схема хранилища медицинских данных

209e9dcaa37631c225f1dde178321fa49bd6

Обзор методов проектирования хранилища данных для области здравоохранения

Примерно 2015 год

Семь требований были определены из клинических данных характеристик:

целостность данных,

звуковая временная схема,

выразительность запроса,

гетерогенная интеграция данных,

интеграция знаний и эволюции источников,

прослеживаемость и

управляемая автоматизация

Выделены характеристики хранилищ:

1. Проектный подход (D. App.). - подход, основанный на источнике (также называемый управляемым данными, управляемым предложением, снизу вверх) - начинается с источников данных для получения схемы DW, - подход, основанный на требованиях (также называемый управляемым спросом, ориентированным на цель, сверху вниз), начинается с требований пользователя, - гибридный подход (также называемый смешанным подходом) объединяет оба подхода. - основанный на знаниях подход, ориентированный на знание предметной области для определения соответствующих концепций для структурирования - и разработать схему DW.

2. Процесс (в смысле процесс создания хранилища). - полностью автоматизирован, в основном автоматизирован, частично автоматизирован или не полностью автоматизирован

3. Представление знаний (К. Реп.). Этот критерий определяет модель, используемую для представления знаний о предметной области. Также называемая моделью смысла, она представляет собой надежное представление сущностей предметной области и отношений между ними.

4. Требование представления (Р. Реп.). Этот критерий определяет модель, используемую для представления требований конечного пользователя.

5. Источник представления (С. Реп.). Этот критерий определяет модель, используемую для представления источника (не путать с типом источника).

6. Исходный анализ (С. Ана.). Исходный анализ может быть выполнен по структуре (она же метаданные, S), данным (D) или обоим (D-S).

7. Несколько источников (Multi. S.). Схема DW может быть получена из нескольких источников. Это предполагает, что метод учитывает интеграцию данных и структуру источников.

8. Алгоритмы определения (Алго.). Авторы опубликовали все необходимые алгоритмы таким образом, чтобы они могли быть реализованы независимо.

9. Концептуальная модель данных (МЧР). МЧР направлен на выявление и описание концепций в том виде, как они понимаются конечными пользователями. Типичные значения: - Модель сущности-отношения (ERM), - Онтологическая модель (ОМ), - модель факт-фактора (DFM) [20] и т. Д.

10. Логическая модель данных (LDM). Типичные значения:

- Реляционная модель (РДТ),

- модель Звезда (Star),

- модель помеченного графа (TGM) и т. д.

11. Физическая модель данных (PDM). Типичные значения:

- SQL (точнее, СУБД, реализующая язык SQL, такой как Oracle или PostgreSQL),

- МОЛАП,

- РОЛАП,

- OBDW (как Onto DB [14]) и т. Д.

12. Временная модель данных (TDM). Этот критерий определяет временную модель данных:

- BCDM [79 Снодграсс, Р.Т .: Разработка ориентированных на время приложений баз данных в SQL. Morgan Kaufmann Publishers, Сан-Франциско, Калифорния (2000)],

- TRM [13 Date, C.J., Darwen, H., Lorentzos, N.A .: Время и реляционная теория: временные базы данных в реляционной модели и SQL. Morgan Kaufmann, Waltham, MA (2014)],

- А. В. [30 Джонстон Т., Вейс Р.: Управление временем в реляционных базах данных: как проектировать, обновлять и запрашивать временные данные. Morgan Kaufmann / Elsevier, Амстердам; Бостон (2010)] и др.

13. Тип DW (производится процессом DWD):

- реляционный [9],

- размерный [37],

- Якорь [72],

- хранилище данных [25] и т. Д.

Еще один ключевой вывод нашего исследования заключается в том, что использование предметных знаний необходимо для улучшения выбора и интерпретации соответствующих данных. Это также способствует автономии пользователей, так как они могут использовать данные непосредственно через соответствующее представление знаний, а не в зависимости от требований. Как следствие, методы должны стремиться объединить исходные знания и знания предметной области, но оптимальный метод представления знаний на данный момент остается труднодостижимым.

