Фак. ИКСС
Каф. ИКС доц. Шалаев А.Я.
декабрь 2017г

Тема 7.

Среда интеграции Frameworx

- *Технологически нейтральная архитектура TNA NGOSS*
- Среда интеграции Frameworx, SOA
- Интерфейсы API
- *Взаимодействие между компонентами системы посредством общей коммуникационной среды CCV.*

TNA (*Technology Neutral Architecture*) - Технологически нейтральная архитектура

В основе независимой от технологии архитектуры TNA лежит компонентный подход - любая система OSS рассматривается как набор компонентов.

- **Компонент – это элемент архитектуры,** представляющий собой механизм предоставления ***сервисов.***

  В TNA компонент - это единица внедрения, которая предоставляет один или несколько сервисов. Компоненты – это составляющие программного обеспечения, каждый из которых реализует определенные функции.

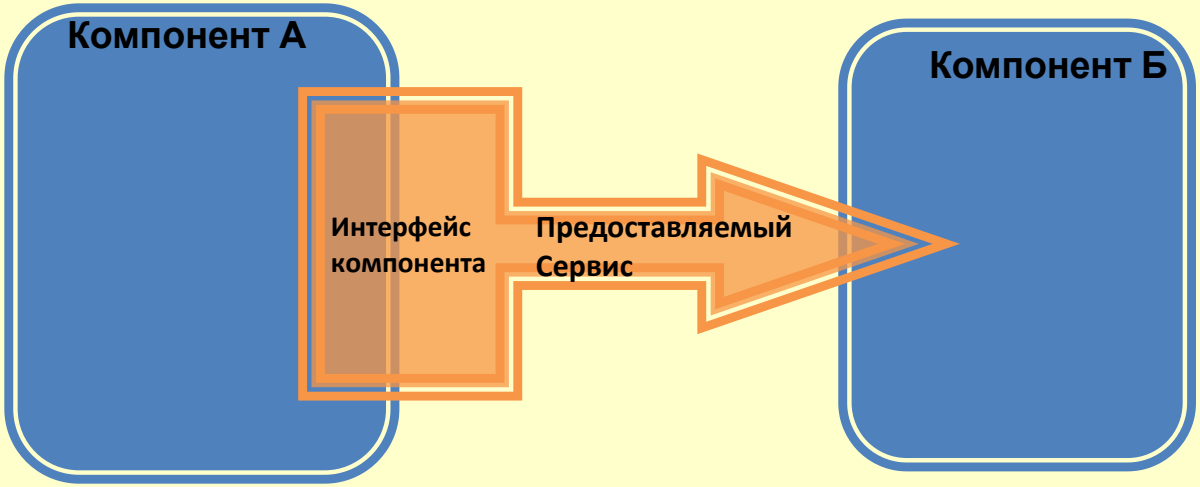
При проектировании решения, система OSS и определяющие ее поведение сервисы описываются в спецификациях, которые не привязаны к какой-либо технологии реализации. После чего в отдельных документах излагаются методы отображения TNA на конкретную технологию для разработки и развертывания решения.

Архитектура TNA не случайно называется технологически нейтральной: она не задает способ или набор технологий для реализации решения, а лишь задает общие принципы, которым должна соответствовать построенная система OSS.

Технологически нейтральная архитектура TNA задает базовые принципы разработки **программного обеспечения**, формирующего OSS-решение. Она регламентирует такие аспекты построения решения, как использование**:**

- единых интерфейсов между компонентами распределенной архитектуры;
- общей среды передачи данных;
- управление политиками и бизнес-процессами.

Компонентный подход



Понятие архитектуры NGOSS/Frameworx базируется на наборе требований, пришедших из практики построения масштабных информационных систем:

1. Архитектура OSS должна базироваться на использовании компонентов. По своей сути компонент является механизмом предоставления одного или нескольких сервисов
2. Взаимодействие компонентов системы должно осуществляться посредством интерфейсов, определение которых содержит описание интерфейса и операций предоставляемого компонентом сервиса
3. В системе OSS программное обеспечение, осуществляющее автоматизацию сквозных бизнес-процессов, должно быть отделено от жестко прописанного поведения компонентов. То есть, система OSS должна состоять из четко определенных сервисов, которые могут быть оркестрованы с использованием технологий управления бизнес-процессами

Требования к архитектуре TNA

1. Определение интерфейсов между компонентами.
2. Технологически нейтральная компонентная модель.
3. Отделение бизнес-процессов и политик от реализации компонентов.
4. Поддержка безопасности.
5. Поддержка применения политик.
6. Совместное использование информации и данных. Для обеспечения интеграции и взаимодействия компонентов должна использоваться единая информационная модель (SID).
7. Прозрачность распределенной архитектуры. *Она обеспечивается:*

- *репозиторием, в который заносится информация, используемая при ее функционировании.*
- *использованием для взаимодействия компонентов единой коммуникационной среды. Ее использование позволяет стандартизировать операции, сообщения и события в масштабе всей системы и передавать их любым ее компонентам.*

рий, хранилище — место, где хранятся и поддерживаются какие-либо данные (программы, объекты, метаданные и т. п.) Русское сообщество Subversion рекомендует использовать вместо термина репозиторий термин ХРАНИЛИЩЕ, поскольку он полностью соответствует как прямому переводу слова «repository» - (склад, хранилище) , так и его понятию.

