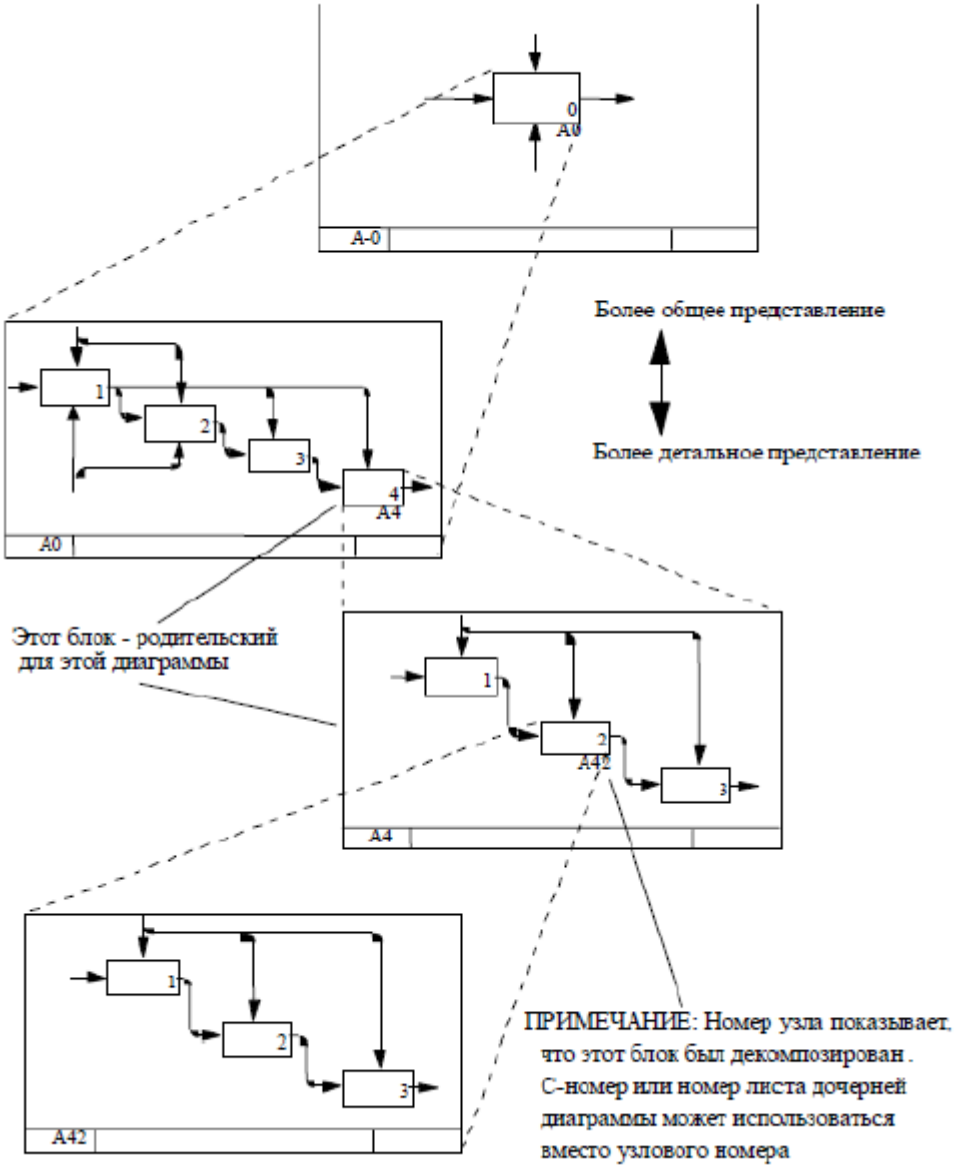


Рис. 4. Декомпозиция диаграмм в IDEF0

Существуют некоторые критерии для определения момента завершения моделирования:

- ✓ блок содержит достаточно деталей;
- ✓ необходимо изменить уровень абстракции, чтобы достичь большей детализации блока;
- ✓ необходимо изменить точку зрения, чтобы детализировать блок;
- ✓ блок очень похож на другой блок той же модели;
- ✓ блок очень похож на блок другой модели;
- ✓ блок представляет тривиальную функцию.

4.5. Стандарт IDEFlx

IDEF1X является методом для разработки реляционных баз данных и использует условный синтаксис, специально разработанный для удобного построения концептуальной схемы. Использование метода IDEF1X наиболее целесообразно для построения логической структуры базы данных после того как все информационные ресурсы исследованы и решение о внедрении реляционной базы данных, как части корпоративной информационной системы, было принято. Средства моделирования IDEF1X специально разработаны для построения реляционных информационных систем, и если существует необходимость проектирования другой системы, например, объектно-ориентированной, то лучше избрать другие методы моделирования.

Существует несколько очевидных причин, по которым IDEF1X не следует применять в случае построения нереляционных систем. Во-первых, IDEF1X требует от проектировщика определить ключевые атрибуты, для того чтобы отличить одну сущность от другой, в то время как объектно-ориентированные системы не требуют задания ключевых ключей, в целях идентифицирования объектов. Во-вторых, в тех случаях, когда более чем один атрибут является однозначно идентифицирующим сущность, проектировщик должен определить один из этих атрибутов первичным ключом, а все остальные вторичными. И, таким образом, построенная проектировщиком IDEF1X-модель и переданная для окончательной реализации программисту является некорректной для применения методов объектно-ориентированной реализации, и предназначена для построения реляционной системы. Подробно данная модель рассматривается в курсах, связанных с базами данных.

4.6. Методология IDEF2. Динамическое моделирование системы

Методология IDEF2 реализует динамическое моделирование системы. В данной методологии модель разбивается на четыре подмодели:

- подмодель возможностей, которая описывает их инициаторов;
- подмодель потока сущностей, которая определяет их трансформацию;

- подмодель распределения ресурсов, необходимых для осуществления переходов между состояниями;
- подмодель системы, которая описывает внешние взаимодействия.

Методология предполагает, что набор подмоделей может быть переведен в динамическую модель.

В связи с весьма серьезными сложностями анализа динамических систем от этого стандарта практически отказались, и его развитие приостановилось на самом начальном этапе. В настоящее время известны алгоритмы и их компьютерные реализации, позволяющие превращать набор статических диаграмм IDEF0 в динамические модели, построенные на базе «раскрашенных сетей Петри» (CPN- Color Petri Nets).

Классические сети Петри ввел Карл Адам Петри в 60-х гг. XX в. С тех пор их использовали для моделирования и анализа самых разных систем с приложениями от протоколов, аппаратных средств и внедренных систем до гибких производственных систем, пользовательского взаимодействия и бизнес-процессов.

4.7. Основные определения сетей Петри

Сети Петри используются для моделирования параллельных процессов: для моделирования компонентов компьютера, параллельных вычислений, в робототехнике и даже для описания музыкальных структур. Вообще, сети

Петри используют для нахождения дефектов в проекте системы, хотя имеют и многие другие применения. Они обладают многими свойствами блок-схем и конечных автоматов.

Сети Петри были разработаны и используются для моделирования параллельных и асинхронных систем. При моделировании в сетях Петри позиции символизируют какое-либо состояние системы, а переход символизируют какие-то действия, происходящие в системе. Система, находясь в каком- то состоянии, может порождать определенные действия, и наоборот, выполнение какого-то действия переводит систему из одного состояния в другое.

Сетью Петри называется совокупность множеств $C = \{P,T,I,O\}$, где:

P - конечное множество, элементы которого называются позициями;

T - конечное множество, элементы которого называются переходами, $P \cap T = \varnothing$;

I - множество входных функций, $I: T \rightarrow P$;

O - множество выходных функций, $O: T \rightarrow P$.

Сеть Петри представляет собой ориентированный граф с вершинами двух типов (позициями и переходами), в котором дугами могут соединяться только вершины различных типов. В позиции сети помещаются специальные маркеры («фишки»), перемещение которых и отображает динамику моделируемой системы. Изменение маркировки (движение маркеров) происходит в результате выполнения (срабатывания) перехода на основе соответствующего внешнего события. Точнее, переход срабатывает, если во всех его входных позициях имеются маркеры и происходит соответствующее переходу событие. При этом из каждой входной позиции срабатываемого перехода маркер удаляется, а в каждую выходную позицию — заносится.

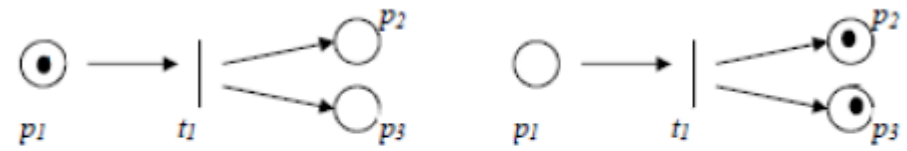
Сеть Петри называется маркированной, если существует функция p, называемая маркировкой (разметкой) сети, которая ставит в соответствие неотрицательное целое число каждому элементу множества P. Если p - позиция, то μ(p) называется разметкой позиции p. Таким образом, маркированная сеть Петри задается пятеркой $C = \{P,T,I,O,\mu\}$, где μ - целочисленный вектор $\mu = (\mu_1, \mu_2, \ldots \mu_n)$, $n = |P|$, $\mu_i = \mu(p_i)$, $i=1...n$.

Разметка множества на графе указывается с помощью черных точек, называемых метками (фишками), помещенных в кружки, которые обозначают позиции. Количество меток можно также указывать числом, записанном в кружке. Если кружок позиции p пуст, это означает, что в p меток нет. Маркировка сети Петри аналогична состоянию конечного автомата.

При моделировании гибких производственных систем позиции отражают отдельные операции производственного процесса (например: транспортировка заготовки к конвейеру, передвижение заготовки к станку конвейером,

обработку детали) или состояния компонентов гибкой производственной системы (например: робота, конвейера, станка). Наличие метки в одной из позиций соответствует состоянию выполнения некоторой из технологических операций либо состояние, в котором пребывают некоторые из компонентов гибкой производственной системы.

Переходы соответствуют событиям, отображающим начало или завершение моделируемых операций. Например, переход интерпретируется как событие, связанное с завершением операции транспортирования заготовки роботом и ее установки на конвейере, а также с началом операции перемещения заготовки конвейером к станку.



$C\mu = \{P,T,I,O,\mu\}$,
$P = \{p_1; p_2; p_3\}$; $T = \{t_1\}$; $I(t_1) = \{p_1\}$; $O(t_1) = \{p_2; p_3\}$; $\mu = (1;0;0)$; $\mu_1 = (0;1;1)$

Рис. 5. Пример сети Петри C_μ

4.8. Пример построения сетей Петри

Построить сеть Петри, моделирующую работу рабочей станции, обслуживающей группу пользователей. Пользователь присылает заявку на обработку задания. Если станция свободна, она начинает обработку задания. После выполнения задания станция передает обслуженную заявку, освобождается и либо начинает обрабатывать новую заявку (если заявка поступила), либо ждет поступления новой заявки.

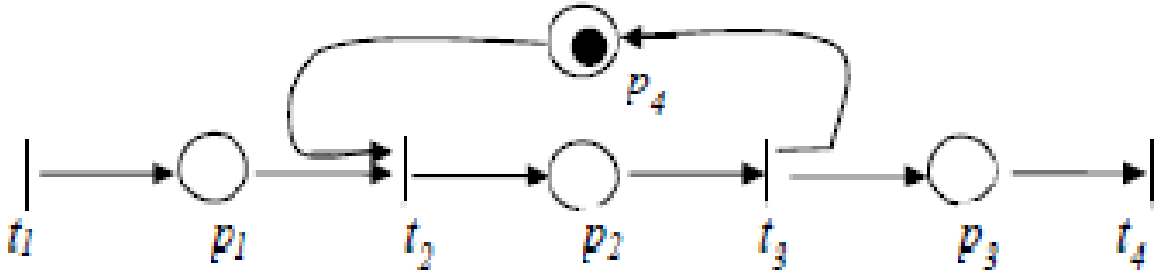


Рис. 6. Пример сети Петри

t_1 - поступила заявка на обработку;

t_2 - задание начинает обрабатываться;

t_3 - конец обработки задания;

t_4 - передача выполненной заявки;

p_1 - задание ждет освобождения станции;

p_2 - задание обрабатывается;

p_3 - задание ожидает очереди на выход;

p_4 - рабочая станция свободна.

Позиция p4 показывает, свободна ли рабочая станция. Наличие метки в позиции указывает на то, что станция свободна. Как только задание начинает обрабатываться, срабатывает переход t2, и маркировка позиции обнуляется. После окончания обработки запускается переход t3 и позиция p4 вновь получает метку. Таким образом, пока не сработает переход t3, новая заявка не может быть обработана.

4.9. Методология документирования процессов IDEF3

Стандарт IDEF3 - это методология описания процессов, рассматривающая последовательность выполнения и причинно-следственные связи между ситуациями и событиями для структурного представления знаний о системе. При помощи IDEF3 описывают логику выполнения работ, очередность их запуска и завершения, т.е. IDEF3 предоставляет инструмент моделирования сценариев действий сотрудников организации, отделов, цехов и т.п., например, порядок обработки заказа или события, на которые необходимо реагировать за конечное время, выполнение действий по производству товара и т.д.

IDEF3 как инструмент моделирования фиксирует следующую информацию о процессе:

- объекты, которые участвуют при выполнении сценария;
- роли, которые выполняют эти объекты, например, агент, транспорт;
- отношения между работами в ходе выполнения сценария процесса;

- состояния и изменения, которым подвергаются объекты;
- время выполнения и контрольные точки синхронизации работ;
- ресурсы, которые необходимы для выполнения работ.

Средства документирования и моделирования IDEF3 позволяют выполнять следующие задачи:

- документировать имеющиеся данные о технологии выполнения процесса, выявленные, например, во время опроса специалистов предметной области, ответственных за организацию рассматриваемого процесса или участвующих в нем;
- анализировать существующие процессы и разрабатывать новые;
- определять и анализировать точки влияния потоков сопутствующего документооборота на сценарий технологических процессов;
- определять ситуации, в которых требуется принятие решения, влияющего на жизненный цикл процесса, например, изменение конструктивных, технологических или эксплуатационных свойств конечного продукта;
- содействовать принятию оптимальных решений при реорганизации процессов;
- разрабатывать имитационные модели технологических процессов по принципу «как будет, если...».

Таким образом, IDEF3 - это методология, способная фиксировать и структурировать описание работы системы. При этом сбор сведений может производиться из многих источников, что позволяет зафиксировать информацию от экспертов о поведении системы.

4.10. Основные элементы IDEF3-диаграмм

Основные элементы IDEF3-диаграмм следующие:

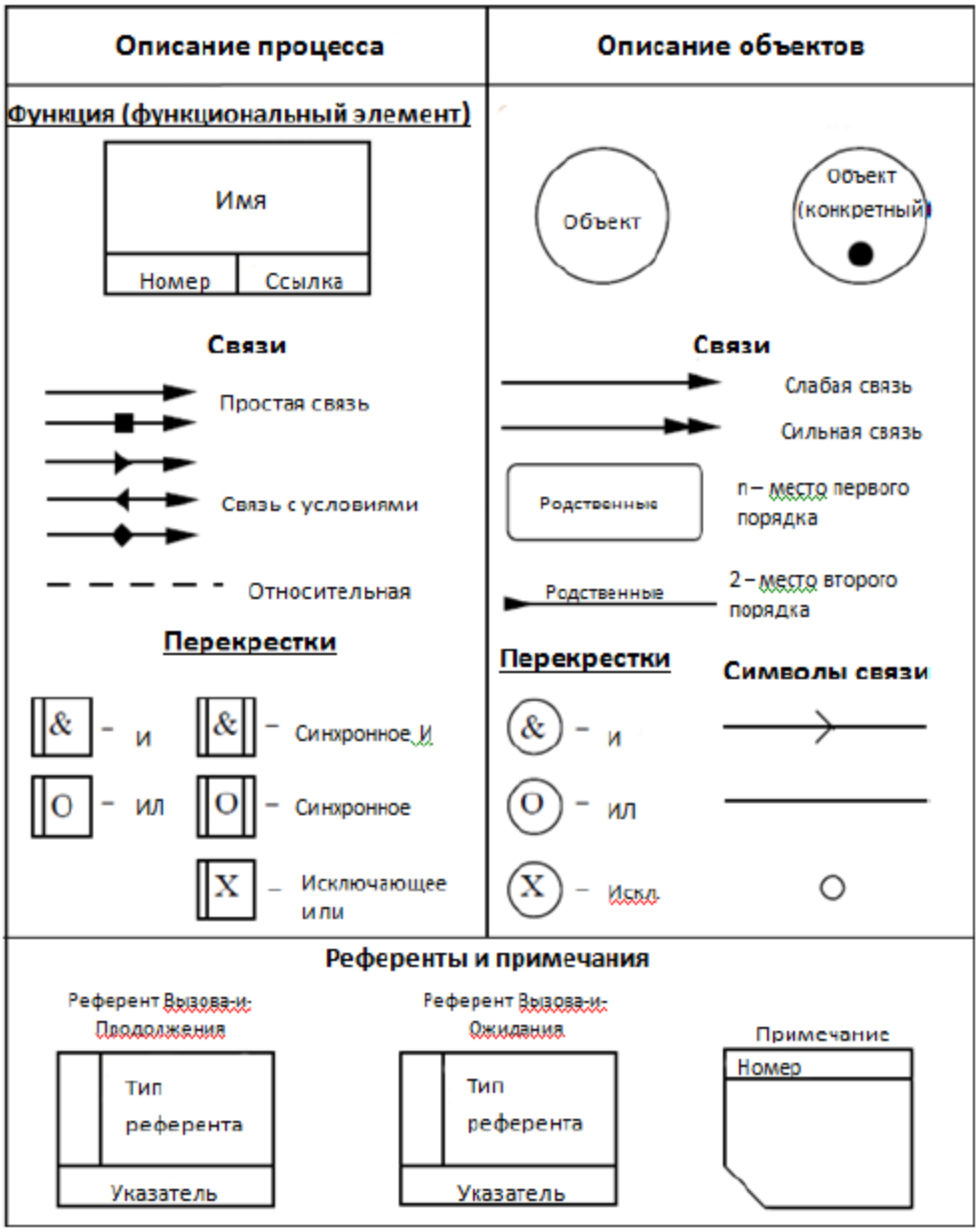


Рис. 6. Элементы IDEF3-диаграмм.