Т.е. ассоциации нужны, но их нет

Пояснения:

Двухвременная концептуальная модель данных (BCDM) эффективно выражает двухвременный механизм, который позволяет получить двумерное выражение о виртуальном времени и времени транзакции. Основываясь на стандартизации временной метки в BCDM, документ отображает временную информацию на основе модели моментальных снимков RDBMS (Relational Database Management System), затем анализирует, какая нормальная формула может быть выражена в BCDM, затем декомпозирует временную модель в 3NF на основе модели моментальных снимков RDBMS. , https://www2.cs.arizona.edu/~rts/pubs/ISDec94.pdf

В графических представлениях битемпорального пространства мы выбираем ось X в качестве измерения времени транзакции, а ось Y в качестве измерения действительного времени.

Факты пишутся в состояниих, например:

факт (Jake, Ship) удаляется из текущего состояния, а факт (Jake, Load) вставляется.

https://habr.com/ru/post/348188/

**Введение в Data Vault**

Ищете более функциональную альтернативу схеме «звезды» и Третьей Нормальной Форме?

У Вас уже есть хранилище данных, но его тяжело дорабатывать?

Нужна хорошая поддержка историчности, а текущая архитектура для этого не подходит?

Возникают проблемы при сборе данных из нескольких источников?

[**Data Vault**](https://en.wikipedia.org/wiki/Data_vault_modeling) — гибридный подход, объединивший достоинства знакомой многим [схемы «звезды»](https://ru.wikipedia.org/wiki/%D0%A1%D1%85%D0%B5%D0%BC%D0%B0_%D0%B7%D0%B2%D0%B5%D0%B7%D0%B4%D1%8B) и 3-ей нормальной формы. Идея - создание прослойки перед витринами данных.

Data Vault состоит из трех основных компонентов:

**Хаб (Hub)** - основное представление сущности (Клиент, Продукт, Заказ) с позиции бизнеса. Также содержит мета-поля load timestamp и record source, в которых хранятся время первоначальной загрузки сущности в хранилище и ее источник (название системы, базы или файла, откуда данные были загружены)

**Ссылка (Link)** - связывают несколько хабов связью многие-ко-многим.

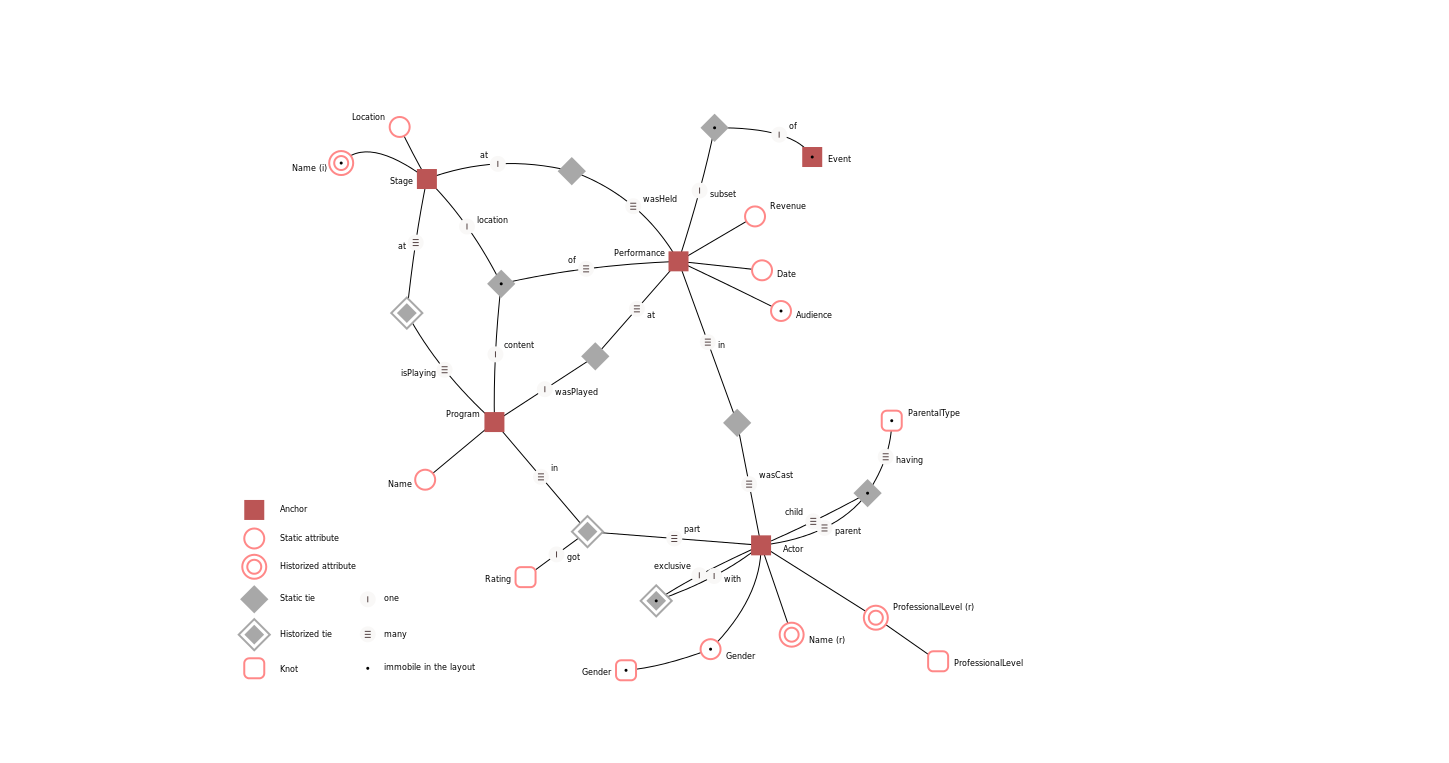
**Сателлит (Satellite) -** Все описательные атрибуты Хаба или Ссылки (контекст) помещаются в таблицы-Сателлиты. Помимо контекста Сателлит содержит стандартный набор метаданных (*load timestamp* и *record source*) и **один и только один** ключ «родителя». В Сателлитах можно без проблем хранить историю изменения контекста, каждый раз добавляя новую запись при обновлении контекста в системе-источнике.