Описание компонента

- Спецификация
- Интерфейс
- Описание реализации
- Описание внедрения

Достоинства компонентного подхода к построению OSS:

- При разработке приложений, имеются четко определенные требованиями к ним.
- Компоненты могут разрабатываться и тестироваться одновременно и независимо друг от друга
- Возможно использование компонентов от различных производителей
- Использование стандартизованных интерфейсов, с помощью которых компоненты легко интегрируются друг с другом и с унаследованными (иными словами с уже существующими) системами
- Возможно многократно использовать компоненты *(например, при разработке новой системы будут добавляться нужные компоненты, а некоторые можно выбрать из уже имеющихся в наличии).*

Типы компонентов TNA

1. Служебные компоненты (Framework Service Component, FSC)

   предоставляют один или более фундаментальных **инфраструктурных** сервисов, которые обеспечивают функции архитектуры OSS (например, автоматическую интеграцию компонентов);

2. Бизнес-компоненты (Business Service Component, BSC)

   включают в себя сервисы, которые поддерживают (бизнес-) ***функциональность*** архитектуры NGOSS, например биллинг, тарификацию, управление данными о работе сети и т. п.;

3. Управляющие компоненты (Mandatory Business Service Component, MBSC)

   включают в себя сервисы, предоставляющие сервисы управления бизнес-процессами, сервисы политик и безопасности.

Типы сервисов TNA NGOSS

1. Служебные сервисы

- Сервис регистрации
- Сервис репозитория
- Сервис имен
- Сервис местоположения

2. Бизнес-сервисы

- Сервисы поддержания бизнес-функциональности архитектуры NGOSS

3. Сервисы управления

- Сервис управления бизнес-процессами
- Сервис политик
- Сервис безопасности

Описание сервисов

Для описания предоставляемых компонентом Сервисов требуются:

- метаданные, необходимые для описания интерфейса;
- метаданные, необходимые для описания возможных действий над Сервисом;
- для каждого действия описываются результаты, которые могут возникнуть после реализации этого действия.

Метаданные, в общем случае, — это данные, характеризующие или поясняющие другие данные (аналогично метаязыку)

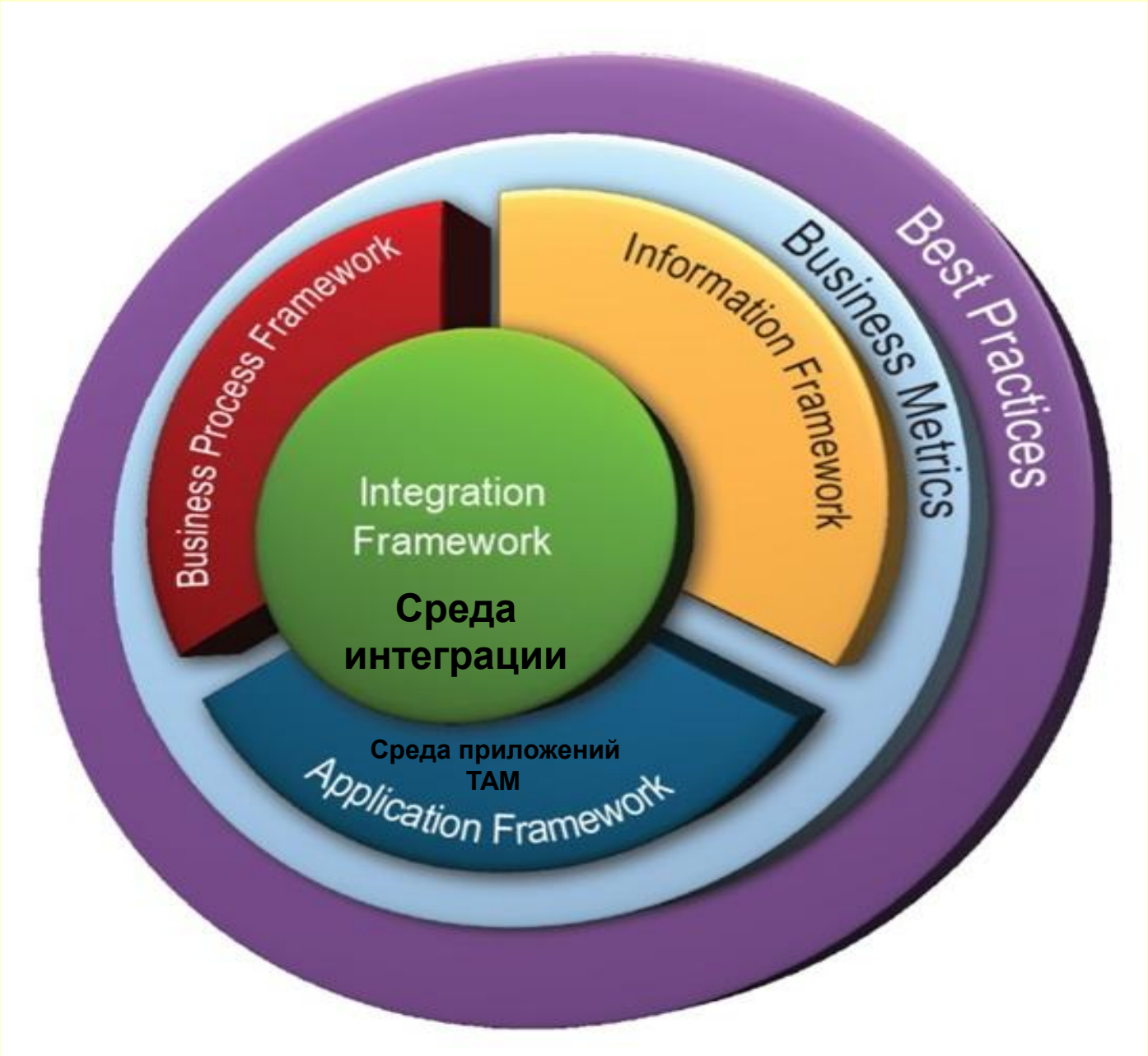
Для описания возможного поведения Сервисов необходимы:

- предусловия, при которых данное действие можно выполнить (т.е. набор условий, которые должны быть реализованы, чтобы действие могло быть произведено);
- постусловия, описывающие все состояния компонента, возможные после выполнения действия.

TNA предполагает следующие **ограничения**:

- Архитектура не должна определять интерфейсы и сервисы внутри компонента;
- Она должна определять поведение компонента в целом и то, каким образом компонент делает свои сервисы доступными (через какие интерфейсы).

Интегрированные среды Frameworx

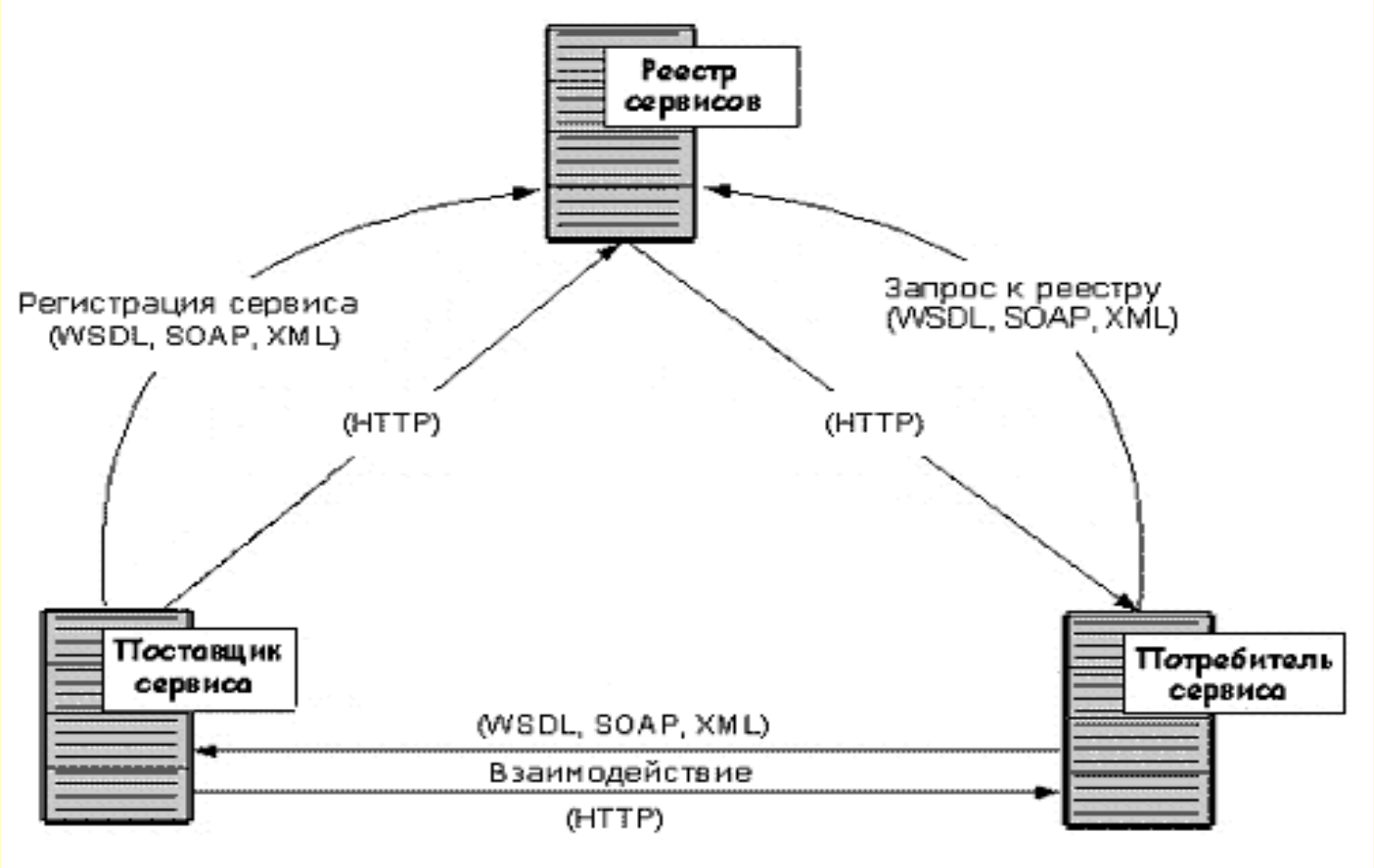


Принцип **SOA** (*Service-Oriented Architecture)* сервисно-ориентированной архитектуры – основа интеграции систем во Frameworx

SOA – подход к организации распределенных информационных систем, в котором компоненты системы предоставляют набор выполняемых ими функций в виде так называемых **сервисов**, к которым можно обращаться стандартизованным способом.