Разветвление или объединение связей может быть только через перекресски. Перекрестки используются для отображения логики отношений между множеством событий и временной синхронизации активизации элементов IDEF3-диаграмм. Различают перекрестки для слияния (Fan-in Junction) и разветвления (Fan-out Junction) стрелок. Перекресток не может использоваться одновременно для слияния и для разветвления. При внесении перекрестка в диаграмму необходимо указать его тип. Тип перекрестка определяет логику и временные параметры отношений между элементами диаграммы. Все перекрестки на диаграмме нумеруются, каждый номер имеет

префикс «J». Тип перекрестка обозначается внутри элемента: & - логический И; ∪ - логический ИЛИ; X - логический перекресток неэквивалентности.

Стандарт IDEF3 предусматривает разделение перекрестков типа & и ∪ на синхронные и асинхронные. Это разделение позволяет учитывать в диаграммах описания процессов синхронизацию времени активизации.

Методология IDEF3 использует пять логических типов для моделирования возможных последствий действий в сценарии.

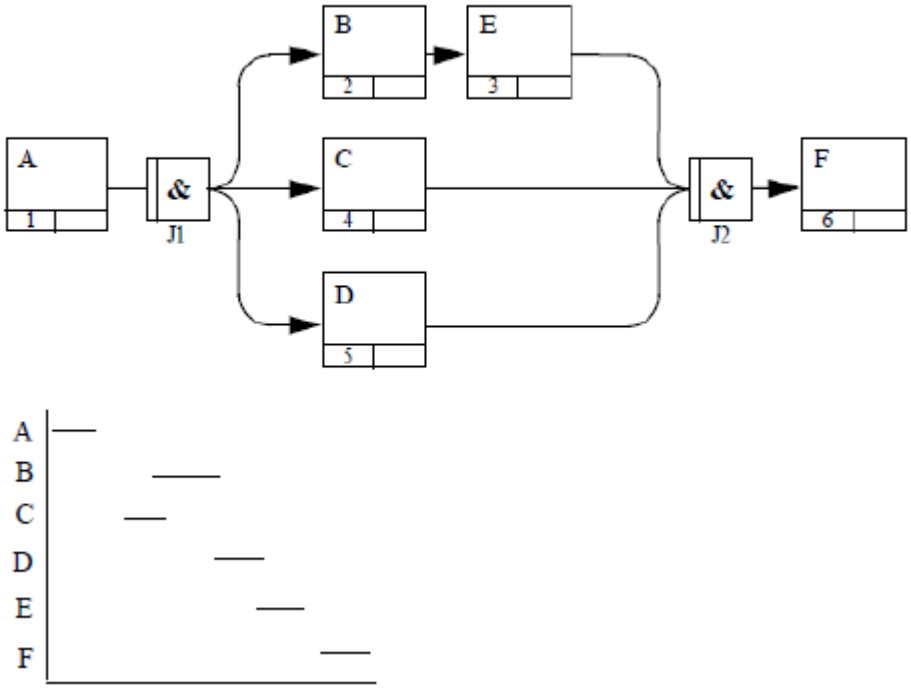


Рис. 7. Пример использования перекрестков.

4.11. Декомпозиция описания процесса

Методология IDEF3 дает возможность представлять процесс в виде иерархически организованной совокупности диаграмм. Диаграммы состоят из нескольких элементов описания процесса IDEF3, причем каждый функциональный элемент UOB потенциально может быть детализирован на другой диаграмме. Такое разделение сложных комплексных процессов на их структурные части называется декомпозицией.

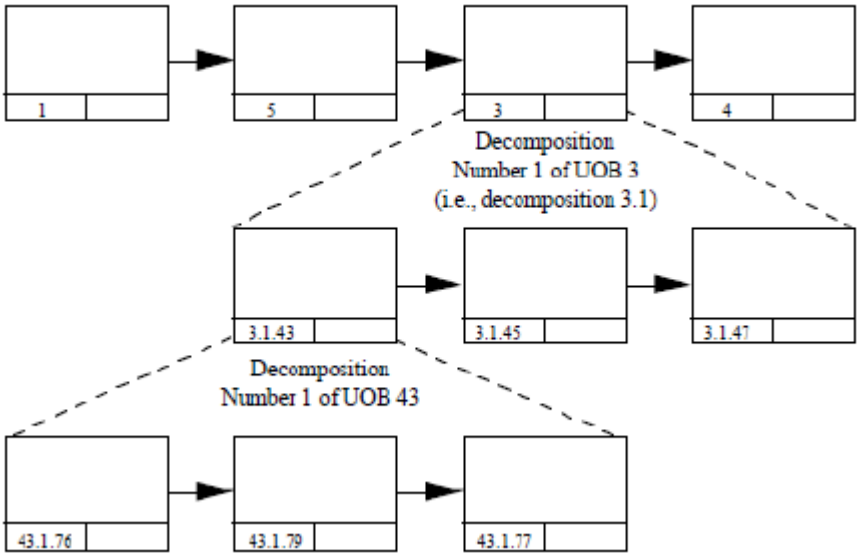


Рис. 8. Пример декомпозиции диаграммы IDEF3

4.12. Диаграммы потоков данных (DFD)

Целью методики является построение модели рассматриваемой системы в виде диаграммы потоков данных (Data Flow Diagram - DFD), обеспечивающей правильное описание выходов (отклика системы в вида данных) при заданном воздействии на вход системы (подаче сигналов через внешние интерфейсы). Диаграммы потоков данных являются основным средством моделирования функциональных требований к проектируемой системе.

Для изображения DFD традиционно используются две различные нотации: Йодана (Yourdon) и Гейна-Сарсона (Gane-Sarson), представленные на рисунке (рис. 8.).

Потоки данных являются механизмами, использующимися для моделирования передачи информации (или физических компонент) из одной части системы в другую. Потоки на диаграммах обычно изображаются именованными стрелками, ориентация которых указывает направление движения информации.

Назначение процесса (работы) состоит в продуцировании выходных потоков из входных в соответствии с действием, задаваемым именем процесса. Имя процесса должно содержать глагол в неопределенной форме с последующим дополнением (например, «вычислить максимальную высоту»). Кроме того, каждый процесс должен иметь уникальный номер для ссылок на

него внутри диаграммы. Этот номер может использоваться совместно с номером диаграммы для получения уникального индекса процесса во всей модели.

Хранилище (накопитель) данных позволяет на определенных участках определять данные, которые будут сохраняться в памяти между процессами. Фактически хранилище представляет «срезы» потоков данных во времени. Информация, которую оно содержит, может использоваться в любое время после ее определения, при этом данные могут выбираться в любом порядке. Имя хранилища должно идентифицировать его содержимое и быть существительным.

Внешняя сущность представляет собой материальный объект вне контекста системы, являющейся источником или приемником системных данных. Ее имя должно содержать существительное, например, «склад товаров». Предполагается, что объекты, представленные такими узлами, не должны участвовать ни в какой обработке.

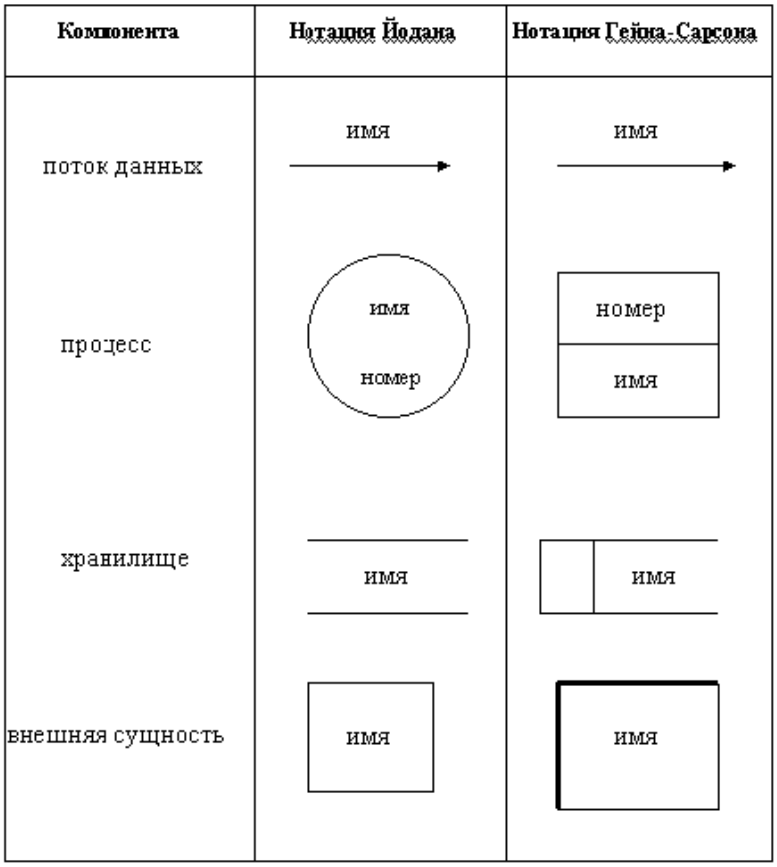


Рис. 8. Основные символы диаграммы потоков данных DFD.

Декомпозиция DFD-диаграммы осуществляется на основе процессов: каждый процесс может раскрываться с помощью DFD нижнего уровня. Важную специфическую роль в модели играет специальный вид DFD - контекстная диаграмма, моделирующая систему наиболее общим образом. Контекстная диаграмма отражает интерфейс системы с внешним миром, а именно, информационные потоки между системой и внешними сущностями, с которыми она должна быть связана. Она идентифицирует эти внешние сущности, а также, как правило, единственный процесс, отражающий главную цель или природу системы. И хотя контекстная диаграмма выглядит тривиальной, несомненная ее полезность заключается в том, что она устанавливает границы анализируемой системы. Каждый проект должен иметь ровно одну контекстную диаграмму, при этом нет необходимости в нумерации единственного ее процесса.

DFD первого уровня строится как декомпозиция процесса, который присутствует на контекстной диаграмме. Построенная диаграмма первого уровня также имеет множество процессов, которые в свою очередь могут быть декомпозированы в DFD нижнего уровня. Таким образом, строится иерархия DFD с контекстной диаграммой в корне дерева. Этот процесс декомпозиции продолжается до тех пор, пока процессы могут быть эффективно описаны с помощью коротких (до одной страницы) миниспецификаций обработки (спецификаций процессов).

При таком построении иерархии DFD каждый процесс более низкого уровня необходимо соотнести с процессом верхнего уровня. Обычно для этой цели используются структурированные номера процессов.

Кроме основных элементов в состав DFD входят словари данных и миниспецификации.

Словари данных являются каталогами всех элементов данных, присутствующих в DFD, включая групповые и индивидуальные потоки данных, хранилищи и процессы, а также все их атрибуты.

Миниспецификации обработки описывают DFD-процессы нижнего уровня. Фактически миниспецификации представляют собой алгоритмы описания задач, выполняемых процессами: множество всех миниспецификаций является полной спецификацией системы.

Для обеспечения декомпозиции данных и некоторых других сервисных возможностей к DFD добавляются следующие типы объектов:

- групповой узел предназначен для расщепления и объединения потоков. В некоторых случаях может отсутствовать, то есть фактически вырождаться в точку слияния/расщепления потоков на диаграмме;
- узел-предок позволяет увязывать входящие и выходящие потоки между детализируемым процессом и детализирующей DFD;
- неиспользуемый узел применяется в ситуации, когда декомпозиция данных производится в групповом узле, при этом требуются не все элементы входящего в узел потока;
- узел изменения имени позволяет неоднозначно именовать потоки, при этом их содержимое эквивалентно. Например, если при проектировании разных частей системы один и тот же фрагмент данных получил различные имена, то эквивалентность соответствующих потоков данных обеспечивается узлом изменения имени. При этом один из потоков данных является входным для данного узла, а другой - выходным;
- текст в свободном формате в любом месте диаграммы.

Возможный способ изображения этих узлов приведен на рисунке.

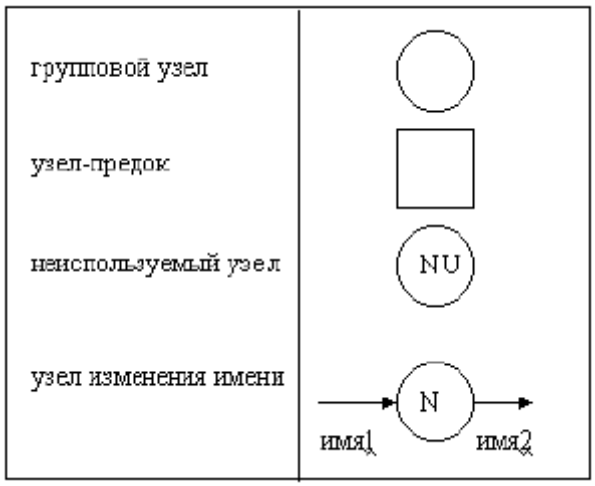


Рис. 9. Расширения диаграммы потоков данных.

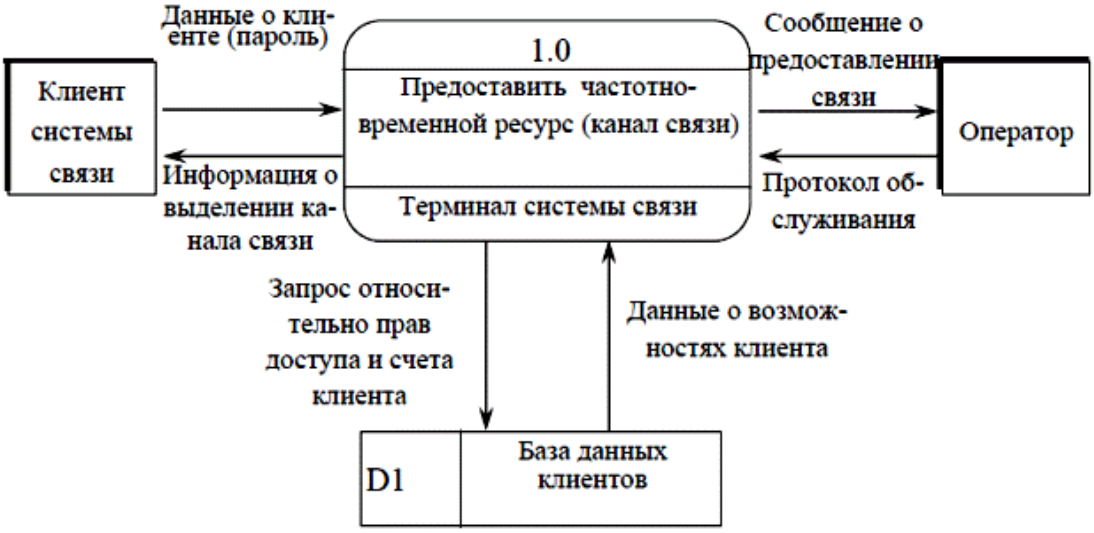


Рис. 10. Пример диаграммы DFD

5. ПАТТЕРНЫ В АРХИТЕКТУРЕ ИНФОРМАЦИОННЫХ СИСТЕМ

Шаблон проектирования или паттерн (англ. design pattern) в разработке программного обеспечения — повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Обычно шаблон не является законченным образцом, который может быть прямо преобразован в код; это лишь пример решения задачи, который можно использовать в различных ситуациях. Объектно-ориентированные шаблоны

показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться.

«Низкоуровневые» шаблоны, учитывающие специфику конкретного языка программирования, называются идиомами. Это хорошие решения проектирования, характерные для конкретного языка или программной платформы, и потому не универсальные.