<https://ru.wikipedia.org/wikiЯкорная> модель

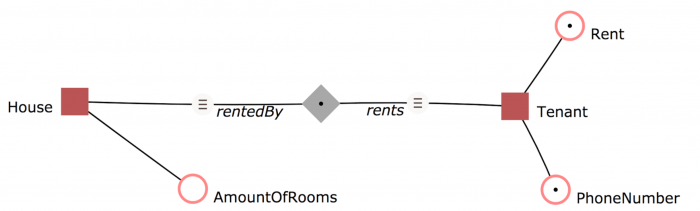
**Якорное моделирование** - это технология моделирования гибкой базы данных, подходящая для информации, которая со временем изменяется как по структуре, так и по содержанию. Он обеспечивает графическое обозначение, используемое для концептуального моделирования, аналогичное [моделированию отношений сущностей](https://ru.wikipedia.org/wiki/Entity-Relationship), с расширениями для работы с временными данными. В методике моделирования используются четыре модели моделирования: якорь, атрибут, связь и узел, каждый из которых отражает различные аспекты моделируемого домена. Полученные модели могут быть переведены в физические проекты баз данных с использованием формализованных правил. Когда такой перевод сделан, таблицы в реляционной базе данных будут в основном в [шестой нормальной форме](https://ru.wikipedia.org/wiki/%D0%A8%D0%B5%D1%81%D1%82%D0%B0%D1%8F_%D0%BD%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%84%D0%BE%D1%80%D0%BC%D0%B0).

Якорное моделирование имеет четыре основные концепции моделирования: **якоря, атрибуты, связи и узлы**. Якоря используются для моделирования сущностей и событий, атрибуты используются для моделирования свойств якорей, связи моделируют отношения между якорями, а узлы используются для моделирования общих свойств, таких как состояния. Атрибуты и связи могут быть истолкованы, когда изменения в информации, которую они моделируют, необходимо сохранить.

Примерная модель, показывающая различные графические символы для всех концепций, приведена ниже. Символы напоминают те, что используются в модели [моделированию отношений сущностей](https://ru.wikipedia.org/wiki/Entity-Relationship), с несколькими расширениями. Двойной контур по атрибуту или привязке указывает, что сохраняется история изменений, и также доступен символ узла (обведенный квадрат с закругленными краями).



<https://blog.codecentric.de/en/2017/07/agile-database-design-using-anchor-modeling/>



<https://habr.com/ru/company/avito/blog/322510/>

**Vertica+Anchor Modeling = запусти рост своей грибницы**

3 марта 2017 в 14:02

**Anchor** — это существительное, объект реального мира. Товар, пользователь, платеж. Соответственно, каждому существительному — своя таблица.

**Attribute** — это таблица для хранения свойства, атрибута объекта. Названия товара, логина и даты рождения пользователя, суммы платежа. Одно свойство у объекта — одна Attribute-таблица. Десять свойств у объекта (имя, фамилия, дата рождения, пол, адрес регистрации, ...) — десять Attribute-таблиц.