- В данном контексте **сервис** – некоторый абстрактный ресурс, имеющий имя, по которому к нему можно многократно обращаться извне, и способный выполнять на заданном уровне безопасности и по определенным правилам некоторые функции на основе получаемой им информации.
- Отличительная особенность сервисного подхода состоит в том, что между модулями взаимодействующих информационных систем нет навсегда установленной жесткой связи: она заменяется легко модифицируемой слабой связанностью компонентов. Т.е. архитектура системы может видоизменяться и настраиваться, позволяя из набора готовых сервисов собирать ту конфигурацию, которая необходима на данный момент.
- Концепция SOA нашла свое воплощение в такой распространенной технологии, как **веб-сервисы.** Технология веб-сервисов предназначена для обеспечения доступа к функциям прикладных систем через сеть вне зависимости от используемых платформ. Она базируется на наборе открытых стандартов HTTP (транспорт), SOAP (обмен сообщениями), WSDL (описание интерфейсов), UDDI (публикация и поиск сервисов).

Механизм взаимодействия сервисов **SOA**



Взаимодействие сервисов осуществляется по принципу «публикация/регистрация – поиск/запрос к реестру – взаимодействие» и включает трех основных участников:

- **поставщика сервиса** – приложение, предоставляющее некоторый сервис;
- **потребителя сервиса** – приложение, которому необходима функциональность данного сервиса;
- **реестр сервисов**.

Поставщик регистрирует сервис, размещая информацию о нем в реестре сервисов.

Потребитель обращается к реестру с запросом об интересующем его сервисе и, получив необходимую информацию, устанавливает с ним соединение.

Среда интеграции Frameworx - *Integration Framework*

Совокупность принципов и моделей, определяющих посредством описания *стандартизованных программных интерфейсов* и технологически нейтральных интеграционных элементов – *бизнес-сервисов,* - подход к автоматизации бизнес-процессов и информационного обмена, осуществляемых приложениями и системами карты TAM [3].

Программный интерфейс, или просто интерфейс, описывает в виде протокола ожидаемое поведение взаимодействующих систем и задает формат и последовательность сообщений, которые системы посылают друг другу. Описание интерфейса **нейтрально** по отношению к используемым технологиям сетевого транспорта и участникам взаимодействия. Набор стандартных интерфейсов разрабатывается в рамках программы **TIP** (TM Forum Integration Program – программа TMF по обеспечению интеграции).

Бизнес-сервис является разновидностью сервиса SOA и описывает, каким образом приложения карты **TAM** взаимодействуют через программные интерфейсы для выполнения тех или иных задач в рамках бизнес-процессов **eTOM**. При этом данные, которыми обмениваются приложения, описываются в терминах информационных сущностей модели **SID** и их атрибутов. Таким образом, бизнес-сервис позволяет связать воедино модели **eTOM, SID и TAM** и трансформировать эту связку в конкретное решение.

Элементы спецификации бизнес-сервиса Framework

1. Поддерживаемые бизнес-процессы eTOM
2. Задействованные приложения TAM
3. Интерфейсы для обмена информацией между приложениями, **Open API**
4. Сущности SID и их атрибуты, которые будут использоваться при информационном взаимодействии
5. Описание поведения участников, хореография взаимодействия
6. Условия соглашения SLA и измеряемые бизнес-показатели
7. Декомпозиция бизнес-сервиса на более детализированные бизнес-сервисы

Открытые интерфейсы API /*Open Application Programming Interfaces*

Стандартизованные открытые интерфейсы программирования приложений / Open API от ТМ Форума важны как для бизнеса, так и для разработчиков.

Для первых, потому что предприятиям нужен общий язык и модель разработки для быстрой интеграции и взаимодействия с клиентами и партнерами. И для разработчиков - чтобы они могли, основываясь на существующем, создавать что-то новое, не тратя время, энергию и ресурсы, начиная с нуля.

В ТМ Forum на конец 2017г. есть набор из **более чем 50 Open API**, которые являются согласованными в отрасли, многоразовыми строительными блоками. Они могут иметь стратегическое влияние на модель разработки внутри организации. Повторно используя эти API, организация может инвестировать больше ресурсов в стратегические инновационные разработки новых услуг.