На наивысшем уровне существуют архитектурные шаблоны, они охватывают собой архитектуру всей программной системы.

Алгоритмы по своей сути также являются шаблонами, но не проектирования, а вычисления, так как решают вычислительные задачи.

Можно сказать, что паттерны в программировании применяются с 1987 г. В сравнении с полностью самостоятельным проектированием, шаблоны обладают рядом преимуществ. Основная польза от использования шаблонов состоит в снижении сложности разработки за счёт готовых абстракций для решения целого класса проблем. Шаблон даёт решению своё имя, что облегчает коммуникацию между разработчиками, позволяя ссылаться на известные шаблоны. Таким образом, за счёт шаблонов производится унификация деталей решений: модулей, элементов проекта, — снижается количество ошибок. Применение шаблонов концептуально сродни использованию готовых библиотек кода. Правильно сформулированный шаблон проектирования позволяет, отыскав удачное решение, пользоваться им снова и снова. Набор шаблонов помогает разработчику выбрать возможный, наиболее подходящий вариант проектирования. Вместе с тем, хотя легкое изменение кода под известный шаблон может упростить понимание кода, по мнению Стива Макконнелла, с применением шаблонов могут быть связаны две сложности. Во-первых, слепое следование некоторому выбранному шаблону может привести к усложнению программы. Во-вторых, у разработчика может возникнуть желание попробовать некоторый шаблон в деле без особых оснований.

Основное отличие паттернов от компонентов состоит в том, что компонент является модулем, который после настройки можно включить в состав системы, т.е. его можно рассматривать как готовый к употреблению строительный блок, а паттерн — это только заготовка, которую еще надо обработать, т. е. добавить код, определяющий функциональность.

Известны различные подходы к классификации паттернов. Так, паттерны предлагается классифицировать с точки зрения уровня абстракции на концептуальные, проектирования и программные.

Концептуальные паттерны — это паттерны, функционирование которых описывается в терминах предметной области. Такие паттерны относятся к приложению в целом или крупным подсистемам ИС.

Паттерны проектирования — это паттерны, для описания которых используются термины, относящиеся к разработке программных систем, такие как объект, класс, модуль. Паттерны проектирования описывают решение общих проблем в конкретном контексте.

Программные паттерны — это паттерны, для описания которых используются такие относительно низкоуровневые понятия как деревья, списки и т. п.

Используется также классификация, в соответствии с которой выделяются следующие типы паттернов: архитектурные, системные, структурные, поведенческие, производящие, параллельные программирования [15]. Кроме того, паттерны можно разделить на паттерны общего назначения и доменно-ориентированные паттерны. Паттерны общего назначения не привязаны явным образом к той или иной предметной области (домену), а доменно-ориентированные паттерны имеют такую привязку.

Архитектурный паттерн (architectural patterns) описывает структуру программной системы и определяет состав подсистем, их основные функции и допустимые способы компоновки подсистем. Архитектурные паттерны называют также архитектурными стилями. Архитектурные паттерны являются описанием, которое не зависит ни от платформы, ни от языка

программирования. Архитектурные паттерны можно рассматривать как паттерны высокого уровня. Архитектурные стили были рассмотрены ранее.

Системные паттерны (system patterns) представляют собой приложение на верхнем (системном) уровне. Системные паттерны могут применяться в приложении для реализации типовых процессов и даже для поддержки взаимодействия разных приложений. Системные паттерны можно рассматривать как паттерны, использование которых позволяет получить улучшенные архитектурные решения.

Структурные паттерны с одинаковой эффективностью применяются как для разделения, так и для объединения элементов приложения.

Структурные шаблоны могут быть использованы для решения различных задач. Например, шаблон Адаптер может обеспечить возможность двум несовместимым системам обмениваться информацией, тогда как шаблон Фасад позволяет отобразить упрощенный пользовательский интерфейс, не удаляя ненужных конкретному пользователю элементов управления.

Поведенческие паттерны (behavioral patterns) применяются для передачи управления в системе. Существуют методы организации управления, применение которых позволяет добиться значительного повышения как эффективности системы, так и удобства ее эксплуатации. Поведенческие шаблоны представляют собой набор проверенных на практике методов и обеспечивают понятные и простые в применении эвристические способы организации управления.

Производящие паттерны (creational patterns) предназначены для создания объектов в системе.

В ходе работы большинства объектно-ориентированных систем, независимо от уровня их сложности, создается множество экземпля ров объектов. Производящие паттерны облегчают процесс создания объектов, предоставляя разработчику следующие возможности:

- ✓ единый способ получения экземпляров объектов: в системе обеспечивается механизм создания объектов без необходимости идентификации определенных типов классов в программном коде;
- ✓ простота создания объектов: разработчик полностью избавлен от необходимости написания большого и сложного программного Кода для получения экземпляра объекта;
- ✓ учет ограничений при создании объектов.

Паттерны параллельного программирования ориентированы на обеспечение корректного взаимодействия асинхронно протекающих процессов и ориентированы на решение двух основных задач (совместное использование ресурсов и управление доступом к ресурсам):

- ✓ совместное использование ресурсов. Если конкурирующие операции обращаются к одним и тем же данным или к общему ресурсу, то они могут конфликтовать друг с другом, если эти операции осуществляют доступ к ресурсу в один и тот же момент. Чтобы обеспе чить корректное выполнение таких операций, их нужно ограничить таким образом, чтобы доступ к общему ресурсу в конкретный момент времени получала только одна операция, однако чрезмерное ограничение операций может привести к их взаимной блокировке;
- ✓ управление доступом к ресурсам. Если операции получают доступ к общему ресурсу одновременно, возникает необходимость в том, чтобы они обращались к общему ресурсу в определенном порядке, например, объект не может быть удален из структуры данных до тех пор, пока данный объект не будет добавлен в некоторую другую структуру данных.

Для разных типов паттернов могут использоваться разные способы описания. Чаще всего используется описание, предложенное в [16], в соответствии с которым полное описание паттерна выглядит следующим образом:

- название и тип;
- назначение;
- другие названия (если имеются);
- мотивация — какие проблемы можно решить с помощью данного паттерна;
- условия, при которых целесообразно применять данный паттерн;
- структура паттерна (в объектно-ориентированной нотации);
- объекты и паттерны, используемые в данном паттерне;
- результаты работы паттерна;
- рекомендации по применению;
- пример кода;
- примеры использования;
- родственные паттерны.

Как указывалось выше, паттерны — это классы проверенных практикой проектных решений, использование которых приводит к положительным результатам.

Вместе с паттернами существуют так называемые антипаттерны. Антипаттерны (antipatterns), также известные как ловушки (pitfalls) — это классы наиболее часто внедряемых плохих решений проблем. Они изучаются как категория, в случае, когда их хотят избежать в будущем, и некоторые отдельные случаи их могут быть распознаны при изучении неработающих систем [1].

Частью хорошей практики программирования является избегание антипаттернов.

Данная концепция также прекрасно подходит к машиностроению, строительству и другим отраслям. Несмотря на то что термин нечасто используется вне программной инженерии, концепция является универсальной.

Следует отметить, что общепринятой классификации антипаттернов не существует. Применительно к ИС можно выделить следующие типовые группы антипаттернов: *в управлении разработкой ПО, антипаттерны в разработке*

ПО, в объектно-ориентированном проектировании, в области программирования, методологические и организационные антипаттерны.

6. ФРЕЙМВОРКИ

Фре́ймворк, иногда фреймво́рк (англицизм, неологизм от framework «остов, каркас, структура») — заготовки, шаблоны для программной платформы, определяющие структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных модулей программного проекта.

Уместно использование термина «каркас». Некоторые авторы используют его в качестве основного, не опираясь на англоязычный аналог. Можно также говорить о каркасном подходе, как о подходе к построению программ, где любая конфигурация программы строится из двух частей:

1. Постоянная часть — каркас, не меняющийся от конфигурации к конфигурации и несущий в себе гнёзда, в которых размещается вторая, переменная часть;
2. Сменные модули (или точки расширения).

«Фреймворк» отличается от понятия библиотеки тем, что библиотека может быть использована в программном продукте просто как набор подпрограмм близкой функциональности, не влияя на архитектуру программного продукта и не накладывая на неё никаких ограничений. «Фреймворк» же диктует правила построения архитектуры приложения, задавая на начальном этапе разработки поведение по умолчанию — «каркас», который нужно будет расширять и изменять согласно указанным требованиям. Пример программного фреймворка — C.M.F. (Content Management Framework), а пример библиотеки — модуль электронной почты.

Также, в отличие от библиотеки, которая объединяет в себе набор близкой функциональности, «фреймворк» может содержать большое число разных по тематике библиотек.

Другим ключевым отличием «фреймворка» от библиотеки может быть инверсия управления: пользовательский код вызывает функции библиотеки (или классы) и получает управление после вызова. Во «фреймворке» пользовательский код может реализовывать конкретное поведение, встраиваемое в более общий — «абстрактный» код фреймворка. При этом «фреймворк» вызывает функции (классы) пользовательского кода.

Фреймворк программной системы - это каркас программной системы (или подсистемы). Может включать: вспомогательные программы, библиотеки кода, язык сценариев и другое ПО, облегчающее разработку и объединение разных компонентов большого программного проекта. Обычно объединение происходит за счёт использования единого API.

В качестве примеров можно указать веб-каркасы, такие как Rocket на Rust, Zend Framework и Symfony на PHP, Django, написанный на Python.

Одно из главных преимуществ при использовании «каркасных» приложений — «стандартность» структуры приложения. «Каркасы» стали популярны с появлением графических интерфейсов пользователя, которые имели тенденцию к реализации стандартной структуры для приложений. С их использованием стало гораздо проще создавать средства для автоматического создания графических интерфейсов, так как структура внутренней реализации кода приложения стала известна заранее. Для обеспечения каркаса обычно используются техники объектно-ориентированного программирования (например, части приложения могут наследоваться от базовых классов фреймворка).

Одним из первых коммерческих фреймворков приложения был MacApp, написанный Apple для Macintosh. Первоначально созданный с помощью расширенной (объектно-ориентированной) версии языка Object Pascal, впоследствии он был переписан на C++. Другие популярные каркасы для Macintosh включали:

Metrowerks PowerPlant и MacZoop (все основаны на Carbon);

WebObjects от NeXT.

В различной степени фреймворки приложения представляют собой Cocoa для Mac OS X, а также свободные фреймворки, существующие как часть проектов Mozilla, OpenOffice.org, GNOME и KDE.

Реализация фреймворка. «Фреймворк» определяется как множество конкретных и абстрактных классов, а также определений способов их взаимоотношения. Конкретные классы обычно реализуют взаимные отношения между классами. Абстрактные классы представляют собой точки расширения, в которых каркасы могут быть использованы или адаптированы.

Точка расширения — это та «часть» фреймворка, для которой не приведена реализация. Соответственно, каркас концептуальной модели состоит из концептуальных классов, а каркас программной системы — из классов языка программирования общего назначения.

Процесс создания фреймворка заключается в выборе подмножества задач проблем и их реализаций. В ходе реализаций общие средства решения задач заключаются в конкретных классах, а изменяемые средства — выносятся в точки расширения.

Microsoft создала похожий продукт для Windows, который называется Microsoft Foundation Classes (MFC). На данный момент основным продуктом Microsoft для разработки ПО предлагается .NET Framework.

Кроссплатформенными каркасами приложений (для операционных систем Linux, Macintosh и Windows) являются, например, widget toolkit, wxWidgets, Qt, MyCoRe[de] или FOX toolkit.

Фреймворки и паттерны имеют много общего и, в первую очередь, — это подходы к повторному использованию кода. Различие между паттернами и фреймворками состоит в следующем. Фреймворк можно рассматривать как реализацию системы паттернов проектирования (поведенческих паттернов). В то время как фреймворк — это исполняемая программа, а паттерн — это знание и опыт как решать конкретную задачу.

Еще одно отличие фреймворка от паттерна состоит в том, что фреймворк представляет собой скелетное решение достаточно крупной задачи и обычно

включает в себя большое количество как паттернов, так и компонентов. Кроме того, фреймворки могут использовать подклассы и другие механизмы.

Фреймворки можно классифицировать по месту применения в ИТ-системе, способу использования и масштабу применения [9]. По месту применения фреймворки можно разделить на три типа: инфраструктурные фреймворки (System Infrastructure Frameworks); фреймворки уровня промежуточного ПО (Middleware Frameworks); фреймворки, ориентированные на приложения, относящиеся к конкретному предметному домену (Enterprise Application Frameworks), архитектурные фреймворки. Использование **инфраструктурных фреймворков** упрощает разработку инфрастуктурных элементов, таких как, например операционные системы. Обычно такие фреймворки используются внутри организации и не поступают в продажу.

Фреймворки уровня промежуточного ПО используются для интеграции распределенных приложений и компонентов.

Фреймворки, ориентированные на приложения, используются, в первую очередь, для поддержания процесса разработки систем, ориентированных на конечного пользователя и принадлежащих некоторому конкретному предметному домену.

Международный стандарт ISO/IEC 42010 определяет **архитектурный фреймворк** как «совокупность соглашений, принципов и практик, используемых для описаний архитектур и принятых применительно к некоторому предметному домену и (или) в сообществе специалистов (заинтересованных лиц)»[21].

Типовой архитектурный фреймворк включает в себя следующие основные элементы: типовые для домена заинтересованные лица; типовые для домена проблемы; архитектурные точки зрения; правила интеграции различных точек зрения.

Можно выделить два типовых подхода к использованию фреймворков:

- по принципу белого ящика;
- по принципу черного ящика.

Кроме того, возможно использование гибридных подходов, которые называют серым ящиком.

Фреймворки, используемые по принципу белого ящика, называют также архитектурными фреймворками (architecture-driving framework). Эти фреймворки применяют наследование и динамическое связывание для формирования скелета приложения. Фреймворк, работающий по принципу белого ящика, определяется через интерфейсы объектов, которые разработчик может добавлять в систему. Основным недостатком данного подхода является то, что для того чтобы работать с фреймворками, необходимо иметь подробную информацию о классах, которые будут расширяться.

Фреймворки, используемые по принципу черного ящика, называют также фреймворками, управляемыми данными. При использовании фреймворков данного типа в качестве основных механизмов формирования приложения выступают композиция компонентов и параметризация. При этом требуемая функциональность достигается посредством добавление в фреймворк дополнительных компонентов. В общем случае работать с фреймворками, использующими принцип черного ящика, проще, чем с фреймворками, реализующими принцип белого ящика, но их разработка является более сложной.

Большинство реальных фреймворков используют комбинацию рассмотренных выше подходов и их называют **фреймворками, используемыми по принципу серого ящика (grey-box).**

По масштабу применения фреймворки можно разделить на три группы: фреймворки уровня приложений, фреймворки уровня домена (организации), вспомогательные фреймворки.

Фреймворки уровня приложения (application frameworks) обеспечивают полный набор функций, которые реализуются типовыми приложениями. Обычно сюда входят GUI, базы данных, документация. Примером таких фреймворков могут быть MFC (Microsoft Foundation Classes),

которые служат для создания приложений, ориентированных на работу в среде MS Windows.

Фреймворки уровня домена (Domain frameworks) используются для создания приложений, относящихся к определенному предметному домену.

В качестве домена может выступать как информационная система, включающая несколько взаимодействующих между собой приложений, например, системы сбора и обработки телеметрической информации, поступающих от сложной технической системы, так и целой организации, в качестве которой могут выступать, например, промышленные корпорации, органы государственной власти, правительственные ведомства и т. п.

В последнем случае речь идет о фреймворках уровня организации (enterprise). Термин «организация» понимается в самом широком смысле и включает коммерческие и некоммерческие организации, целые корпорации и их подразделения, различного рода ассоциации типа совместных предприятий и т.д. Следует особо отметить, что термин «организация» включает в себя такие элементы, как людей, собственно бизнес, информацию, технологии, а не только информационную систему.

Фреймворки уровня домена можно классифицировать по следующим признакам:

- ✓ назначению;
- ✓ принципам построения;
- ✓ гибкости при использовании;
- ✓ условиям распространения.

Возможны различные концептуальные подходы к построению фреймворков, в частности, ключевым элементом фреймворка может быть онтология. В основу фреймворка может быть положен тот или иной процесс разработки или ориентация на данные (данноцентри- ческий подход), фреймворк может быть ориентирован на эффективную поддержку сетевых взаимодействий.

Вспомогательные фреймворки (Support frameworks) ориентированы на решение частных задач, таких как управление памятью или файловой системой.

Фреймворки могут распространяться на коммерческой основе либо быть бесплатными для некоммерческого использования. Кроме того, фреймворк может быть ориентирован на использование внутри организации.

С точки зрения возможности реконфигурирования и возможности настройки на конкретное применение фреймворки можно разделить на «жесткие» и «гибкие». **Жесткие фреймворки** не предусматривают возможности настройки, а гибкие — разрешают настраивать фреймворк для решения конкретной задачи. Жесткие фреймворки могут требовать использовать конкретный инструментарий и конкретную методологию проектированию.