**Tie** — это таблица для хранения связей между объектами. Например, таблица для хранения факта наличия у покупателя гражданства в определенной стране. Соответственно, таблица должна содержать суррогатный ключ левого объекта (customer\_id), правого объекта (country\_id) и, по необходимости, даты историчности и технических полей.

Важный нюанс с точки зрения моделирований — Anchor Modeling сильно отличается от Data Vault тем, что в Data Vault можно вешать данные (сателлиты) на связь (link), а в Anchor Modeling данные (Attribute) можно повесить только на Anchor, на Tie нельзя (важно — НЕЛЬЗЯ). Это на первый взгляд избыточное ограничение позволяет более точно моделировать реальный физический мир. Например, традиционная связь со свойствами в Data Vault — это факт продажи товара клиенту, свойством которого является сумма продажи. Anchor Modeling заставляет немного подумать и понять, что факт продажи товара клиента — это не элемент реального мира, а абстракция. Элементом реального мира является чек (бумажка) с номером, датой и т.п. Соответственно, в Anchor modeling описанный пример описывается тремя Anchor — Покупатель, Чек, Товар, и двумя Tie: Покупатель-Чек и Чек-Товар. 

https://www.healthcatalyst.com/knowledge-center/insights/category/data-warehouse/

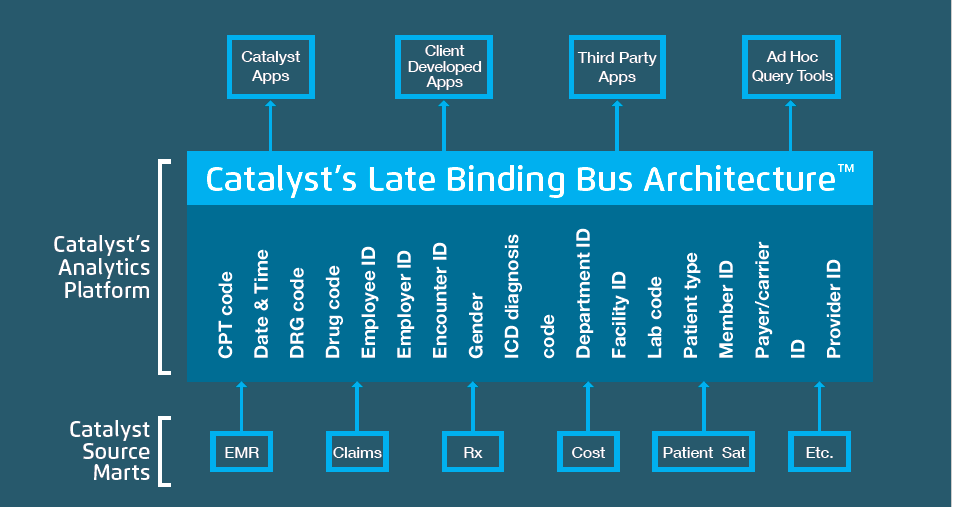
Подходы раннего или позднего связывания к хранилищу медицинских данных: что для вас лучше? Майк Дойл Опубликовано в Enterprise Data Warehouse / Операционная система данных. … Есть некоторые характеристики, которые делают ранние подходы менее привлекательными в качестве модели данных для здравоохранения. Чтобы начать использовать аналитику в здравоохранении, требуется много времени из-за усилий и усилий, связанных с привязкой данных и согласованием бизнес-правил. Кроме того, раннее связывание имеет ограниченную способность адаптироваться к новым бизнес-правилам и вариациям, которые распространены в здравоохранении.



В хранилище данных есть шесть точек, в которых данные могут быть связаны с правилами и словарями. Как потоки данных Слева направо на диаграмме ниже пункты 1 и 2 подходят для привязки к правилам и словарям, которые показывают низкая волатильность; то есть те правила и словари, которые изменяются нечасто, такие как идентификаторы пациентов и поставщиков идентификаторы. Позднее связывание - в пунктах 5 и 6 - подходит для правил и словарного запаса, которые могут измениться в на регулярной основе, или для которых не существует стандартного правила или словарного запаса. Например, привязка в слое визуализации подходит для анализа сценария «что если», который связан с моделированием различных моделей возмещения или определением болезненные состояния. Как только эта предварительная фаза «что если» завершена, новые модели и определения могут быть заблокированы и связаны в пунктах 3, 4 или 5. Рекомендуется хранить записи привязок в хранилище данных. Эта запись позволит аналитикам быстро запускать модели на основе правил и словарного запаса (например, от МКБ-9 до МКБ-10), которые меняются со временем, что полезно для прогнозирования и прогнозная аналитика.