Вот несколько достижений Open API от TM Forum за 2017 год:

- Партнерство MEF и реализация архитектуры LSO с Open API-интерфейсом
- Поставка API активации функций и конфигурации ресурса, чтобы помочь стандартизировать работу виртуальной сети (VNF) и сетевого жизненного цикла
- Реализация архитектур на основе микросервисов с использованием Open API
- Расширение таблицы API с 31 до 52 API

Фрагмент TM Forum Open API Table

Название Open API	Release version
Account Management API	17.0.0*
Activation and Configuration API	15.5.1
Alarm Management API The Alarm Management API applies lessons that were learned in previous generations of similar APIs that were implemented in the Telecommunication industry, starting from ITU recommendations,, TM Forum OSS/J, MTOSI and TIP interfaces, NGMN alignment initiative between 3GPP and TM Forum interfaces, and the more recent ETSI work on requirements for NFV interfaces.	17.0.0*
Loyalty Management API	17.0.0*
NFV Entity Provisioning API REST API for NFV Entity Provisioning i.e. provisioning and lifecycle management of Network Services composed from Physical and Virtual Network Functions	15.5.0
Product Ordering API	16.5.1
Service Inventory Management API	16.5.1
SLA Management API	14.5.1
Trouble Ticket API **Customer Bill Management API**	14.5.1 17.5.0*

Библиотека стандартных интерфейсов TM Forum

1. OSS/J (OSS through Java). *Интерфейсы данной группы предназначены главным образом для обеспечения функций уровня услуг и бизнес-управления, но не управления сетью.*
2. MTOSI. *Интерфейсы для управления сетями и услугами.*
3. OMI (Operations Management Interfaces). Интерфейсы управления операционной (эксплуатационной) деятельностью. *Для управления сетями и услугами в конвергентной среде. Построены на основе лучших компонентов из интерфейсов TM Forum OSS/J и MTOSI, а также 3GPP. На текущий момент включает интерфейсы для управления решение проблем на уровне услуг, доступа к каталогу ресурсов, обеспечения безопасного доступа в систему, управления неисправностями, управления производительностью и др.*

4. SMI (SES Management Interface, ***реализации SOAP и RESTful)*** – интерфейсы управления SES (Software Enabled Service), обеспечивающие доступ к функционалу реализованных сервисов SOA. *SMI предоставляет возможности для SES устойчиво взаимодействовать с системами управления на любом этапе цепочки создания стоимости в* ***многопартнерской*** *среде.*

Немного о REST API и SOAP API

Стоит отметить, что на сегодняшний день есть два основных подхода к построению программного интерфейса веб-приложений: REST (RESTful) API и SOAP API:

- ***REST*** (***Representational State Transfer*** – «передача состояния представления») обеспечивает общение между клиентом (как правило, это браузер) и сервером с помощью обычных HTTP-запросов (GET, POST, PUT, DELETE и т. д), передавая информацию от клиента в параметрах самих запросов, информацию от сервера – в теле ответа (который может быть, например, **JSON**-объектом или **XML**-документом). **REST является архитектурным стилем**, а не стандартом.
- ***SOAP*** (***Simple Object Access Protocol*** – простой протокол доступа к объектам, вплоть до спецификации 1.2) характеризуется использованием HTTP(S)-протокола лишь как транспорта (чаще всего, методом POST). Все детали сообщений (в обе стороны – от клиента к серверу и обратно) передаются в стандартизованном XML-документе. SOAP может работать и с другими протоколами прикладного уровня (SMTP, FTP), но чаще всего он применяется поверх HTTP(S). **SOAP является протоколом** и имеет спецификацию.

В широком смысле компоненты в REST взаимодействуют наподобие взаимодействия клиентов и серверов во Всемирной паутине. REST является альтернативой RPC.

Если уподобить HTTP-запрос бумажному носителю, то можно сказать, что REST API в большинстве случаев передает простые записки, а время от времени – письмо в конверте (возможно, написав при этом часть послания и на самом конверте). В свою очередь, SOAP API передает все инструкции в подробном письме стандартного вида, используя конверт (единичный HTTP-запрос) лишь как средство доставки.

Тенденция ТМ Форума заключается в интеграции с использованием **API на основе REST**.

Представление API с помощью **JSON**

JSON (*JavaScript Object Notation*) — текстовый **формат** обмена данными, основанный на JavaScript. Как и многие другие текстовые форматы, JSON легко читается людьми. Несмотря на происхождение от JavaScript, формат считается полностью независимым от языка реализации и может использоваться практически с любым языком программирования. Использует соглашения, знакомые программистам C-подобных языков, таких как C, C++, C#, Java, JavaScript, Perl, Python и многих других. Эти свойства делают JSON идеальным языком обмена данными.

Нотация JSON выглядит так:

Объект - неупорядоченный набор пар ключ/значение. Объект начинается с **{** (открывающей фигурной скобки) и заканчивается**}** (закрывающей фигурной скобкой). Каждое имя сопровождается **:** (двоеточием), пары ключ/значение разделяются **,** (запятой).

В частности, первые два свойства объекта ниже – некорректны:

```
{
    name: "Вася",          // ошибка: ключ name без кавычек!
    "surname": 'Петров', // ошибка: одинарные кавычки у значения 'Петров'!
    "age": 35,             // .. а тут всё в порядке.
    "isAdmin": false       // и тут тоже всё ок
}
```

В формате JSON не поддерживаются комментарии. Он предназначен только для передачи данных.

JSON

Следующий пример показывает JSON-представление объекта, описывающего человека.