7. СЕРВИСНО-ОРИЕНТИРОВАННЫЕ АРХИТЕКТУРЫ (СОА) И WEB-СЕРВИСЫ ИНФОРМАЦИОННЫХ СИСТЕМ

Сервисно-ориентированные архитектуры. Сервисно-ориентированная архитектура СОА (service-oriented architecture, SOA) — это подход к созданию ИС, основанный на использовании сервисов или служб (service). Сервис и службу можно рассматривать как синонимы. СОА — это, в первую очередь, интеграционная архитектура, использование которой позволяет обеспечить гибкую интеграцию ИС. При использовании СОА приложения взаимодействуют, вызывая сервисы, входящие в состав других приложений. Сервисы объединяются в более крупные последовательности, реализуя бизнес-процессы, которые могут быть доступны как сервисы.

СОА можно рассматривать так же как подход к построению слабосвязанных (loosely coupled) систем, реализующих механизмы асинхронного взаимодействия. К слабосвязанным системам обычно относятся такие системы, как электронная почта и системы очередей сообщений.

Сервисы можно рассматривать как строительные блоки, которые могут использоваться для построения как сервисов более высокого уровня, так и

законченных распределенных ИТ-систем. Сервисы могут вызываться независимо внешними или внутренними потребителями для выполнения элементарных функций либо могут объединяться в цепочки для формирования более сложных функций и для быстрого создания новых функций.

При использовании СОА организации могут создавать гибкие КИС, которые позволяют оперативно реализовывать быстро изменяющиеся бизнес-процессы и многократно использовать одни и те же компоненты в рамках одной ИТ-системы, в рамках семейств продуктов и в независимых ИТ-системах.

Сервисом можно назвать любую дискретную функцию, которая может быть предложена внешнему потребителю. В качестве сервиса может выступать как отдельная бизнес-функция, так и набор функций, которые образуют бизнес-процесс.

СОА не принуждает пользователя использовать конкретный протокол для доступа к сервису. Идея заключается в том, что сервис не определяется используемым протоколом, напротив, описание сервиса составляется независимым от протокола способом, позволяющим использовать для доступа к сервису разные протоколы. В идеале служба определяется только один раз при помощи интерфейса службы и должна иметь несколько реализаций для работы с разными протоколами доступа. Такой подход позволяет максимально расширить возможности повторного использования сервиса [2, 3, 4].

Web-сервисы. В самом общем виде понятие Web-сервиса можно определить как сервис (услугу), которая предоставляется через WWW с использованием языка XML и протокола HTTP.

Практически все ведущие ИТ-компании положительно относятся к использованию Web-сервисов, поэтому Web-сервисы можно использовать в качестве механизма интеграции приложений, реализованных на любых платформах. Существует много разных определений понятия Web-сервиса.

В качестве «официального» определения можно рассматривать определение, которое дается консорциумом W3C:

«Web-сервис представляет собой приложение, которое идентифицируется строкой URI. Интерфейсы и привязки данного приложения описываются и обнаруживаются с использованием XML-средств. Приложения взаимодействуют посредством обмена сообщениями, которые пересылаются с использованием интернет-протоколов».

7.1. Язык XML при работе с Web-сервисами

Структура XML-документа. Язык XML был создан для хранения, транспортировки и обмена данными, с его помощью можно реализовать обмен данными между различными системами. XML документ состоит из частей, называемых элементами. Элементы составляют основу XML-документов. Они образуют структуры, которые можно обрабатывать программно или с помощью таблиц стилей. Элементы размечают именованные разделы информации. Элементы строятся с помощью тегов разметки, обозначающих имя, начало и конец элемента. Элементы могут быть вложены друг в друга, на верхнем уровне находится элемент, называемый элементом документа или корневым элементом в котором содержатся остальные элементы. Например:

```
<?xml version-"1.0"?>
<planets>
        <planet ID="1">
                <name>Mercury</name>
         </planet>
         <planet ID="2">
                <name>V enus</name>
         </planet>
<!-- There are more planets. -->
</planets>
```

Документ XML может располагаться в одном или нескольких файлах, причем некоторые из них могут находиться на разных машинах. В XML используется специальная разметка для интеграции содержимого разных файлов в один объект, который можно охарактеризовать как логическую структуру. Благодаря тому, что документ не ограничен одним файлом, XML позволяет создавать документ из частей, которые могут располагаться где угодно.

Документ XML обычно содержит следующие разделы:

- XML-декларация;
- Пролог;
- Элементы;
- Атрибуты;
- Комментарии.

Декларация XML-документа. XML-декларация обычно находится в первой строке XML-документа. XML-декларация не является обязательной. Однако, если она существует, она должна располагаться в первой строке документа, и до нее не должно быть больше ничего, в том числе пробелов. XML-декларация в схеме документа состоит из следующих элементов:

• Номер версии:

<?xml version="1.0"?>.

Это обязательный аргумент. Текущая версия — 1.0.

• Декларация кодировки:

<?xml version="1.0" encoding="UTF-8"?>

Это необязательный параметр. Если он используется, то декларация кодировки должна располагаться сразу после информации о версии в XML-декларации. Декларация кодировки должна содержать значение, представляющее собой существующую кодировку символов.

• Декларация автономности, например:

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>. Декларация автономности, как и декларация кодировки, необязательны. Если декларация автономности используется, то она должна стоять на последнем месте в XML-декларации.

Пролог XML-документа. Прологом называются данные, расположенные после открывающего тега документа или после корневого элемента. Он включает сведения, относящиеся к документу в целом — кодировка символов, структура документа, таблицы стилей.

<?xml version-'1.0" encoding="UTF-8"?>

<?xml-stylesheet type="text/xsl" href="book.xsl"?>

<!DOCTYPE book SYSTEM "schema.dtd">

<!--Some comments-->

В языке XML есть возможность включения в документ инструкций, которые несут определенную информацию для приложений, которые будут обрабатывать тот или иной документ. Инструкции по обработке в XML создаются следующим образом.

<? Приложение Содержимое ?>

В XML инструкции по обработке заключаются в угловые кавычки со знаком вопроса. В первой части процессинговой инструкции определяется приложение или система, которой предназначена вторая часть этой инструкции или ее содержимое. При этом инструкции по обработке действительны только для тех приложений, которым они адресованы. Примером процессинговой инструкции может быть следующая инструкция:

<?serv cache-document?>

7.2. Элементы XML-документа

Элементы в XML документе отвечают за организацию информации и являются основными структурными единицами языка XML. Элементы оформляются следующим образом:

<ElementName> Содержимое элемента </ElementName>

Теги устанавливают границы вокруг содержимого элемента, если таковое имеется.

У каждого элемента должно быть имя. Имена XML-элементов должны подчиняться следующим правилам:

- Названия могут содержать буквы, цифры и другие символы;
- Названия не могут начинаться с цифры или знака препинания;
- Названия не могут начинаться с букв xml;
- В названии не должно быть пробелов.
- Нельзя допускать пробелов у кавычек (<);
- Имена элементов являются регистрозависимыми;

- Все элементы должны иметь закрывающий тэг.

В XML элементы могут быть двух типов - пустые и непустые. Пустые элементы не содержат в себе никаких данных, таких как текст или другие конструкции, и могут сокращенно записываться следующим образом:

<ElementName />

В XML документе обязательно должен присутствовать единственный корневой элемент, все остальные элементы являются дочерними по отношению к единственному корневому элементу. При этом должен строго соблюдаться порядок вложенности элементов:

<person>

<givenName>Peter</givenName>

<familyName>Kress</familyName>

</person>

В данном случае элемент <person> содержит два других элемента, <givenName> и <familyName>. Элемент <givenName> содержит текст Peter, а элемент <familyName> — текст Kress.

7.3. Атрибуты XML-документа

В XML элементы могут содержать атрибуты с присвоенными им значениями, которые помещаются в одинарные или двойные кавычки. Атрибуты позволяют добавлять сведения об элементе с помощью пар «имя-значение». Атрибуты часто используются для определения тех свойств элементов, которые не считаются содержимым элемента, хотя в некоторых случаях (например, HTML-элемент img) содержимое элемента определяется значениями атрибута. Атрибуты могут отображаться в открывающих или пустых тегах, но не в закрывающих тегах. Атрибут для элемента задается следующим образом: <myElement attribute="value" ></myElement>

Синтаксические правила создания атрибута:

- Декларируются в открывающем тэге;
- Количество атрибутов не ограничено;
- Несколько атрибутов разделяются пробелами;

- Атрибут состоит из имени и значения
    - Каждое имя должно быть уникально в рамках одного элемента;
    - Нельзя использовать пробелы в именах атрибутов;
    - Значение атрибута должно быть в кавычках.

Значение атрибутов может заключаться как в одинарные, так и в двойные кавычки. Возможно также использование одних кавычек внутри других, например:

<myElement attribute="value" > </myElement>

<myElement attribute-'value' ></myElement>

<myElement attribute="'value"' ></myElement>

Комментарии XML-документа. Содержимое, не предназначенное для синтаксического анализа (например, замечания о структуре документа или редактировании), можно заключить в комментарии. Комментарии начинаются с группы символов <!-- и заканчиваются группой символов -->:

<!—Текст комментария -->

Комментарии могут находиться в прологе документа, в том числе в определении типа документа (DTD), после документа и в текстовом содержимом. Комментарии не могут находиться внутри значений атрибутов. Они также не могут находиться внутри тегов.

Синтаксический анализатор считает концом комментария символ >; после этого символа он возобновляет обработку XML-документа в обычном режиме. Поэтому строка > не может находиться внутри комментария. За исключением этого, в комментариях могут использоваться любые допустимые символы XML. Поэтому комментарии очень полезны, если нужно убрать комментарий из области обработки синтаксического анализатора, не удаляя само содержимое из документа. Для временного удаления разметки можно использовать следующие комментарии:

<!— <test pattem-'SECAM" /><test pattem-'NTSC" /> --><!--Текст комментария -->

При создании комментария необходимо учитывать следующее:

1 .В тексте комментария не может быть двух символов «-» подряд.

2.Комментарий не может заканчиваться символом «-».

7.4. Текстовые данные XML-документа

Благодаря поддержке набора символов Юникода XML поддерживает целый диапазон символов, в том числе буквы, цифры, знаки препинания и другие символы. Большинство управляющих символов и символы совместимости Юникода не допускаются.

Весь текст XML-документа анализируется парсером, т.е. является парсируемым (PCDATA), но в случае необходимости определенные разделы можно «скрыть» для проверки и они будут игнорироваться парсером (т.н. непарсируемые данные, CDATA). Разделы CDATA дают возможность сообщить средству синтаксического анализа, что среди символов, содержащихся в разделе CDATA, отсутствует разметка. Это упрощает создание документов с разделами, в которых могут появиться отдельные символы разметки, но на самом деле разметки нет. В разделы CDATA часто помещают содержимое на языке сценариев, а также образцы содержимого XML и HTML.

Раздел CDATA в схеме документа использует следующую конструкцию:

<![CDATA[Some information using <, >,]]>

При обнаружении начального тега <![CDATA[, средство синтаксического анализа XML рассматривает все последующее содержимое как символы, не пытаясь интерпретировать их как разметку элемента или сущности. Ссылки на символы в разделах CDATA не работают. Обнаружив завершающий тег]]>, средство синтаксического анализа возобновляет нормальный синтаксический анализ. Например, в XML-документ можно включить любой из приведенных далее разделов CDATA, и средство синтаксического анализа не воспримет их как ошибочные.

<![CDATA[</this is malformed!</malformed</malformed & worse>]]> или <script>

```
<![CDATA[
        function matchwo(a,b) {
```

```
if (a < b && a < 0) then
  {
        return 1;
  }
      else
        {
              return 0
        }
}
```

]]>

</script>

Содержимое разделов CDATA может содержать только символы, разрешенные для содержимого XML; нельзя экранировать таким образом управляющие символы и символы совместимости. Кроме того, в раздел CDATA не может входить последовательность]]>, поскольку эти символы означают завершение раздела. Это значит, что разделы CDATA не могут быть вложенными друг в друга. Эта последовательность также используется в некоторых сценариях. В сценариях обычно можно вместо последовательности]]> использовать]]>.

Помимо непарсируемых данных в XML-тексте можно использовать ссылки на символы и сущности. Ссылки на символы и сущности позволяют включать данные в XML-документ, ссылаясь на них, вместо того чтобы вводить символы в документ напрямую. Такой способ удобно применять в следующих ситуациях:

- Символы нельзя ввести в документ напрямую, так как они будут интерпретироваться как разметка.
- Символы нельзя ввести в документ напрямую из-за ограничений устройства ввода.
- Невозможна надежная передача символов через процессор, ограниченный однобайтными символами.

- Символьная строка или фрагмент документа часто повторяются и могут быть сокращены.

Для представления содержимого в XML используются числовые или синтаксические конструкции, начинающиеся с символа амперсанда (&) и заканчивающиеся точкой с запятой (;). Ссылки на символы позволяют вставлять символы Юникода, для которых в качестве кодовой точки Юникода задан числовой код. Кодовые точки можно задавать либо в десятичном, либо в шестнадцатеричном представлении:

& #value; - синтаксис, используемый для десятичных ссылок.

&# xvalue; - синтаксис, используемый для шестнадцатеричных ссылок.

Например, чтобы вставить символ евро, который до сих пор отсутствует на многих клавиатурах, можно вставить в документ ссылку € или €.

В приведенной таблице 1 перечислены пять встроенных сущностей для символов, используемых в XML-разметке:

Таблица 1.

Встроенные сущности XML

Сущность	**Ссылка на сущность**	**Значение**
lt	<	< (меньше чем)
gt	>	> (больше чем)
amp	&	& (амперсанд)
apos	'	' (апостроф или одиночная кавычка)
quot	"	" (двойная кавычка)

В ситуациях, когда символ может привести к ошибочной интерпретации структуры документа средством синтаксического анализа XML, необходимо использовать сущность вместо ввода символа. Ссылки на сущности ' и " чаще всего используются в значениях атрибутов. Например, чтобы написать «Me&You», надо использовать Me&You. Чтобы написать «a<b», надо использовать a<b. Чтобы написать «b>c», нужно использовать b>c.

Можно определить собственные сущности, как HTML определяет набор сущностей для использования в HTML. Значение &ap не распознается как HTML-файл. Для преобразования в HTML нужно использовать $# .....

При работе с определением типа документа (DTD), определяющим сущности, можно ссылаться на эти сущности в содержимом документа, используя следующий синтаксис:

&entityName;

При обработке пробелов в документе XML необходимо учитывать, что в спецификации XML, опубликованной консорциумом World Wide Web (W3C), различные способы обозначения конца строки приведены к единому способу обозначения, однако при этом сохраняются все остальные пробелы, за исключением пробелов в значениях атрибутов. Кроме того, в XML предусмотрен набор средств, с помощью которых документы могут сообщать приложениям, следует ли сохранять пробелы. Согласно существующему стандарту XML 1.0 пробелы перед XML- декларацией не допускаются.

```
<?xml version="1.0"?>
<root>
    <element>something</element>
</root>
```

Если перед XML-декларацией есть пробелы, они будут рассматриваться как инструкция по обработке. Информация (в частности, о кодировке) не обязательно используется средством синтаксического анализа.

Средства синтаксического анализа XML обязательно сообщают обо всех пробелах, найденных в содержимом элементов внутри документа. Поэтому следующие три документа с точки зрения средства синтаксического анализа XML будут различными:

Документ 1:

```
<root>
    <data>1</data>
    <data>2</data>
```

```
    <data>3</data>
</root>
```

Документ 2:

```
<root><data> 1</data><data>2</data><data>3</data></root>
```

Документ 3:

```
<root> <data>1</data> <data>2</data> <data>3</data> </root>
```

Для XML-приложений, ориентированных на документы, пробелы могут быть чрезвычайно важны.

Авторы документов могут использовать атрибут xml:space для обозначения частей документов, где пробелы представляют определенную важность. Атрибут xml:space также может использоваться в таблицах стилей, обеспечивая сохранение пробелов в представлении документа. Однако поскольку многие XML-приложения не распознают атрибут xml:space, его использование может быть лишь рекомендацией.

Атрибут xml:space может принимать два значения:

- default - это значение позволяет приложению обрабатывать пробелы при необходимости. Если не включить атрибут xml:space, то результат будет такой же, как при использовании значения default.
- preserve - это значение показывает приложению, что пробелы следует сохранить в неизменном виде, так как они могут представлять важность.

Значения атрибутов xml:space применяются ко всем потомкам содержащего атрибут элемента, кроме случаев, когда значение переопределяется в одном из дочерних элементов.