МКБ = Международная статистическая классификация болезней и проблем со здоровьем Пять основных подходов • Корпоративная информационная модель Раннее связывание - Защита Биллом Инмом и Клаудией Имхофф • I2B2 - Защищено академической медициной • Звездная схема ↓ - Защита Ральф Кимбалл • Архитектура шины позднего связывания - Защищено Дейлом Сандерсом • Ассоциация файловой структуры - популяризируется мэйнфреймами IBM в 1960-х - появляется в Hadoop и NoSQL - Нет традиционной реляционной модели данных Позднее связывание

Все объединяется через шину данных: Основные элементы данных показаны выше, иллюстрируя, как эти элементы данных используют модели данных исходных систем для работы в качестве шины данных для платформы Catalyst Late-Binding ™ Warehouse. Этот подход позволяет выполнять запросы к разнородному содержимому исходной системы в хранилище данных точно так же, как теоретические преимущества модели данных предприятия, но не требует разработки и соответствия модели данных предприятия.

****

https://www.healthcatalyst.com/whitepaper/3-approaches-healthcare-data-warehousing Сравнение трех основных подходов к хранению данных в здравоохранении: глубокий обзор погружений (Белая книга) Изменчивые данные, которые должны быть связаны поздно: Length Расчет продолжительности пребывания (LOS) − Присвоение поставщика первичной медицинской помощи конкретному пациенту с хроническим заболеванием − Расчет распределения доходов (или расходов) и прогнозы для отделения или врача − Определения данных общих состояний болезни для реестров пациентов Criteria Определение критериев исключения пациентов для лечения заболеваний и / или населения. Rules Определение правил приема, выписки и перевода пациентов Энергонезависимые данные, которые могут быть связаны рано: Identif идентификатор объекта Identif идентификатор провайдера Identif идентификатор пациента − Пол − Дата − время прибытия

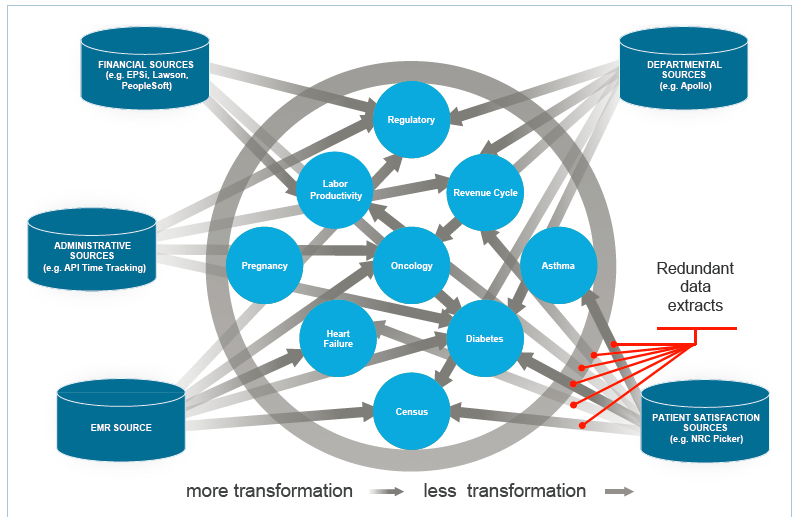
В настоящее время существует три основных типа хранилищ данных, из которых системы здравоохранения могут хранить и извлекать свои данные. Модели хранилища данных следующие:

1. Модель предприятия (= Корпоративная информационная модель, поддерживаемая Биллом Инмом) - монолитная система, вообще не подходит.