В объекте есть ***строковые*** поля имени и фамилии, объект, описывающий адрес, и массив, содержащий список телефонов. Как видно из примера, ***значение*** может представлять собой вложенную структуру.

```
{
    "firstName": "Иван",
    "lastName": "Иванов",
    "address": {
        "streetAddress": "Московское ш., 101, кв.101",
        "city": "Ленинград",
        "postalCode": 101101
    },
   "phoneNumbers": [
        "812 123-1234",
        "916 123-4567"
   ]
}
```

Язык XML

XML */e**X**tensible **M**arkup **L**anguage* - расширяемый язык разметки, предназначен для структурирования, хранения и передачи данных, а также для создания на его основе производных специализированных языков.

Разработан Консорциумом Всемирной паутины (W3C/*World Wide Web Consortium*) и опубликован в качестве стандарта в 1998г.

Спецификация XML описывает XML-документы и частично описывает поведение XML-процессоров (программ, читающих XML-документы и обеспечивающих доступ к их содержимому).

XML разрабатывался как язык с простым формальным синтаксисом, удобный для создания и обработки документов программами и одновременно удобный для чтения и создания документов человеком, с нацеленностью на использование в Интернете.

Язык называется расширяемым, поскольку он не фиксирует разметку, используемую в документах: разработчик волен создать разметку в соответствии с потребностями к конкретной области, будучи ограниченным лишь синтаксическими правилами языка. Расширение XML — это конкретная грамматика, созданная на базе XML и представленная словарём тегов и их атрибутов, а также набором правил, определяющих какие атрибуты и элементы могут входить в состав других элементов.

Структура XML-документа

XML-документ состоит из:

- Деклараций
- Элементов
- Комментариев
- Специальных символов
- Директив

XML: элементы и атрибуты

В XML данные описывают с помощью задаваемых пользователем ***тегов*** – последовательностей символов, ассоциирующихся с фрагментами данных и служащих основой для их идентификации, классификации и обработки.

XML — это ***теговый*** язык разметки документов. Иными словами, любой документ на языке XML представляет собой набор *элементов*, причем начало и конец каждого элемента обозначается специальными пометками, называемыми ***тегами***.

- ***Элемент*** состоит из трех частей: начального тега, содержимого и конечного тега. *Тег* — это текст, заключенный в угловые скобки "**<**" и "**>**". Конечный тег имеет то же имя, что начальный тег, но начинается с косой черты "**/**". Пример XML-элемента:

<author>Сергей Довлатов</author>

- Имена элементов зависят от регистра, т. е. <author>, <Author> и <AUTHOR> — это имена различных элементов. Наличие закрывающего тега всегда обязательно. Если тег является пустым, т. е. не имеет содержимого и закрывающего тега, то он имеет специальную форму: <элемент/>

XML: элементы и атрибуты *(продолж)*

Любой элемент может иметь ***атрибуты***, содержащие дополнительную информацию об элементе. *Атрибуты* всегда включаются *в начальный тег элемента* и имеют вид:

имя_атрибута="значение_атрибута"

Атрибут обязан иметь значение, которое всегда должно быть заключено в одинарные или двойные кавычки. Пример элемента, имеющего атрибут:

<author country="USA">Сергей Довлатов</author>

Элементы должны либо следовать друг за другом, либо быть вложены один в другой:

```
<books>
  <book isbn="5887821192">
  <title>Часть речи</title>
  <author>Бродский, Иосиф</author>
  <present/>
 </book>
  <book isbn="0345374827">
  <title>Марш одиноких</title>
  <author>Довлатов, Сергей</author>
  <present/>
 </book>
</books>
```

XML: пролог и директивы

Любой XML-документ состоит из *пролога и корневого элемента*, например:

```
<?xml version="1.0"?>
<books>
  <book isbn="0345374827">
    <title>Марш одиноких</title>
    <author>Довлатов, Сергей</author>
    <present/>
  </book>
</books>
```

В этом примере пролог сводится к единственной директиве (первая строка документа), указывающей версию XML. За ней следует XML-элемент с уникальным именем, который содержит в себе все остальные элементы и называется корневым.

Директива (англ. processing instruction) — это выражение, заключенное в специальные теги "<?" и "?>", которое содержит указания программе, обрабатывающей XML-документ.

XML: Комментарии

XML-документы могут содержать комментарии, которые игнорируются приложением, обрабатывающим документ. Комментарии строятся по тем же правилам, что и в HTML:

- начинайте комментарий с символов "<!--",
- завершайте комментарий символами "-->",
- не используйте внутри комментария символов "--".