Например, в следующих документах относительно пробелов задано одинаковое поведение:

Документ 1:

```
<poem xml:space="defaulf’>
    <author>
        <givenName>Alix</givenName>
        <familyName>Krakowski</familyName>
```

```
    </author>
    <verse xml:space="preserve">
        <line>Roses are red,</line>
        <line>Violets are blue.</line>
        <signature xml:space="default">-Alix</signature>
    </verse>
</poem>
```

Документ 2:

```
<poem xml:space="default">
    <author xml:space="default">
        <givenName xml:space="default">Alix</givenName> <familyName
        xml:space="default">Krakowski</familyName>
    </author>
    <verse xml:space="preserve">
        <line xml:space="preserve">Roses are red,</line>
        <line xml:space="preserve">Violets are blue.</line>
        <signature xml:space="default">-Alix</signature>
    </verse>
</poem>
```

В обоих примерах приложение получает сообщение, что все пробелы в строках стихотворения следует сохранить, но пробелы в других частях документа обрабатываются по необходимости.

По умолчанию службы Microsoft XML Core Services (MSXML) не обрабатывают атрибут xml:space. Если приложение должно учитывать атрибут xml:space, то перед синтаксическим анализом для свойства preserveWhiteSpace объекта DOMDocument следует задать значение True.

```
xmldoc= new ActiveXObject("Msxml2.DOMDocument.5.0");
xmldoc.preserveWhiteSpace = true;
xmldoc.load(url);
```

В службах MSXML также предусмотрены параметры, которые позволяют

перепоручить обработку пробелов в приложении средству синтаксического анализа. Дополнительные сведения см. в разделе «White Space and the DOM» (пробелы и модель DOM) документации по пакету MSXML SDK.

Приложения по обработке XML сохраняют все пробелы в содержимом элементов, но часто нормализуют пробелы в значениях атрибутов. Символы табуляции, символы возврата каретки и пробелы обрабатываются как единичные пробелы. В определенных типах атрибутов удаляются пробелы, находящиеся до или после тела значения. Повторяющиеся пробелы внутри значения заменяются на один пробел. (При наличии определения DTD эта обработка проводится для всех атрибутов, не принадлежащих к типу CDATA). Например, XML- документ может содержать следующие данные:

```
<whiteSpaceLoss note1="this is a note." note2="this
is
a
note.">
```

Средство синтаксического анализа XML передаст оба значения атрибута в виде "this is a note." с преобразованием разрывов строк в одиночные пробелы.

Средства синтаксического анализа XML обрабатывают сочетание символов возврата каретки и перевода строки (CRLF) как одиночные символы возврата каретки или перевода строки. Все они передаются как единичные символы перевода строки (LF). При сохранении документов приложения могут использовать соответствующие соглашения об обработке конца строки.

7.5. Пространства имен XML-документа

Поскольку имена элементов в XML не фиксированы, часто случаются конфликты, когда два различных документа используют одно и то же имя для описания двух различных типов элементов, например:

```
<Order>
    <Employee>
        <Name>Jane Doe</Name>
        <Title>Developer</T itle>
```

```
    </Employee>
    <Product>
        <Title>The Joshua Tree</Title>
        <Artist>U2</Artist>
    </Product>
</Order>
```

Пространства имен XML дают возможность избежать конфликтов имен элементов. Они задаются при помощи уникальных идентификаторов URI. Чтобы упростить работу, были разработаны специальные префиксы пространств имен, которые позволяют с легкостью определить, какой схеме принадлежит тот или иной элемент документа. Общий синтаксис:

```
<ElementName xmlns:Prefix-'http://contoso.msft/namespace_for_examples">
    <Prefix:AnyElement>Some Data</Prefix:AnyElement>
    <AnotherElement>More Data</AnotherElement>
```

Например:

```
    <Order xmlns:hr="http://hrweb" xmlns:mkt="http://market">
    <hr:Name>Jane Doe</hr:Name>
    <hr:T itle>Developer</hr:T itle>
    <mkt:Title>The Joshua Tree</mkt:Title>
    <mkt:Artist>U2</mkt:Artist>
</Order>
```

Префиксы пространств имен задаются как атрибуты с именами, которые начинаются последовательностью xmlns. Созданный префикс пространств имен может использоваться только в собственном имени и во вложенных элементах, но не за пределами элемента в котором он был создан. Сами префиксы не относят элемент к той или иной схеме. Эту функцию выполняют уникальные идентификаторы, которые поставлены в соответствие этим префиксам. Таким образом, два элемента с разными префиксами которым заданы одинаковые идентификаторы будут считаться принадлежащими к одной схеме.

Существует еще один способ, который позволяет не проставлять

префиксы в элементах. Для этого достаточно задать пространство имен по умолчанию. В этом случае все вложенные элементы будут принадлежать пространству имен родительского элемента. При этом не теряется возможность использовать другие пространства имен для дочерних элементов. Для этого достаточно вручную прописать нужное пространство имен, используя атрибут xmlns.

```
<Order>
<Employee xmlns="http://hrweb">
        <Name>Jane Doe</Name>
        <Title>Developer</T itle>
    </Employee>
    <Product xmlns="http://market">
        <Title>The Joshua Tree</Title>
        <Artist>U2</Artist>
    </Product>
</Order>
```

В представленном примере прослеживается так называемое наследование. Если для элемента не указано пространство имен, то ему автоматически присваивается пространство имен ближайшего родительского элемента.

Обычно вышеприведенный способ не очень популярный и чаще всего используются префиксы пространств имен. Но в некоторых случаях их можно опустить и использовать способ с заданием пространства имен по умолчанию.

Появление имени тега без префикса в документе, использующем пространство имен, означает, что имя принадлежит пространству имен по умолчанию (default namespace).

В рамках схемы можно определять новые типы данных, например:

```
<element name = "student"
<complexType>
```

<element name = "name" type = "xsd: string" /> <element name = "rating" type = "xsd: float" /> </complexType>

</element>

В данном случае определяется новый комплексный тип.

7.6. Протокол XML-RPC.

Непосредственным предшественником Web-сервисов являлся протокол XML-RPC.

Это очень простой протокол, предназначенный для вызова удаленных процедур. В отличие от традиционного RPC для вызова удаленной процедуры используются XML-послания. Ответ приходит также в форме XML-послания.

Рассмотрим пример. Пусть имеется удаленная процедура, осуществляющая поиск синонимов в словаре. Для обращения к процедуре используется строка вида public String getSynonym (String word). В качестве аргумента используется исходное слово (word). Процедура возвращает слово — синоним исходного.

Запрос в рассматриваемом случае выглядит следующим образом:

```
<?xml version="l.0"?>
<methodCall>
    <methodName>getSynonym</methodName>
    <params>
        <param>
            <value>
                <string>cai*«MeT</string>
            </value>
        </param>
    </params>
</methodCall>
```

Корневым является элемент <methodCall>, в который вложен элемент <methodName>, в теле которого записывается имя вызываемой процедуры. Параметры вызываемой процедуры помещаются в элемент <params>. В элемент

<params> вложены нуль или несколько элементов <рагаш>, в которые помещаются параметры вызываемой процедуры.

В рассматриваемом примере требуется получить синоним слова «самолет».

Ответ также поступает в форме XML-сообщения:

```
<?xml version="1.0"?>
<methodResponse>
    <params>
        <param>
            <value>a3pomiaH</value>
        </param>
    </params>
</methodResponse>
```

Если произошла ошибка при выполнении запроса, то в ответном пакете указывается код ошибки. XML-RPC позволяет использовать такие типы данных как целые строки, вещественные, структуры, массивы.

Механизм XML-RPC был создан в середине 1990-х гг., однако не получил широкого распространения. Причины этого достаточно понятны и состоят в следующем:

- запросы, выполняемые в рамках XML-RPC, только отчасти являются самоописываемыми, поскольку пространства имен в нем не определены;
- XML-RPC подобно классическому RPC не поддерживают работу с объектами;
- при работе с XML-RPC обнаруживаются проблемы, вызванные использованием XML. На стороне клиента необходимо сформировать запрос. На стороне сервера необходимо его разобрать. То же самое требуется сделать с ответом. Все это усложняет код и требует существенных временных затрат;

- сообщения, отправляемые в формате XML, за счет тегов имеют большую избыточность.

Несмотря на отмеченные недостатки, данная технология получила дальнейшее развитие [25].

7.7. Протокол SOAP.

SOAP — это основанный на XML протокол, определяющий механизм обмена сообщениями. Протокол SOAP появился в 1998 г. как W3C спецификация. В ранних версиях спецификации SOAP расшифровывался как Simple Object Access Protocol. В последних версиях этот термин никак не расшифровывается. На данный момент написания была версия 1.2 от апреля 2007 г. (спецификация находится по адресу https://www.w3.org/TR/soap/).

Хотя SOAP имеет много общею с XML-RPC, но имеются и принципиальные отличия от него.

Во-первых, протокол SOAP не различает вызов процедуры и ответ на него, а просто определяет формат послания (message) в виде документа XML, который может содержать вызов процедуры, ответ на него, запрос на выполнение каких-то других действий или просто текст.

Во вторых, SOAP-послание представляет собой самоописываемый документ, включающий, в частности, описания пространств имен. Структуру SOAP-послания определяет соответствующая XML Schema. Для пересылки SOAP-посланий обычно используется метод HTTP POST, хотя можно использовать и другие протоколы, такие как FTP, SMTP.

SOAP-послание включает два элемента: заголовок (Header) и тело (Body), которые помещаются в контейнер (Envelope). SOAP-послание представляет собой правильный XML-документ. Заголовок является факультативным элементом, а тело — обязательным.

В качестве примера рассмотрим пример сервера, который находит синоним слова с посредством вызова метода getSynonym:

```
<?xml version="l.0" encoding="Windows-1251"?>
<SOAP-ENV:Envelope
```

```
SOAP-ENV:encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAPENV="
    http://schemas.xmlsoa.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/
        XMLSchema-instance"
    xmlns:SOAPENC="
    http://schemas.xmlsoap.org/soap/encoding/"> <SOAP-ENV:Body>
    <getSynonym>
    <word xsi:type="xsd:string">a3ponnaH</word>
    </getSynonym>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

В теле запроса в элемент word, который имеет тип XSD string, помещается исходное слово. В ответ на запрос клиенту поступает ответное послание, которое имеет вид:

```
<?xml version="l.O" encoding="Windows-1251"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/ envelope/"
    xmlns:xsd=http://www.w3.org/2001/XMLSchema
    xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance >
    <SOAP-ENV:Body>
        <getSynonymResponse
        SOAP-ENV:encodingStyle=
        http://schemas.xmlsoap.org/soap/encoding/ >
        <getSynonymResult xsi:type=
        "xsd:string">самолет</getSynonymResult>
        </getSynonymResponse>
    </SOAP-ENV:Body>
```

```
</SOAP-ENV:Envelope>
```

Рассмотрим пример создания простейшей службы, которая может находить синоним.

Имеется много разных систем поддержки работы с Web-сервисами. Одной из популярных является Apache extensible Interaction System (AXIS).

Создание серверной части не вызывает проблем. Для работы с AXIS достаточно написать текст сервиса. Для этого создаем файл SynonymService.java:

```
// SynonymService.java
public class SynonymService {
        public String getSynonym(String word) {
                //Находим в словаре требуемый синоним
                .....
        //Возвращаем синоним
        return "" + synonym;
}
}
```

Данный файл необходимо переименовать в SynonymService.jws и не компилируя поместить в каталог webapps (при работе с AXIS). После этого сервис готов к использованию.

Клиентская часть приложения. При работе с AXIS вначале создается объект класса service, который предназначен для установления связи с Web-сервисом. Он предоставляет экземпляр класса call, в который заносятся параметры запроса, в частности, адрес и имя Web-сервиса «getSynonym». После того как запрос сформирован, AXIS обращается к Web-сервису посредством вызова метода invoke(). Запрос направляется серверу приложений, работающему по указанному в запросе адресу. На сервере запускается сервлет AxisServiet, разбирающий SOAP-послание. Он находит требуемый jws-файл, компилирует и запускает с требуемыми параметрами. После выполнения требуемых действий сервлет формирует SOAP-послание, в которое помещается

результат. Послание отправляется клиенту. Там оно разбирается. Для клиента результат доступен как результат выполнения метода invoke() [15].

7.8. WSDL-описание

WSDL используется для описания интерфейсов Web-сервисов. Средствами WSDL можно описать, где находится Web-сервис и каким образом к нему следует обращаться. WSDL-описание позволяет создавать независимое от платформы и языка реализации описание Web-сервиса. Нетрудно заметить, что WSDL имеет много общего с IDL, используемым в RPC. WSDL-описание представляет собой XML-документ, структура которого показана на рисунке.

В качестве корневого элемент WSDL-описания используется элемент «definitions;», в который вкладываются семь элементов, пять из которых являются основными, а два — вспомогательными.

В качестве основных выступают элементы <types>; <message>; <portType>; <binding>; <service>. В качестве вспомогательных выступают элементы <import>, <documentation>.

Таблица 2.

Структура XML-документа

<deinitions>

<types>	Что делается
<message>	
<portType>	
<binding>	Как делается
<service>	Где находится
<import>	
<documentation>	

</deinitions>

Каждый элемент имеет собственное имя, определяемое обязательным атрибутом name. Имена используются для ссылки на отдельные элементы.

Элемент <types> присутствует, если требуется определить сложные типы с помощью языка XSD. В противном случае данный элемент отсутствует.

Элемент <message> описывает каждое из SOAP посланий в терминах «запрос-ответ». В этот элемент вкладываются элементы <part>. При использовании процедурного стиля в каждый такой элемент описывается имя и тип одного аргумента запроса или возвращаемого значения. При использовании документного стиля элементы <part> могут описывать отдельную часть послания MIME-типа «multipart/related».

Элемент <por<Type> описывает интерфейс Web-сервиса, который также называют пунктом назначения (endpoint) или портом (port) сервисов, называемых операциями. Отдельная операция описывается как вложенный элемент <operation>, который описывает отдельный сервис. Имеются четыре вида сервисов: два простых действия (получение послания и отправка ответа) и два комбинированных действия (отправка послания — получение ответа и получение послания — отправка ответа). Действия типа получение и отправка посланий описываются вложенными элементами <input> и <output>, а также сообщением об ошибке (элемент <fault>). Элемент <serviceType> содержит множество вложенных элементов <portType>. Один и тот же порт может быть связан с несколькими сервисами.

Элемент <binding> описывает формат пересылки послания: протоколы и его тип. Каждый элемент <message> может быть связан с несколькими такими элементами < binding >, по одному для каждого способа пересылки.

Элемент <service> определяет местонахождение сервиса как один или несколько портов, каждый из которых описывается вложенным элементом <port>, содержащим адрес интерфейса Web-сервисы.

Элемент <lmport> включает файл с XSD схемой описания WSDL или другой WSDL-файл.

Элемент <documentation> представляет собой комментарий, который можно включить в любой элемент описания WSDL.

Принято говорить, что элементы <types>, <message> и <portType> определяют, какие именно услуги предоставляет данный сервис.

Элементы <binding> определяет, как реализована Web-служба, т. е. как к ней обращаться.

Элементы <service> определяет, где располагается сервис.

В качестве примера рассмотрим фрагмент WSDL-описания.