2. независимая модель витрины данных,

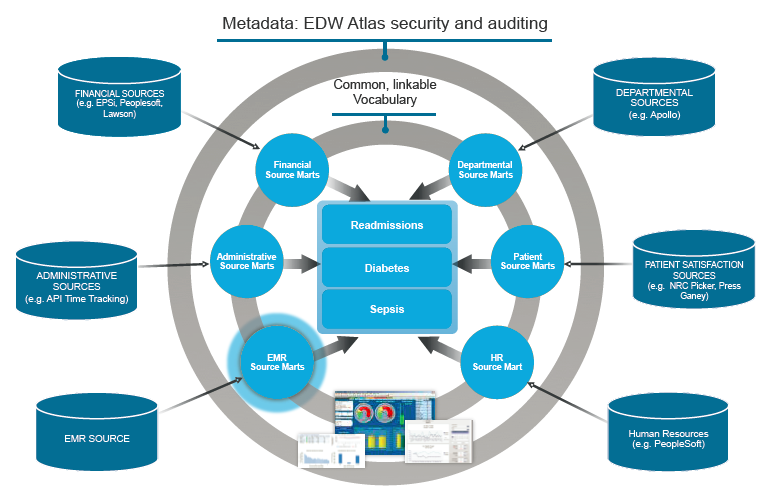
3. модель архитектуры позднего связывания.

2. Независимая модель витрины данных - организация начинает с малого, создавая отдельные витрины данных как места для хранения конкретной информации для каждого отделения больницы. С помощью независимой модели витрины данных организация создает аналитическую витрину данных для конкретного отдела - например, сердечной недостаточности - собирает необходимые данные непосредственно из исходных систем и отображает их в различных областях.



(+) - дешево (-) - Недостаточно детальных данных (в одной базе - только фамилии пациента, в других - все встречи в нем) (-) - не хватает эффективности. Представьте себе встраивание нового канала из электронной медицинской карты (EHR) в каждый построенный витрины данных: сердечная недостаточность, беременность, астма, диабет, онкология ... и этот список можно продолжать и продолжать. (-) - Требуется раннее связывание данных. в каждом независимом витрине данных данные отображаются в предопределенную модель данных (соответствие и нормализация). Например, процесс сопоставления и согласования данных с этими моделями раннего связывания в хранилище данных о доставке в здравоохранение обычно занимает от 18 до 24 месяцев или дольше. Когда новые источники данных добавляются в хранилище данных - как это происходит при слияниях, приобретениях и партнерских отношениях ACO - этот длительный период времени снова и снова повторяется. При любых изменениях базы нужно переделывать, т.е. процесс повторять.

3. Архитектура Late-Binding ™ ускоряет время до получения стоимости, требуя меньшего преобразования при перемещении данных из исходных систем в EDW, а комплексная интеграция осуществляется выборочно на уровне предметной области (центральные прямоугольники).



архитектура хранилища данных с поздним связыванием

2500067

Разработка архитектуры хранилища клинических данных для поддержки инициатив по улучшению качества

Пример реализации позднего связывания

Потому что у нас не было ресурсов для полной интеграции данных между системами и

Данные EHR представляли новый тип данных, мы сосредоточились на процессе извлечения, чтобы упростить доступ к данным

и смоделировали или преобразовали данные в зависимости от необходимости (например, позднее связывание).



<https://www.tieto.com/ru/what-we-do/it-services/esb/>

**Интеграционная шина предприятия** (ESB). ESB выступает в роли программного обеспечения, которое обеспечивает взаимосвязь между различными информационными системами предприятия по различным протоколам. Каждое приложение подключается только к ESB. Шина выполняет маршрутизацию сообщений между системами, гарантирует доставку сообщения до системы (даже если информационная система не работала в момент формирования сообщения). При необходимости шина обогащает сообщение дополнительной информацией, отсутствующей в исходном сообщении.

<https://www.datareon.ru/products/esb-servisnaya-shina-dannykh/>

<https://wiseadvice-it.ru/o-kompanii/blog/articles/integracii-s-1s-na-osnove-shiny-esb/>

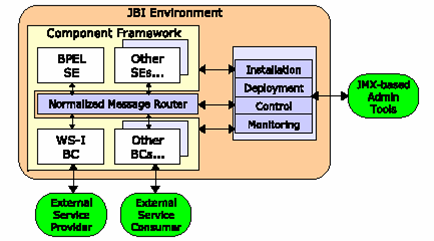
Интеграционная шина ESB (Enterprise Service Bus) – это комплекс программных продуктов для интеграции корпоративных приложений на основе событийной модели, в которых основной платформой является 1С, хотя и не обязательно.