Пример комментариев:

```
<!-- это комментарий -->
<!-- а вот еще комментарий,
     занимающий более одной строки -->
```

Достоинства XML

- язык разметки, позволяющий отобразить двоичные данные в текст, читаемый человеком и анализируемый компьютером;
- поддерживает Юникод;
- в формате XML могут быть описаны такие структуры данных как записи, списки и деревья;
- это самодокументируемый формат, который описывает структуру и имена полей так же как и значения полей;
- имеет строго определённый синтаксис и требования к анализу, что позволяет ему оставаться простым, эффективным и непротиворечивым. Одновременно с этим, разные разработчики не ограничены в выборе экспрессивных методов (например, можно моделировать данные, помещая значения в параметры тегов или в тело тегов, можно использовать различные языки и нотации для именования тегов и т. д.);
- формат, основанный на международных стандартах;
- Иерархическая структура XML подходит для описания практически любых типов документов, кроме аудио и видео мультимедийных потоков, растровых изображений, сетевых структур данных и двоичных данных;
- представляет собой простой текст, свободный от лицензирования и каких-либо ограничений;
- не зависит от платформы;
- является подмножеством SGML (который используется с 1986 года). Уже накоплен большой опыт работы с языком и созданы специализированные приложения;
- не накладывает требований на расположение символов в строке;
- В отличие от бинарных форматов, XML содержит метаданные об именах, типах и классах описываемых объектов, по которым приложение может обработать документ неизвестной структуры (например, для динамического построения интерфейсов);
- имеет реализации парсеров для всех современных языков программирования;
- поддерживается на низком аппаратном, микропрограммном и программном уровнях в современных аппаратных решениях.

Недостатки XML

- Синтаксис XML избыточен.
- Размер XML-документа существенно больше бинарного представления тех же данных. В грубых оценках величину этого фактора принимают за 1 порядок (в 10 раз).
- Размер XML-документа существенно больше, чем документа в альтернативных текстовых форматах передачи данных и особенно в форматах данных, оптимизированных для конкретного случая использования.
- Избыточность XML может повлиять на эффективность приложения. Возрастает стоимость хранения, обработки и передачи данных.
- XML содержит метаданные (об именах полей, классов, вложенности структур), и одновременно XML позиционируется как язык взаимодействия открытых систем. При передаче между системами большого количества объектов одного типа (одной структуры), передавать метаданные повторно нет смысла, хотя они содержатся в каждом экземпляре XML описания.
- Для большого количества задач не нужна вся мощь синтаксиса XML и можно использовать значительно более простые и производительные решения.
- Неоднозначность моделирования.
- Нет общепринятой методологии для моделирования данных в XML, в то время как для реляционной модели и объектно-ориентированной такие средства разработаны и базируются на реляционной алгебре, системном подходе и системном анализе.
- В природе есть множество объектов и явлений, для описания которых разные структуры данных (сетевая, реляционная, иерархическая) являются естественными, и отображение объекта в неестественную для него модель является болезненным для его сути. В случае с реляционной и иерархической моделями определены процедуры декомпозиции, обеспечивающие относительную однозначность, чего нельзя сказать о сетевой модели.
- В результате большой гибкости языка и отсутствия строгих ограничений, одна и та же структура может быть представлена множеством способов (различными разработчиками), например, значение может быть записано как атрибут тега или как тело тега и т. д. Например: или или <a>1<c>1</c> или <a><c value="1"/> или <a><fields b="1" c="1"/> и т. д.
- Поддержка многих языков в именовании тегов дает возможность назвать, например вес русским словом, в таком случае компьютер никак не сможет установить соответствия этого поля с полем weight в англоязычной версии программы и с полями в версиях модели объекта на множестве других языков.
- XML не содержит встроенной в язык поддержки типов данных. В нём нет строгой типизации, то есть понятий «целых чисел», «строк», «дат», «булевых значений» и т. д.
- Иерархическая модель данных, предлагаемая XML, ограничена по сравнению с реляционной моделью и объектно-ориентированными графами и сетевой моделью данных.
- Выражение неиерархических данных требует дополнительных усилий
- Кристофер Дейт, специалист в области реляционных баз данных, автор классического учебника «An Introduction to Database Systems», отмечал, что «…XML является попыткой заново изобрести иерархические базы данных…»[1](в 1980-е года иерархические базы данных были вытеснены реляционными базами данных).
- Пространства имён XML сложно использовать и их сложно реализовывать в XML-парсерах.
- Существуют другие, обладающие сходными с XML возможностями, текстовые форматы данных, которые обладают более высоким удобством чтения человеком/