```
<?xml version="l.0" encoding="UTF-8"?>
name="SynonymService"
targetNamespace="http://www.vocab.ru/Thesaurus.wsdl"
xmlns=http://www.w3.org/ns/wsdl
.....
<wsdl: definitions.....
    <wsdl:message name="getSynonymResponse">
        <wsdl:part name="synonym" type= "SOAP-ENC:string"/>
    </wsdl:message>
    <wsdl:message name="getSynonymRequest">
        <wsdl:part name="word" type=
            "SOAP-ENC:string"/>
    </wsdl:message>
    <wsdlrportType name="Synonyms">
        <wsdl:operation name="getSynonym"
            parameterOrder="word">
                <wsdl:input message="intf:
                    getSynonymRequest"/>
                <wsdl:output message="intf:
                getSynonymResponse"/>
            </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="getSynonymSoapBinding"
        type="intf:Thesaurus">
            <wsdlsoap:binding style="rpc"
                transport="http://schemas.xmlsoap.org/soap/ http"/>
```

```
<wsdl:operation name="getSynonym">
<wsdlsoap:operation soapAction=""/>
<wsdl:input>
<wsdlsoap:body
encodingStyle="http://schemas.
xmlsoap.org/soap/encoding/"
namespace="urn:myDirectory"
use="encoded"/>
</wsdl:input>
<wsdl:output>
<wsdlsoap:body encodingstyle=
http://schemas.xmlsoap.org/soap/ encoding/
namespace="urn:myDirectory" use="encoded"/>
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="SynonymService">
<wsdl:port binding="intf: getSynonymSoapBinding" name="Thesaurus">
<wsdlsoap:address
location="http://localhost:8080/axis/ services/Thesaurus"/>
</wsdl:port>
</wsdl:service>
<documentation> Thesaurus WSDL File</documentation>
</wsdl: definitions>
```

Предполагается, что данное описание находится в файле Thesaurus, wsdl.

Имя сервиса SynonyraService определено в элементе service.

Далее определяются используемые пространства имен (часть определений пропущена).

В элементах <wsdl:message> описаны используемые типы SOAP посланий. Используются два типа сообщений, которые называются

getSynonymRespons и getSynonymReques. Первое содержит поле synonym, а второе — поле word. Оба поля представляют собой строки символов.

Элемент <wsdl:portType> называется Synonyms и выполняет только одну операцию getSynonym, которая получает входную переменную word. Входное послание имеет имя getSynonymRequest, а выходное — getSynonymResponse.

Элемент <wsdl:binding»> называется getSynonymSoapBinding. В нем указывается, что сервис работает в режиме запрос-ответ (использует грс-стиль), использует в качестве транспорта протокол HTTP, работает с SOAP-посланиями и выполняет операцию getSynonym. Затем в терминах SOAP повторяется описание операции getSynonym.

В элементе <wsdl:service>, который называется SynonymService, указывается местонахождение сервиса.

Элемент <documentation> может содержать любые комментарии.

Поскольку в рассматриваемом примере не используются сложные типы, то элемент <types> отсутствует.

WSDL-спецификация может быть получена автоматически из файла реализации Web-сервиса, например, из Java-файла или из описания интерфейса. Имеется много разных систем поддержки работы с Web-сервисом. Одной из наиболее популярных является Apache extensible Interaction System (AXIS).

Можно выделить два альтернативных подхода к разработке Web-сервисов: подход по принципу «сверху-вниз» и подход «снизу- вверх».

В первом случае разработка начинается с разработки WSDL-опи- сания, которое используется для генерации скелетона и стаба, затем к ним добавляется бизнес-логика.

Во втором случае разработка начинается с создания кода для генерации WSDL-описания.

Например, в рамках существующих инструментальных средств имеются утилиты для выполнения упомянутых выше преобразований [25, 13].

8. ОБЩИЕ ПРИНЦИПЫ ОРГАНИЗАЦИИ ВЗАИМОДЕЙСТВИЙ В ИНФОРМАЦИОННЫХ СИСТЕМАХ

Системы обмена данными подразделяют на системы, использующие асинхронную и синхронную связь, а также на системы, работающие по принципу сохранной и несохранной связи [27].

При использовании асинхронной связи (asynchronous communication) отправитель после отправки сообщения немедленно продолжает работу, а сообщение сохраняется в локальном буфере передающего хоста или на ближайшем коммуникационном сервере. При использовании синхронной связи (synchronous communication) работа отправителя блокируется до того момента, когда сообщение будет доставлено получателю, либо сохранено в локальном буфере принимающего хоста.

При использовании синхронной связи (synchronous communication) можно выделить три степени «жесткости»:

- ✓ отправитель может продолжать работу после того, как сообщение помещено во входной буфер получателя;
- ✓ работа отправителя блокируется до момента получения сообщения непосредственно пользователем, в этом случае от пользователя часто требуется подтвердить прием сообщения;
- ✓ работа отправителя блокируется до момента получения ответа.

При сохранной связи (persistent communication) сообщение, предназначенное для отсылки, хранится в коммуникационной системе до тех пор, пока его не удастся передать получателю. Сообщение сохраняется на коммуникационном сервере до тех пор, пока его не удастся передать на следующий коммуникационный сервер. Поэтому у приложения, отправляющего сообщение, нет необходимости после отправки сообщения продолжать работу. Приложение, принимающее сообщение, не обязательно должно находиться в рабочем состоянии во время отправки сообщения.

При использовании сохранной связи сообщения никогда не теряются и не пропадают.

Альтернативой использования механизмов сохранной связи является использование несохранные связи (transient communication. При применении

нерезидентной связи сообщение хранится в системе только в течение времени работы приложений, которые отправляют и принимают это сообщение.

Если коммуникационный сервер не имеет физической возможности передать сообщение следующему серверу или получателю, то сообщение уничтожается. Следует отметить, что все коммуникационные сервисы транспортного уровня поддерживают только нерезидентную связь.

На практике применяются различные комбинации этих типов взаимодействия.

Системы, основанные на вызове удаленных процедур или обращении к удаленным объектам, по большей части являются синхронными и реализуют несохранные связи. Хотя после того, как стало очевидно, что данные виды связи не всегда самые подходящие, были созданы средства для менее жестких форм нерезидентной синхронной связи, таких как асинхронные вызовы ИРС и отложенные синхронные операции.

Значительная часть существующих приложений представляет собой распределенные приложения, причем очень часто речь идет об интеграции уже существующих подсистем в единую ИС.

Типовыми проблемами, возникающими при создании распределенных ИС и, в частности КИС, являются следующие:

- ✓ различия между приложениями;
- ✓ необходимость внесения изменений в код интегрируемых приложений;
- ✓ ограниченная скорость передачи данных;
- ✓ ненадежность сетевой инфраструктуры.

Отдельные приложения, входящие в состав распределенной системы, могут работать на разных аппаратных и программных платформах; написаны могут быть на разных языках программирования, использовать разные форматы данных и т.д.

Некоторые приложения на этапе разработки не были рассчитаны на работу в составе распределенных ИС, и их интеграция требует внесения изменений в исходный код.

Практически все интеграционные решения предполагают наличие каналов передачи данных устройствами. В отличие от процессов, выполняющихся в пределах одного хоста, в распределенных ИС данные передаются через маршрутизаторы, коммутаторы, общедоступные сети и спутниковые каналы связи, что может приводить к задержкам и рискам потери и искажения данных.

Обычно выделяют четыре базовых механизма интеграции приложений, входящих в состав распределенной ИС [27]:

- разделяемые файлы;
- разделяемая база данных;
- удаленный вызов процедуры и методов;
- обмен сообщениями.

Разделяемые файлы. Несколько приложений имеют доступ к одному и тому же файлу. Одно приложение создает файл, а другое считывает его. Приложения должны согласовать имя файла, его расположение, формат, время записи и считывания, а также процедуру удаления.

Это один из самых старых подходов к интеграции приложений. Основная идея данного подхода состоит в том, что файл рассматривается как универсальный механизм обмена данными. Можно выделить два альтернативных подхода: распределенные файловые системы и системы, основанные на пересылке файлов.

При использовании распределенных файловых систем обмен осуществляется через общие файлы, которые включаются в состав файловых систем взаимодействующих приложений.

Альтернатива рассмотренному выше варианту взаимодействия приложений состоит в том, что файлы, в которых содержится информация, определяющая взаимодействие, пересылаются между хостами. По этому

принципу построена, например, система Unix-Unix Copy и ряд других систем. При использовании данного подхода одним из наиболее важных решений является выбор общего формата файлов. В ранних приложениях наиболее распространенным стандартным форматом файлов являлся простой текстовый файл. В современных интеграционных решениях обычно используется XML-формат.

Основное достоинство рассматриваемого подхода — простота, поскольку единственными общедоступными интерфейсами приложений являются создаваемые этими приложениями файлы. В данном случае имеет место слабый вариант связи между приложениями. Кроме того, данный подход обусловливается отсутствием необходимости в привлечении дополнительных средств или пакетов для интеграции. Вместе с тем это приводит к возрастанию нагрузки, ложащейся на плечи разработчиков интеграционного решения. Объединяемые приложения должны использовать общее соглашение об именовании и расположении файлов.

Один из наиболее существенных недостатков передачи файла заключается в сложности синхронизации процессов и сложности разработки кода.

Данный вариант взаимодействия может иметь смысл, если взаимодействие между приложениями носит эпизодический характер.

Разделяемая база данных. Основная идея данного подхода состоит в том, что данные хранятся в центральной базе данных, доступной для всех интегрируемых приложений. Общая база данных обеспечивает согласованность хранящейся в ней информации. Синхронизация доступа к данным реализуется посредством использования, например механизма транзакций.

Самый простой способ реализации общей базы данных состоит в использовании реляционной СУБД с поддержкой SQL. Язык запросов SQL поддерживается практически всеми платформами для разработки приложений, что позволяет не беспокоиться о различии в форматах файлов и избавляет от необходимости изучения новых языков программирования и новых технологий.

Все вопросы, связанные с интерпретацией данных, могут быть решены на этапе проектирования и реализации интеграционного решения.

С возрастанием числа обращений к общей базе данных она становится узким местом, что приводит к задержкам в работе приложений. Если обращения к разделяемой базе данных осуществляются через локальную и особенно глобальную сеть, то это лишь усугубляет ситуацию с задержками.

Подход к интеграции, основанный на использовании разделяемой базы данных, предполагает использование общей логической структуры данных.

Разделяемая база данных представляет собой неинкапсулирован- ную структуру. Изменения в приложении могут потребовать изменений в структуре базы данных, что, в свою очередь, скажется на других приложениях. Поэтому организации, использующие общую базу данных, обычно без энтузиазма относятся к необходимости ее изменения, что затруднит внедрение новых бизнес-решений.

Удаленный вызов процедуры и методов. С точки зрения интеграции приложений, удаленный вызов процедуры представляет собой применение принципов инкапсуляции данных. Если приложение хочет получить доступ к данным, которые поддерживаются другим приложением, то оно обращается к требуемым данным посредством вызова функции, т. е. каждое приложение самостоятельно обеспечивает целостность своих данных и может изменять их формат, не затрагивая при этом другие приложения.

Удаленный вызов процедуры и работа с удаленными объектами поддерживается множеством технологий, таких как RPC, Java RMI, CORBA, DCOM, .NET Remoting. Несмотря на внешнюю схожесть, удаленный вызов процедуры и вызов локальной функции имеют принципиальные различия, способные оказать существенное влияние на интеграционное решение [15]. Следует отметить, что удаленный вызов процедуры характеризуется самой сильной степенью связывания приложений.

Обмен сообщениями. Идея обмена сообщениями состоит в том, что реализуется асинхронный механизм взаимодействия между приложениями,

который позволяет им регулярно обмениваться небольшими порциями информации. Асинхронный обмен сообщениями устраняет большинство недостатков, присущих системам, основанным на вызове удаленных процедур, поскольку для передачи сообщения не требуется одновременной доступности отправителя и получателя. Кроме того, сам факт использования асинхронного обмена данными побуждает разработчиков к созданию приложений, не требующих частого удаленного взаимодействия.

9. ИНТЕГРАЦИЯ ПРИЛОЖЕНИЙ

Рассмотрим типы интеграционных задач. В широком смысле под термином «интеграция» можно понимать объединение ИТ-систем и отдельных приложений, входящих в их состав, интеграцию компаний (бизнеса) или людей. В более узком смысле под интеграцией можно понимать объединение отдельных приложений в ИТ-систему, объединение отдельных ИТ-систем в более крупную систему и организацию взаимодействия между отдельными ИТ-системами по принципу B2B.

Интеграция приложений может принимать различные формы, прежде всего можно выделить внутреннюю и внешнюю интеграцию. Внутреннюю интеграцию обычно называют интеграцией корпоративных приложений (Enterprise Application Integration, EAI), а внешнюю — интеграцией бизнес-бизнес (Business-to-Business Application Integration, B2B).

Исходя из последнего определения выделим четыре типовых подхода к решению задачи интеграции [15]:

- ✓ интеграция на уровне данных;
- ✓ бизнес-функции и бизнес-объекты;
- ✓ бизнес-процессы;
- ✓ порталы.

Конечно же приведенный выше список ни в коем случае не претендует на звание исчерпывающей классификации задач интеграции. Тем не менее он дает определенное представление о проектах в области интеграционных решений.

Некоторые интеграционные задачи объединяют в себе сразу несколько типов интеграции.

Интеграция на уровне данных. Данный подход называют также интеграцией, ориентированной на информацию (Information-Oriented Integration) [15].

Этот подход ориентирован, в первую очередь, на интеграцию данных, которые хранятся в базах данных и обычно имеет целью создание API, позволяющего программисту унифицированным образом работать с множеством БД, которые могут быть территориально разнесены и принадлежать разным производителям. В рамках данного подхода можно выделить, по крайней мере, три группы технологий:

- ✓ системы репликации баз данных;
- ✓ федеративные базы данных;
- ✓ использование API для доступа к стандартным ERP-системам.

В процессе функционирования многих ИТ-систем приложениям часто требуется доступ к одним и тем же данным.

Например, такая информация, как адрес проживания клиента, может использоваться как системой гарантийного обслуживания кли ентов, так и подразделениями, занимающимися рекламой. Некоторые из этих систем могут иметь собственные хранилища данных.

При изменении адреса клиента каждая из подсистем должна получить копию обновленной информации. Этого можно добиться с помощью такого типа интеграции, как репликация данных. Существует множество различных способов реализации репликации данных.

Функция поддержки репликаций может быть встроена в СУБД; нужные сведения можно экспортировать в файл для последующего импорта в другой системе, а также переслать внутри сообщений с помощью соответствующего промежуточного ПО.

Федеративные базы данных (Federated Database System) — это системы, которые позволяют прозрачным для пользователя образом

интегрировать множество автономных баз данных, которые могут располагаться на разных хостах сети. Федеративные базы данных называют также виртуальными БД.

Федеративная (виртуальная) БД предоставляет пользователю единый хорошо определенный интерфейс для доступа к распределенным данным, при этом сами данные не перемещаются и не изменяются, т.е. нет препятствий для того, чтобы одна и та же автономная БД входила в состав более чем одной виртуальной БД.

Использование API для доступа к стандартным ERP-системам предполагает использование хорошо определенных интерфейсов для организации взаимодействия создаваемых пользовательских приложений с такими пакетными приложениями, как Enterprise Resource Planning (ERP) системы, SAP, Oracle, PeopleSoft. Обычно это делается посредством использования адаптеров (коннекторов).

Бизнес-функции и бизнес-объекты. Во многих ИТ-системах можно выделить функциональность, которая является общей для нескольких приложений, входящих в состав ИТ-системы. Например, в рассмотренном выше бизнес-приложении эта информация об адресах покупателей.

Каждую из таких функций можно вынести за пределы приложений и реализовать в виде функций совместного использования, доступных всем системам в виде сервисов (служб). В частности, для рассматриваемого примера можно создать, например сервис GetCustomerAddress.

Если организация разрабатывает несколько проектов, в которых используется данная бизнес-функция, то будет разумно сделать ее общей для нескольких проектов.

Отдельные бизнес-функции можно объединить для создания более сложной бизнес-функции, например такой, как постановка изделия на гарантийное обслуживание. Если используется СОА, то данный подход применяется для создания бизнес-сервисов. Если используется компонентный подход, то создаются бизнес-объекты (бизнес- компоненты).

Совместно используемая бизнес-функция и репликация данных могут преследовать схожие цели. Например, копирование адреса проживания клиента во все требуемые системы можно заменить созданием совместно используемой бизнес-функции GetCustomerAddress. Выбор между двумя разными типами интеграции основывается на многочисленных критериях, таких как степень контроля над интегрируемыми системами (в отличие от помещения информации в базу данных, вызов совместно используемой функции предполагает более глубокое вмешательство в систему) и частота изменения данных (доступ к адресу проживания клиента осуществляется часто, а вот вероятность изменения последнего невысока).

Бизнес-процессы. Данный подход имеет много общего с описанным выше подходом, основанным на использовании бизнес-функций. Основное различие заключается в том, что появляется новый уровень интеграции — уровень бизнес-процессов.

Бизнес-процессы работают поверх уровня сервисов и используют собственный язык для описания последовательности вызова сервисов. Этот язык представляет собой интерпретируемый язык, во многом схожий с такими языками, как Basic или shell.