<https://infostart.ru/public/21506/>

**Интеграция 1С с сервисной шиной OpenESB**

Есть код

Собственно для организации шины ESB вполне можно использовать тот же подход, что и при использовании обычной шины сообщений, таким образом двумя главными элементами будут:   
1. Некая промежуточная шина, которая позволяет принимать и передавать сообщения в согласованном формате.  
2. Множества подключаемых модулей, которые принимают и передают сообщения - этими модулями могут быть коннекторы к внешнем системам или движок сервисов, получивший SOAP сообщение и переправляющий его движку BPEL (Busines Process Execution Language), который тоже оформлен в виде компонента.   
Собственно нижеприведенная схема организации JBI ( [http://en.wikipedia.org/wiki/Java\_Business\_Integration](https://infostart.ru/redirect.php?url=aHR0cDovL2VuLndpa2lwZWRpYS5vcmcvd2lraS9KYXZhX0J1c2luZXNzX0ludGVncmF0aW9u) ) это иллюстрирует:



Таким образом получаем модульную среду, позволяющую объединить самые разные компоненты и системы. Однако не стоит идеализировать ESB, как и в любом другом ПО там могут быть свои собственные ошибки, например, в ранних версиях OpenESB где-то в глубине вылетал NullPointerException при попытке вызвать сервис на Mono и усе, кина не будет.  
   
Другим важным вопросом является движок веб-сервисов, его задачи когда-то были довольно простыми: принять SOAP, приземлить вызов, забрать результат и отдать SOAP. Собственно встроенный движок 1С и находится на таком уровне, но сейчас от движка требуется, помимо этого, реализация множества стандартов, определяющих то или иное поведение, например, WS-ReliableMessaging (для гарантированной доставки сообщений), WS-Security (для шифрования, подписи и аутентификации - заметьте, имено это стандарт, а не поделка 1С с HTTP аутентификацией, потому что веб-сервисы в общем случе могут быть и не "веб"), WS-[Atomic]Transaction (для организации транзакционного поведения). Очевидно, что чем более развитым и надежным является движок, тем лучше. Эти соображения, а также желания сделать универсальный механизм, который можно легко прикрутить к любой конфигурации, сразу же ставят крест на встроенном движке 1С. Собственно выбор не богат, но это не значит, что плох - WCF под .NET.

 Основная идея уже всплывала в комменатариях к предыдущей статье: сделать универсальный windows сервис, который будет приземлять вызовы на 1С через COM.  Собственно схему работы всей конструкции можно выразить одной строкой:  
  
**? <-n SOAP n-> OpenESB <-1 SOAP 1-> OneCService(WCF) <-1 COM n-> 1С**

где ? - произвольные внешние системы, в примере их роль играют тесты

     OneCService - собственно сервис, разработанный в рамках данного примера

     в угловых скобках указаны способы взаимодействия и отношения  
  
Такая схема позволит не вносить серьезных изменений в конфигурацию, а кроме того, получть универсальный механизм, в котором один промежуточный сервис позволяет взаимодействовать с разными базами 1С. В данном пример будет реализовано только взаимодействие с файловыми версиями, для взаимодействия с серверными версиями надо будет просто изменить механизм формирования строки соедиения для V8.Application и добавить дополнительные методы в интерфейс сервиса.

<https://habr.com/ru/post/257925/>

**Как мы работаем со справочниками на интеграционной шине**

подход заключается в том, что каждая бизнес-система хранит справочники локально, и организует у себя ввод данных. При обмене сообщениями между системами интеграционная шина осуществляет трансформацию из формата одной системы в формат другой. При этом происходит и трансформация справочных данных.

Все взаимодействия бизнес-систем происходят через интеграционную шину. Шина (в нашем случае Oracle Service Bus) трансформирует сообщение, которое посылает система Поставщик, в сообщение, понятное системе Потребителю. Такая трансформация включает мапирование значений справочников.  
Данные о том, как справочники мапируются между системами хранятся в реляционной базе данных, в нашем случае — Oracle. В таблицах будет записано, как из значения справочника в одной системе получить значение в другой системе. То есть какая-то такая структура:  
*(source\_system, source\_value, valid\_from, valid\_to, target\_system, target\_value)*  
Данные в справочниках меняются очень редко, а используются очень часто. Чтобы не обращаться каждый раз к базе данных, справочники кэшируются на шине, причем в формате, который шина может сразу использовать.

Якорная система – очень плачевный опыт. При выборке объектов очень низкая скорость данных.

Ускорение возможно только при Денормализация. (Растаскиваю 2 таблицы на 5, более плоские. Линейная структура) увеличение быстродействия будет за счет инструментов субд: …

Ассоц