Общая схема технологически нейтральной архитектуры TNA NGOSS

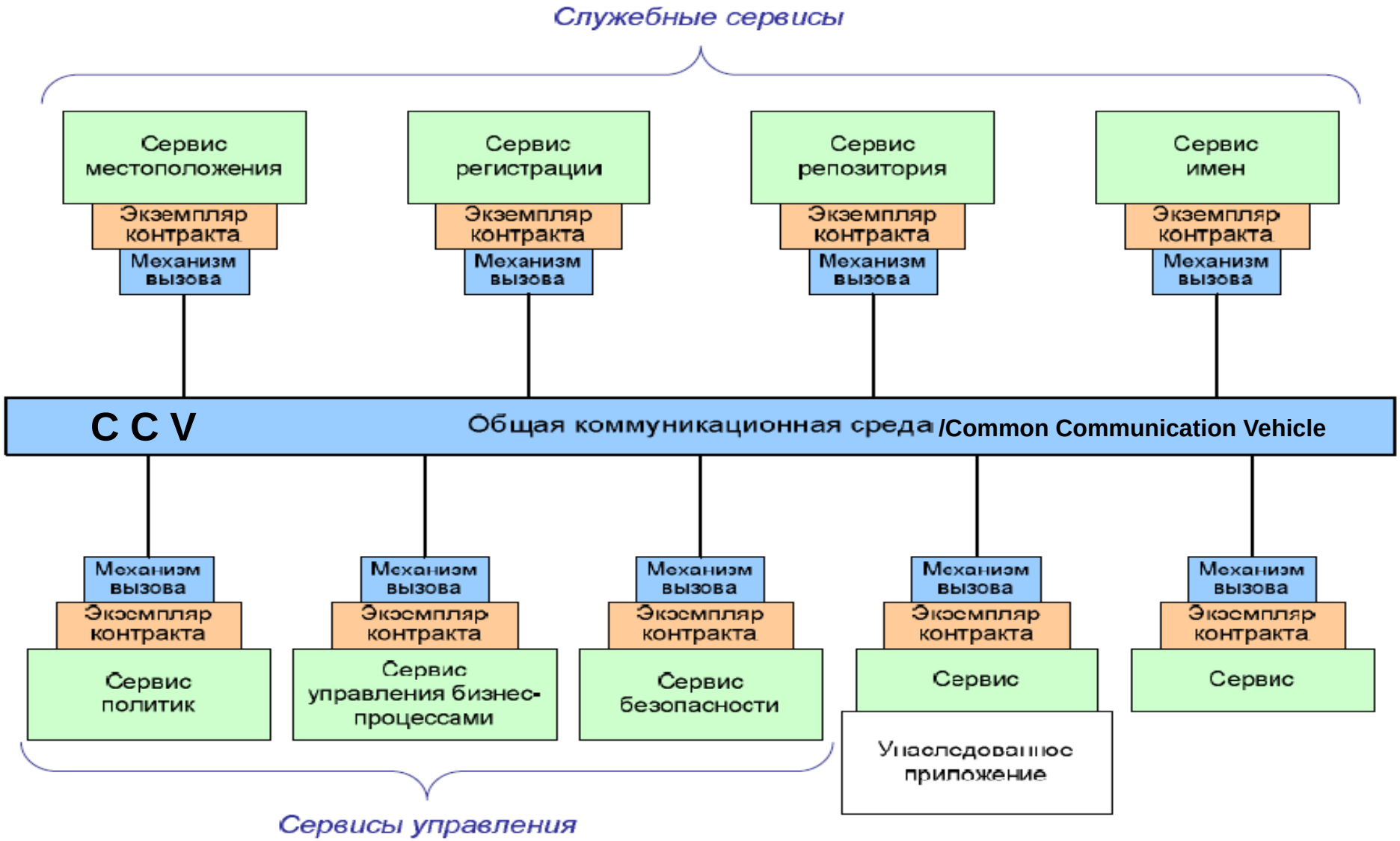


Диаграмма взаимодействия в рамках TNA

Резюме

ТЕХНОЛОГИЧЕСКИ НЕЙТРАЛЬНАЯ АРХИТЕКТУРА

не случайно называется технологически нейтральной: она не задает набор технологий для реализации решения, а лишь определяет общие принципы, которым должна соответствовать построенная система OSS/BSS.

При проектировании, система OSS/BSS описывается в спецификациях, которые не привязаны к какой-либо технологии реализации. После чего в отдельных документах излагаются методы отображения технологически нейтральной архитектуры на конкретную технологию для разработки и развертывания решения.

На сегодняшний день Среда интеграции является наименее завершенным и стабильным стандартом из четырех интегрированных сред Frameworx. Продолжается развитие концепции бизнес-сервисов…

Контрольные вопросы

1. Что такое архитектура TNA и какова ее роль в концепции NGOSS?
2. Какие требования предъявляются к архитектуре NGOSS?
3. Что такое компонент в архитектуре NGOSS?
4. Что такое сервис в архитектуре NGOSS?
5. Какие типы компонентов предусмотрены в архитектуре NGOSS?
6. Какие типы сервисов предусмотрены в архитектуре NGOSS?
7. Какие служебные сервисы вы знаете?
8. Дайте определение Среды интеграции Frameworx / *Integration Framework*
9. Перечислите элементы спецификации бизнес-сервиса Frameworx.
10. Стандартизованные открытые интерфейсы программирования приложений / Open API от TM Форума, таблица и способ описания
11. Структура XML-документа, элементы и атрибуты.
12. Что такое общая коммуникационная среда CCV? Что такое механизм вызова?

Источники

1. Самуйлов К. Е., Серебренникова Н.В., Чукарин А.В., Яркина Н.В**.** Системы следующего поколения для поддержки операционной деятельности инфокоммуникационной компании: Учеб. пособие. – М.: РУДН, 2008. – 123 с.
2. *А.А. Атцик, А.Б. Гольдштейн, М.А. Феноменов.* ЭКСПЛУАТАЦИОННОЕ УПРАВЛЕНИЕ ИНФОКОММУНИКАЦИЯМИ: учебное пособие для практических занятий и лабораторных работ по дисциплинам «Системы управления инфокоммуникациями» и «Бизнес-процессы Операторов связи»: учебное пособие / ГОУВПО СПбГУТ. СПб, 2013 – 80 с.
3. Бизнес-процессы и информационные технологии в управлении современной инфокоммуникационной компанией / А.В. Чукарин, К.Е. Самуйлов, Н.В. Яркина. - М. : Альпина Паблишер, 2016. - 512 с.