Бизнес-процессы, обеспечивающие внутреннюю интеграцию, и бизнес-процессы, обеспечивающие B2B-интеграцию, во многом различны. В бизнес-процессах, ориентированных на внутреннюю интеграцию, обычно задействовано достаточно большое количество сервисов. Типовая задача B2B-интеграции состоит в организации взаимодействия между двумя сервисами. Это может быть, например бизнес-процесс приобретения некоторого товара, при котором стороны договариваются о цене и оформляют покупку.

Порталы. Основная функция информационных порталов заключается в обеспечении представления информации из нескольких источников. В качестве таких источников могут выступать, в частности, приложения, которые участвуют в реализации некоторой функции, реализованной средствами

бизнес-процессов. Порталы, как правило, реализуют персонифицированный доступ к информации, и его вид может настраиваться пользователем.

Порталы можно рассматривать как графический интерфейс бизнес-процессов, в которых участвуют конкретные пользователи. Рассмотрим их более подробно.

9.1. Порталы и портлеты

Порталом (от лат. porta — ворота) принято называть Web-приложение, которое предоставляет пользователю Интернета или Интранета доступ к различным сервисам. Часто термин «портал» определяет единую точку доступа пользователя в информационное пространство, при этом предполагается, что пользователь, зайдя на портал, может получить доступ ко всем необходимым для него источникам информации и приложениям, поэтому порталы иногда определяют как рабочий стол (десктоп) нового поколения.

В процессе работы пользователи, как правило, имеют дело с данными разных форматов и происхождения и используют для их обработки различные приложения. Рабочее место, кроме того, предоставляет средства для интеграции людей и, в частности, средства для коллективной работы. Типовое портальное решение обеспечивает пользователя средствами поиска, обеспечением безопасности, доступом к системам электронного обучения и корпоративного документооборота.

По своей идее портал — это Web-сайт, ориентированный на удовлетворение информационных потребностей определенной категории пользователей. При этом каждый пользователь может настраивать портал под свои собственные нужды и может получать доступ к нему с помощью любого устройства, имеющего доступ в Интернет.

С точки зрения пользовательского интерфейса портал представляет собой многооконное интегрированное приложение. Каждое окно — это «зона», изображение в которой формируется отдельным приложением. За каждую зону отвечает отдельное приложение, однако приложения могут связываться между

собой как автоматически, так и по требованию пользователя, при этом информация может синхронизироваться.

Часто разработчики называют порталами многостраничные сайты. Следует отметить, что не всегда бывает просто провести границу между порталом и сайтом, однако можно выделить, по крайней мере, три существенных различия между сайтом и порталом.

Во-первых, портал многофункционален. Титульная страница портала содержит все или почти все, что нужно для работы конкретного пользователя.

Во-вторых, портал предоставляет пользователю преимущественно динамический контент, который генерируется активными программными компонентами, имеется возможность синхронизации и обмена контентом между приложениями.

Во-третьих, портал может быть персонализирован, т. е. состав и внешний вид рабочего места зависят от роли пользователя и, кроме того, может быть изменен и настроен для удобства работы конкретного пользователя или группы пользователей.Типовой портал кроме возможности одновременной работы с несколькими приложениями обеспечивает пользователю доступ к ряду сервисов общего назначения, таких как:

- сервис однократной регистрации, обеспечивающий аутентификацию пользователя при работе с порталом;
- сервис настройки и персонализации, позволяющий настраивать внешний вид, функциональность и информационное наполнение портала для нужд конкретного пользователя с учетом его роли;
- сервисы доступа к приложениям (сервисы обработки);
- сервисы доступа к бизнес-процессам, позволяющие пользователю быть участником бизнес-процесса в соответствии с определенной ему ролью;
- сервисы доступа к данным, обеспечивающие подключение пользователя к источниками консолидированной информации, такими как СУБД или хранилища данных;
- сервис поиска информации в Интернете и Интранете;

- сервис совместной работы, позволяющий пользователям работать в команде для решения общей задачи (разделяемые библиотеки документов, доски объявлений, онлайновые конференции, новостные группы);
- сервис публикации, позволяющий пользователю сохранять документы в хранилище контента портала;
- сервис подписки, который позволяет пользователю оформлять подписку и получать уведомления об изменении или появлении новой информации. При этом правила доставки/извещения и фильтрации могут настраиваться пользователем;
- сервис администрирования, позволяющий управлять правами доступа пользователей, создавать и удалять пользователей.

Большинство порталов представляют собой Web-порталы, работающие по принципу тонкого клиента, рабочим окном которых является окно Web-браузера.

С точки зрения назначения принято выделять три основных типа порталов:

- горизонтальные;
- вертикальные;
- корпоративные.

Горизонтальные, или общедоступные, порталы ориентированы на самую широкую аудиторию. Обычно контент таких порталов носит общий характер. Как правило, это новостная информация, рассылки, электронная почта и т. п.

Горизонтальные порталы имеют много общего со средствами массовой информации и могут рассматриваться как порталы общего назначения. В качестве примеров горизонтальных порталов могут выступать такие известные порталы, как Rambler, Lycos, Excite, Yahoo!

Вертикальные порталы можно рассматривать как специализированные порталы, предназначенные для информационного обслуживания конкретных групп пользователей. В качестве примеров вертикальных порталов могут служить порталы B2C (Business-to-Consumer), B2B (Business-to-Business), а

также порталы типа B2E (Business-to-Employees), которые обычно называют корпоративными порталами.

Примерами B2C-порталов могут служить порталы турфирм, предоставляющие такие услуги, как заказ билетов, бронирование мест в гостиницах и т. п.

B2B-порталы позволяют пользователям клиентам реализовывать совместные бизнес-операции, такие как выбор поставщиков, закупку товаров, проведение и участие в аукционах и т.д. Число B2B-порталов растет быстрыми темпами.

B2E-порталы, которые обычно называют корпоративными порталами, они предназначены для сотрудников, клиентов и партнеров одного предприятия.

Пользователи корпоративных порталов обычно получают доступ к сервисам и приложениям в зависимости от роли, назначенной конкретному пользователю.

Основным назначением корпоративного портала является предоставление внешним и внутренним пользователям возможности персонифицированного доступа к сервисам, приложениям и корпоративным данным — объединение изолированных моделей бизнеса, интеграция различных корпоративных приложений, включая приложения бизнес-партнеров, обеспечение доступа как стационарных, так и мобильных пользователей к корпоративным ресурсам независимо от местонахождения пользователя.

Следует отметить, что корпоративные порталы развиваются быстрыми темпами. Обычно принято выделять четыре поколения корпоративных порталов [11].

Корпоративные порталы первого поколения ориентированы на предоставление пользователю преимущественно статического Web-контента и Web-документов.

Обычно целью разработки таких порталов является создание единой точки доступа к корпоративной информации, распределенной по разным

подразделениям организации. Такие порталы, как правило, реализует следующий типовой набор функций:

- персонализация;
- системы поиска информации;
- управление контентом и его агрегация;
- наличие средств и инструментов интерации приложений.

В корпоративных порталах второго поколения появляются такие черты, как наличие персонализации контента, присутствие поисковика. Эти порталы ориентированы на использование в качестве составной части КИС, и для них характерны:

- надежная среда реализации приложений;
- мощные инструменты разработки и интеграции приложений;
- соответствие требованиям, предъявляемым к ИС масштаба предприятия;
- поддержка интеграции с ИС партнеров;
- наличие поддержки мобильного доступа к ресурсам.

Корпоративные порталы третьего поколения ориентированы

преимущественно на предоставление сервисов в отличие от порталов первого и второго поколений, которые ориентированы на предоставление контента. Отличительной чертой порталов третьего поколения является то, что в них реализуется идея сотрудничества (collaboration). Порталы третьего поколения предоставляют возможность сотрудникам работать в виртуальном офисе и предоставляют такие возможности как чаты, e-mail, возможность групповой работы над проектами. Порталы третьего поколения — это преимущественно корпоративные порталы.

Для четвертого поколения корпоративных порталов характерны:

- ориентация на электронный бизнес, что подразумевает интеграцию с модифицируемыми, переносимыми в новое окружение приложениями;
- возможность работы не только с сервисами, но и с политиками;

- пользователям предоставляется возможность получать доступ к информации с помощью многих типов устройств, в частности мобильных устройств.

Современный корпоративный портал обычно представляет собой продукт или набор продуктов, который базируется на определенной инфраструктуре, в качестве которой, как минимум, выступают серверы приложений и серверы баз данных.

В составе типовой КИС, ориентированной на использование порталов можно выделить три основных функциональных слоя:

- базовая инфраструктура;
- слой интеграции приложений;
- интерфейсный слой.

Базовая инфраструктура отвечает за предоставление таких базовые сервисов, как управление пользователями, управление безопасностью, управление транзакциями др.

Слой интеграции приложений обеспечивает взаимодействие портала с приложениями, такими как ERP- и CRM-системы, унаследованные приложения, СУБД и др.

К интерфейсному слою принадлежат визуальные и невизуальные компоненты порталов, называемые обычно портлетами. Интерфейсный слой, включающий в себя средства управления информационным наполнением, адаптеры для обмена данными с информационными системами бизнес-партнеров, интерфейсные модули для поддержки взаимодействия с мобильными устройствами и др.

9.2. Портлеты

Информационное наполнение порталов часто предоставляется пользователям в форме портлетов-контейнеров. С точки зрения реализации, портлет представляет собой фрагмент кода, исполняемый на портальном сервере. Когда говорят о портлетах, то речь ведется в терминах JEE, поскольку

эта платформа поддерживает работу с портлетами, хотя порталы с успехом можно строить и на других платформах.

Портлет — это приложение, предоставляющее пользователю некоторую информацию или сервис. Он может быть включен в качестве составной части в портальную страницу. Портлет работает под управлением контейнера, который обрабатывает запросы и генерирует динамический контент, и представляет собой plugin, ответственный за представление информации пользователю.

Динамически сгенерированный контент портлета называют фрагментом. Контент нескольких портлетов можно объединять в рамках одной портальной страницы.

Клиент, обычно это Web-клиент, взаимодействует с портлетом в режиме запрос-ответ. Для разных пользователей портлет может иметь разный внешний вид в зависимости от настроек.

Портлеты могут реализовываться на разных платформах. При реализации на Java-платформе портлет рассматривается как Web- компонент.

На данный момент рабочей является версия 2.0 и дальнейшее изложение ведется применительно к данной спецификации.

Контейнер портлетов представляет собой среду, в которой «живут» портлеты. Управляя жизненным циклом портлетов, контейнер не отвечает за агрегацию портлетов.

Обычно портал или точнее портальный движок и контейнер могут реализовываться как один монолитный компонент или как два независимых компонента.

В самом общем виде работа с портальной страницей происходит следующим образом:

- Web-клиент посылает HTTP запрос к порталу;
- портал получает запрос и определяет, к какому из портлетов, относящихся к данной портальной странице, относится запрос;
- портал запускает портлет через контейнер и получает требуемый контент;

• портал агрегирует полученный контент в портальную страницу и отправляет ее клиенту.

Следует заметить, что портлеты имеют много общего с сервлетами, в частности:

• как сервлеты, так и портлеты, являются Web-компонентами;

• подобно сервлетам портлеты работают под управлением контейнера;

• портлеты подобно сервлетам генерируют динамический контент;

• клиент работает через Web -интерфейс.

Отличие портлетов от сервлетов состоит в следующем:

• портлет представляет собой только часть изображения, портальная страница формируется из фрагментов средствами портального сервера;

Web-клиент взаимодействует с портлетом не напрямую, а через ортальную систему;

• форматом отображения портлета можно управлять;

• на одной портальной странице один и тот же портлет может появляться несколько раз.

Кроме того, портлеты по сравнению с сервлетами, обладают следующей дополнительной функциональностью:

• портлеты работают с конфигурационными файлами;

• портлеты имеют доступ к профилям пользователей;

• портлеты могут сохранять свое состояние;

• портлеты могут взаимодействовать друг с другом и контейнером посредством сообщений.

Контейнер портлетов можно рассматривать как расширение Web-контейнеров.

Портлет генерирует фрагмент в форме гипертекста. Портал формирует окно портлета посредством добавления к сгенерированному фрагменту обрамления, включающее рамку и кнопки. Затем окна портлетов агрегируются в портальную страницу.

Для формирования портлетов могут использоваться либо JSP, либо такие фреймворки, как Java Server Faces (JSF), Struts, Spring [26] и др.

Портлеты могут функционировать в нескольких режимах. Пользователь может работать с информационным наполнением, может настраивать внешний вид портлета в соответствии со своими предпочтениями, а администраторы могут конфигурировать портал для предоставления. Режим, который пользователи выбирают, определяет, какой интерфейс портлета они видят. Представление может находиться в одном из следующих состояний: нормальное (normal), максимизированное (maximized), минимизированное (minimized), закрытое (closed).

Свойства, существенные для размещения портлетов, хранятся в дескрипторе.

Сервер порталов предоставляет портлету сервис получения информационного наполнения и сервис сохранения состояния между том числе предпочтения пользователя, данные о настрой^ ке и т.д.

API портлетов определяет интерфейс между портлетом и контейнером портлетов.

Концепция порталов создавалась постепенно в течение достаточно длительного промежутка времени. Еще за много лет до появления самого терминов «портал» и «портлет» разработчики сайтов широко использовали такие приемы, как введение динамического контента, фреймы, элементы персонализации страницы. Однако различные Web-серверы и серверы приложений обеспечивали эти возможности разными способами. С появлением концепции портала появилась необходимость стандартизации средств построения портальных приложений. В рамках технологии JEE в 2003 г. появилась спецификация JSR-168, которая была разработана совместно фирмами IBM и Sun Microsystems. Она определяла портлеты, написанные на языке Java.

В 2005 году представителем компании IBM был инициирован запрос на спецификацию JSR 286, в котором предлагалось создать новую версию

спецификации Java-портлетов для согласования с концепциями J2EE версии 1.4, а также другими JSR (например, с JSR 188) и спецификацией WSRP второй версии.[16]. Предыдущая версия спецификации JSR 168 никак не затрагивала проблемы интеграции, определяя только компонентную модель. Поэтому вопросы интеграции и межпортлетной коммуникации предлагалось специфицировать в новой версии. Работы над второй версией (V2.0) продлились до 12 июня 2008 года, когда её финальный релиз был утверждён экспертной группой, включающей в себя всех значимых разработчиков порталов, как коммерческих, так и с открытым кодом, разработчиков средств интеграции портлетов и разработчиков сред разработки портлетов. Обе версии совместимы на двоичном уровне и это означает, что все портлеты, создание в соответствии со спецификацией JSR-168, могут работать в контейнере, созданном в соответствии со спецификацией JSR-286.

На практике портлеты, созданные в соответствии со специцификацией JSR-168, продолжают активно использоваться.

Функции для работы с портлетами определены в пакете javax. portlet. API портлетов определены таким образом, чтобы можно было провести четкую границу между портлетом и портальной инфраструктурой.

Хотя API портлетов во многом похожи на API сервлетов, но имеются и отличия, в частности, контейнер портлетов различает и позволяет поддерживать как состояние портлета, так и состояние окна. Состояние окна определяет, какую часть экранной страницы занимает данный портлет. Окно может находиться в одном из трех состояний: нормальном, минимизированном или максимизированном.

В соответствии со спецификацией JSR-168 определяются следующие методы интерфейса Portlet: init(), processAction(), render(), destroy().

Метод init() используется для инициализации портлета.

Метод processAetionO вызывается для обработки действий пользователя в портлете.

Метод render() вызывается при необходимости перерисовки изображения портлета.

Метод destroy() вызывается перед удалением портлета из памяти.

Портлет может находиться в одном из трех состояний:

• View (Представление);

• Edit (Редактирование);

• Help (Помощь).

В состоянии «Представление» портлет реализует свою функциональность, в состоянии «Редактирование» реализуются функции, связанные с настройка портлета, а в состоянии «Помощь» отображаются подсказки.

Наряду с интерфейсом Portlet в пакете определяется класс GenericPortlet, где метод render() определен таким образом, что управление в зависимости от текущего состояния портлета передается одному из методов: doView(), doEdit() или doHelpQ. Большинство реальных портлетов наследуют от класса GenericPortlet.

Ключевым при работе с портлетом безусловно является метод processAction(), который работает с объектами типа запрос и ответ — объекты ActionRequest и ActionResponse соответственно. Для метода render() и вызываемых им методов doXxx() в качестве аргументов и результатов выступают объекты RenderRequest и RenderResponse. Используя эти объекты, портлет может управлять состоянием и взаимодействовать с другими портлетами, сервлетами, принимать данные, вводимые пользователем в экранные формы портлета, создавать изображение для отображения в портале и посылать его клиенту и определять состояние портлета.

Запрос клиента инициируется при помощи ссылок URL, создаваемых портлетом. URL портлета могут быть двух типов: URL действия и URL перерисовки. Портлет создает свои URL, вызывая методы createActionURL и createRenderURL интерфейса RenderResponse. Эти методы возвращают интерфейс PortletURL. URL перерисовки может быть уточнен указанием

состояния портлета, для которого этот URL применяется. Обычно запрос клиента, инициируемый URL- действия, превращается в один запрос действия и много запросов перерисовки — по одному на каждый портлет на странице портала. Если портлет может кешироваться (это определяется в дескрипторе портлета), метод render может не вызываться.

Портлеты — настраиваемые элементы. Настройки портлета сохраняются в виде наборов пар «имя — значение». За сохранение конфигурации портлета отвечает контейнер. Программист работает с настройками посредством объекта типа PortletPreferences, который имеет методы setValue() и getValue() установки (изменения) значений настроек и чтения их значений соответственно.

Портлеты упаковываются как стандартные в JEE Web-архивы (файлы WAR). Кроме дескриптора развертывания web.xml они дополнительно содержат дескриптор портлета — portlet.xml, в котором определены элементы конфигурации портлета.

Основными средствами обеспечения безопасности портлетов в рамках спецификации JSR-168 являются возможность установки в дескрипторе портлета флажка, требующего, чтобы портлет работал только через защищенный протокол HTTPS и возможность аутентификации пользователей и ролей. При этом не определяется, каким образом реализованы пользователи и их роли.

Кроме того, определяется библиотека тегов JSP, помогающая отображать портлет при помощи технологии Java Server Pages. При реализации портлетов довольно часто в методе, например doView(), выполняется логика, связанная с анализом состояния, например некоторая бизнес-логика, а для отображения портлет вызывает страницу JSP, формирующую код фрагмента HTML-страницы.

При вызове страницы JSP ей передаются запрос и отклик портлета. Перед вызовом страницы портлет может сохранить какие-то объекты как атрибуты запроса.

Страница может направлять запрос в портлет, определяя адрес портлета через теги специальной библиотеки.

За агрегацию фрагментов, поставляемых разными портлетами в полную страницу портала, отвечает контейнер портлетов.

Стандартизация портлетов (как и само осознание концепции порталов) несколько задержалась. На момент создания спецификации уже существовало большое число реализаций серверов порталов, в которых использовались фирменные средства и собственные API. Спецификация JSR-168, однако, была принята производителями серверов порталов, и в настоящее время большинство продуктов этого класса проходят или уже прошли процесс адаптации к стандарту. Вместе с тем эти продукты вынуждены поддерживать и свои старые API, так что различия в диалектах API быстро исчезнуть не могут.

Следует отметить, что спецификация JSR-168 определяет только базовые средства работы с портлетами. Обычно развитые портальные серверы предлагают многочисленные и не предусмотренные спецификацией дополнительные возможности. При этом у каждого из производителей имеются собственные расширения для работы с портлетами.

В 2007 г. в рамках Java Community Process была опубликована окончательная вторая версия спецификации портлетов (JSR-286), которая дополняет предыдущую спецификацию, обеспечивая стандартизацию возможностей, не охваченных в JSR-168. В качестве новых элементов, появившихся в рамках спецификации JSR-268, можно выделить следующие:

- обеспечение механизмов обмена событиями между портлетами: портлет может объявлять события, которые он хочет генерировать, и события, которые он хочет получать; контейнер работает как диспетчер событий;
- поддержка работы с фильтрами, которая выражается в возможности преобразовывать «на лету» как входную, так и выходную информацию;
- обеспечение возможности для портлетов совместно использовать общий сеанс (сеанс пользователя) и данные, относящиеся к общему сеансу;

• обеспечение поддержки работы фреймворками Java AJAX, Server faces, Struts и др.

Обычно портал получает информационное наполнение для своих портлетов из многих как внутренних, так и внешних источников. Для того чтобы интегрировать новый источник, администратор портала должен адаптировать информационное наполнение к формату, который воспринимается порталом, что может оказаться весьма длительным и трудоемким процессом.

Возможны два альтернативных подхода к решению данной задачи.

Первый подход предполагает, что в каждом случае создается портлет, обеспечивающий пользовательский интерфейс для выполняемой сервисом функции: ввод параметров запроса и представление результатов. На основании полученных данных портлет получает требуемые данные, например из БД, либо формирует SOAP-запрос к сервису и превращает отклик сервиса в графическое экранное представление. Все портлеты работают на одном портальном сервере. Подобный подход требует написания кода портлета, обеспечивающего пользовательский интерфейс, извлечение данных и развертывания портлета. Альтернативный подход заключается в том, что удаленный Web-сервис сам будет генерировать фрагмент разметки страницы, который непосредственно будет включаться в страницу нашего портала.

Второй подход предполагает использование специальных посредников, использование которых позволяет провайдерам поставлять контент без ручных настроек.

Идея состоит в том, что вместо добавления самого портлета в портальный сервер туда помещают его заместителя, который взаимодействует с удаленным портлетом.

Сам удаленный портлет поддерживается другим сервером порталов или модулем исполнения портлетов. Последний представляет собой упрощенный сервер порталов (среда исполнения портлетов), который дает возможность удаленному портлету реагировать на вызовы посредника. Он может быть

реализован в другой технологии, например .Net, но при этом должен поддерживать протокол удаленного вызова портлета.

Для обеспечения совместимости между портальными серверами разных производителей и поставщиками информационного наполнения требуется стандартизованная модель взаимодействий для посредника портлета и удаленного портлета.

Подобный стандарт создан в рамках организация Organization for the Advancement of Structured Information Standards (OASIS) — Web-службы для удаленных порталов (Web services for remote portals, WSRP).

Спецификация WSRP является продуктом международной организации Organization for the Advancement of Structured Information Standards (OASIS). Организация по развитию стандартизации структурированной информации является консорциумом, содействующим принятию технических стандартов. Версия 1.0 спецификации WSRP была закончена в 2003 г., а версия 2.0 появилась в 2008 г. [15].

Спецификация WSRP определяет интерфейс для взаимодействия с подключаемыми, ориентированными на представление Web-сервисами, которые обеспечивают взаимодействие с пользователями и выступают в качестве поставщиков фрагментов кода на языке разметки для агрегирования в портальные страницы. WSRP представляют собой визуальные компоненты, которые можно подключать с использованием технологии визуального проектирования. В определенном смысле удаленные портлеты, обладающие графическим интерфейсом, можно рассматривать как Web-сервис, и для их описания может использоваться WSDL.

WSRP работает следующим образом. Поставщики информационного контента реализуют свой сервис как Web-сервис удаленного портала и публикуют ее, например в UDDI.

Можно привести, по крайней мере, два довода в пользу создания отдельного стандарта Web-сервисов для портлетов.

Во-первых, обычные Web-сервисы ориентированы на обработку запросов и генерацию откликов, выполняемых преимущественно на программном уровне, в то время как портлеты ориентированы на работу с графическим интерфейсом.

Во вторых, требуется четко определенный интерфейс, определяющий то, как портал взаимодействует с сервисом и собирает фрагменты разметки в портальную страницу. Следует заметить, что удаленные и локальные портлеты могут работать на одном портале и для конечного пользователя они неразличимы.

Работа с удаленным портлетом выглядит следующим образом.

Поставщик предлагает один или несколько портлетов и реализует ряд интерфейсов WSRP, обеспечивающих общий набор операций для конечных пользователей. WSRP-портлет представляет собой подключаемый компонент, формирующий интерфейс конечного пользователя. Он выполняется внутри поставщика WSRP и доступен удаленно через интерфейс, определенный этим поставщиком.

Потребитель WSRP представляет собой клиента Web-сервиса, который вызывает WSRP-сервис и обеспечивает среду взаимодействия пользователя с портлетами, предлагаемыми одним или несколькими поставщиками; обычно роль потребителя WSRP играет портал.

Реально Web-сервисом является не WSRP-портлет, а поставщик WSRP, именно он имеет стандартное описание на языке WSDL и набор конечных точек. Обращение к WSRP-портлету возможно только через поставщика. Поставщик принимает SOAP-запрос, выделяет из него обращение к портлету и упаковывает фрагмент разметки, генерируемый портлетом, в ответное SOAP-сообщение. В функции же потребителя WSRP входит упаковка в SOAP-запрос параметров формы, поступившей от пользователя, и выделение из SOAP-отклика фрагмента разметки, присланного поставщиком и WSRP-портлетом.

В рамках спецификации WSRP определены два обязательных и два необязательных интерфейса, которые должны реализовывать поставщики

WSRP и использовать потребители WSRP для взаимодействия с удаленными портлетами. Стандартизация этих интерфейсов дает возможность потребителю WSRP обращаться к любому поставщику.

Обязательными для реализации интерфейсами WSRP являются интерфейс описания сервиса (Service Description Interface) и интерфейс разметки (Markup Interface).

Через Интерфейс описания сервиса потребители могут запрашивать, какие портлеты предлагает поставщик, и получать информацию о самом поставщике.

Интерфейс разметки предназначен для поддержки взаимодействия поставщика с удаленными портлетами. Он позволяет посылать им формы от пользователя портальной страницы и получать от них информацию о текущем состоянии портлета.

Необязательными интерфейсами WSRP являются интерфейс регистрации (Registration Interface) и интерфейс управления портлетом (Portlet Management Interface).

Интерфейс регистрации предназначен для поддержки процесса регистрации потребителя перед тем, как тот сможет обратиться к сервису. Это позволяет поставщику сервиса адаптировать свое поведение применительно к определенному типу потребителя; через этот интерфейс поставщик и потребитель могут обмениваясь сведениями о себе.

Интерфейс управления портлетом позволяет пользователю управлять жизненным циклом удаленного портлета и настроить его поведение.

Все перечисленные выше интерфейсы WSRP представляют собой XML-протоколы и соответственно платформенно независимы.

Следует особо отметить, что WSRP не заменяет стандарты Web-сервисов, а дополняет их и предствляет собой надстройку над ними.

Все взаимодействия между потребителем и поставщиком WSRP осуществляются по протоколу SOAP, а операции интерфейсов WSRP определяются в WSDL-описании WSRP. Использование подобного подхода

позволяет осуществлять публикацию WSDL-описаний WSRP-портлетов в реестрах UDDI.

Типовой сценарий использования WSRP на основе технологии UDDI выглядит следующим образом.

1. Провайдер разрабатывает набор портлетов, назначая WSRP-поставщика и отображая их как WSRP-портлеты. Если провайдер хочет, чтобы эти портлеты использовались как удаленные, то он публикует их описания в реестре.

2. Пользователь, заинтересованный в работе с удаленными портлетами, ищет нужный ему портлет, используемые предоставляемые порталом средства, либо независимое приложение.

3. После обнаружения желаемого портлета пользователь добавляет новое приложение к одной из своих портальных страниц.

4. Если пользователю не разрешено добавление портлетов к своей странице, то администратор портала находит их в UDDI-реестре и сделает их доступными для пользователей, добавив во внутренний реестр портала.

5. После того как удаленный портлет помещен на портальную страницу пользователя, он увидит на своей портальной странице выполняющийся удаленно портлет, при обращении к которому портальный сервер выполняет запрос к соответствующему Web- сервису посредством отправки SOAP-послания, а в ответ получает SQAP-послание с фрагментом на языке разметки страниц, который портал интегрирует в отображаемую страницу. При этом конечный пользователь полностью избавлен от необходимости знать детали работы WSRR

Как у всякой технологии, у технологии портлетов имеются свои области применения достоинства и недостатки.

Использование портлетов безусловно целесообразно в случае, если цель проекта состоит в том, чтобы объединить Web-приложения и информацию в одном удобном месте, если требуется использовать сервисы внешних поставщиков и (или) выступать в качестве поставщика сервиса. Если

пользователи активно перемещаются и пользуются мобильными устройствами, то портлеты — очевидный выбор.

Если цели проекта отличаются от указанных выше, то следует рассмотреть другие альтернативы.

Достоинства портлетов:

- возможность работать на различных клиентских устройствах, что позволяет пользователям перемещаться с компьютера на компьютер и с одного мобильного устройства на другое, при этом использовать ту информацию и те приложения, которые им нужны;
- внешний вид и функциональность портлетов можно настраивать для различных групп пользователей, а сами пользователи могут настраивать внешний вид портлетов в соответствии со своими предпочтениями;
- выполнение портлетов как Web-сервисов в целях обеспечения доступа к ним внешних пользователей;
- разделение сложных приложений на отдельные задачи по принципу — одна группа тесно связанных задач представлена одним портлетом;
- возможность относительно легко добавлять новую функциональность к уже существующим приложениям;
- хорошая совместимость с брандмауэрами (firewalls).

Портлеты не могут быть решением для каждого проекта. Далее перечислены типовые ситуации, когда не следует использовать портлеты.

Недостатки портлетов:

- сложные пользовательские интерфейсы плохо переводятся в портлеты, использующие такие языки разметки, как HTML и WML;
- если требуется реализовать пользовательские интерфейсы с часто обновляемыми данными, то использование портлетов может привести к замедлению работы системы, поскольку, когда обновляется один портлет, то перерисовывается вся портальная страница;
- высокоинтерактивные пользовательские интерфейсы недостаточно хорошо переводятся в Web-приложения, вообще, и в портлеты, в частности.

Если необходимо, чтобы интерфейс приложения изменялся в процессе работы, например необходимо появление выпадающего меню, вы можете либо использовать форму и обновлять всю страницу полностью, либо использовать скрипты, чтобы изменить портлет. Однако всплывающие окна и скрипты обычно не могут использоваться на мобильных устройствах;

- имеются проблемы при работе с формами (внутренние фреймы разрешены, но только пользователи Microsoft Internet Explorer могут их видеть);
- портлеты полностью не стандартизированы, и они еще не поддерживаются на стольких платформах, как другие Java-технологии.

СПИСОК ЛИТЕРАТУРЫ

1. AntiPatterns / W. Brown, R. Malveau, H. I. McCormick, T. Mowbray. — N. Y.: John Wiley, 1998. — 156 pp.
2. Capability Maturity Model Integration. [Eelectronic resource]: http://www. sei.cmu.edu/cmmi/.
3. Chappell D. Enterprise Service Bus / D. Chappell. — Sebastopol: O'Reilly Media, 2004. — 328 pp.
4. Davies J. The Definitive Guide to SOA: BEA AquaLogic Service Bus / J. Davies, A. Krishna, D. Schorow D. — N.Y.: Apress, 2007. — 613 pp.
5. Duffy D. Domain Architectures. Models and Architectures for UML Applications / D. Duffy. — Datasim Education BV, Amsterdam, Netherlands, 2004. - 412 pp.
6. ISO 9126 (ГОСТ Р ИСО / МЭК 9126-93)— «Информационная технология. Оценка программного продукта. Характеристики качества и руководство по их применению». Введ. 1994-06-30. — М.: Стандартинформ, 1994. — 12 с.
7. ISO/IEC 2382:2015 Information technology — Vocabulary
8. ISO/IEC 42010:2011. System and software engineering — Architecture description. — 2011.

9. Kaisler S. Software Paradigms / S. Kaisler. — New Jersey: John Wiley & Sons, 2005. — 458 pp.
10. Kossiakoff A., Sweet W. N., Seymour S. J., Biemer S. M. Systems Engineering Principles and Practice. — 2-е изд. — Hoboken, New Jersey: A John Wiley & Sons, 2011. — 599 с. — ISBN 978-0-470-40548-2.
11. Polgar J. Building and managing enterprise-wide portals / J. Polgar, R. Bram, A. Polgar. — London: Idea Group Publishing, 2006. — 304 pp.
12. Pyster, A., D. Olwell, N. Hutchison, S. Enck, J. Anthony, D. Henry, and A. Squires (eds). Guide to the Systems Engineering Body of Knowledge (SEBoK) version 1.0. — The Trustees of the Stevens Institute of Technology, 2012.
13. Web Services for Remote Portlets Specification v2.0 OASIS standart [Eelectronic resource]: http://docs.oasis-open.org/wsrp/v2/wsrp-2.0-spec-os-02. html.
14. William S. Davis, David C. Yen. The Information System Consultant's Handbook. Systems Analysis and Design. — CRC Press, 1998. — 800 с. — ISBN 0849370019.
15. Архитектура информационных систем: учебник для студ. учереждений высш. проф. образования / Б.Я. Советов, А.И. Водяхо, В.А. Дубенецкий, В.В. Цехановский. – М. : Издательский центр «Академия», 2012.-288 с.
16. Гамма Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р.Хелм., Р.Джонсон., Дж. Влиссидес. — СПб.: «Питер», 2001. — 368 с.
17. ГОСТ 7.0-99. Система стандартов по информации, библиотечному и издательскому делу. Информационно-библиотечная деятельность. Библиография. Термины и определения
18. ГОСТ Р ИСО/МЭК 15288—2008. Системная инженерия — Процессы жизненного цикла систем. — 2008.
19. Данилин А., Слюсаренко А. Архитектура и стратегия. «Инь» и «Янь» информационных технологий предприятия. — М.: Интернет-университет информационных технологий, 2005. — 504 с. — ISBN 5-9556-0045-0.

20. Одиночкина С.В. Основы технологий XML - СПб: НИУ ИТМО, 2013. – 56 с.
21. Стандарт ISO/IEC 15288 «Системная инженерия — процессы жизненного цикла систем» [Электронный ресурс]: http://www.iso.org/iso/support/faqs.htm.
22. Стелтинг С. Применение шаблонов Java. Библиотека профессионала / С. Стелтинг, О.Маасен. — М.: Издательский дом «Вильямс», 2002. — 576 с.
23. Таненбаум Э. Распределенные системы. Принципы и парадигмы / Э.Таненбаум, М. ван Стеен. — СПб.: Питер, 2003. — 877 с.
24. Фаулер М. Архитектура корпоративных программных приложений.: Пер. с англ. — М.: Издательский дом «Вильямс», 2006. — 544 с. ISBN 5-8459-0579-6
25. Хабибуллин И. Ш. Разработка Web-служб средствами Java / И. Ш. Хабибуллин. — СПб.: БХВ-Петербург, 2003. — 400 с.
26. Хемраджани А. Гибкая разработка приложений на Java с помощью Spring, Hibernate и Eclipse / А.Хемраджани. — М.: Издательский дом «Вильямс», 2008. — 352 с.
27. Хоп Г. Шаблоны интеграции корпоративных приложений / Г.Хоп, Б. Вульф. — М.: Издательский дом «Вильямс», 2007. — 672 с.