ООО «ТриниДата»

РУКОВОДСТВО ПО ИСПОЛЬЗОВАНИЮ

системы АрхиГраф.MDM

г. Екатеринбург, 2015-2022

Оглавление

1. Назначение АрхиГраф.MDM

АрхиГраф.MDM, как компонент платформы виртуализации данных АрхиГраф, предназначен для хранения массива основных данных организации (мастер-данных), и предоставления прикладным программным компонентам программного доступа к ним. АрхиГраф.MDM обеспечивает синхронизацию основных данных между различными приложениями.

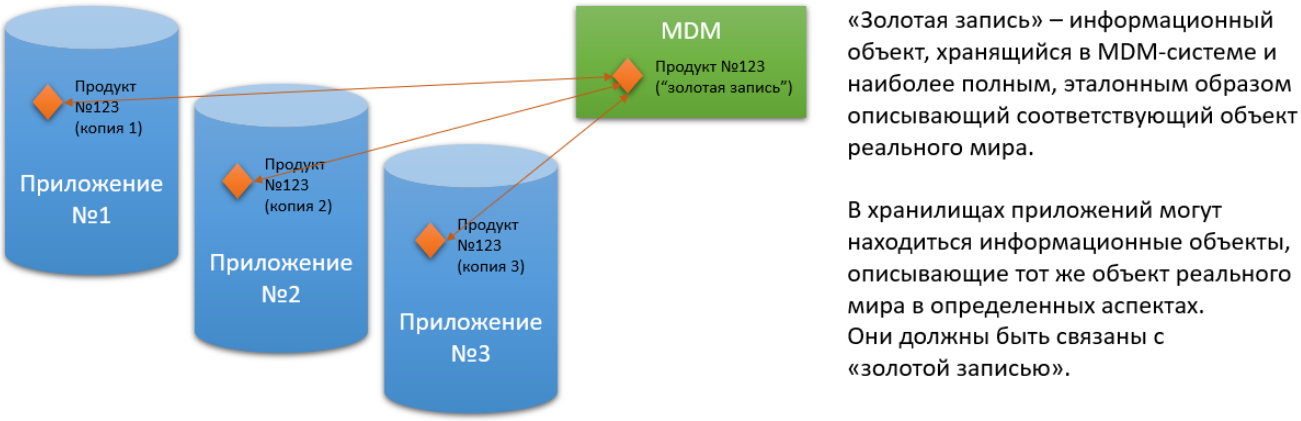

Рис. 1. Принцип синхронизации мастер-данных
между бизнес-приложениями и MDM

Особенность АрхиГраф.MDM состоит в том, что он предоставляет возможность получать доступ при помощи программного интерфейса не только к составу, но и к структуре основных данных – информационной модели. Модель строится по принципам семантических технологий (онтологического моделирования), допуская использование фасетных классификаций объектов (принадлежность объекта более чем к одному типу одновременно), наследуемых наборов атрибутов, множественных значений атрибутов и др. АрхиГраф.MDM обеспечивает разграничение прав доступа приложений-клиентов к мастер-данным, присвоение идентификаторов и контроль уникальности хранимых объектов.

2. Управление данными в АрхиГраф.MDM

Платформа АрхиГраф включает несколько компонентов, каждый из которых предназначен для выполнения функций пользователями определенных ролей. Главные функции каждого из компонентов перечислены на рис. 2. АрхиГраф.Мир – инструмент для онтолога и аналитика предметной области, АрхиГраф.СУЗ – предназначен для инженера по качеству данных и дата-стюарда, Конфигуратор MDM – для администратора информационного обмена.

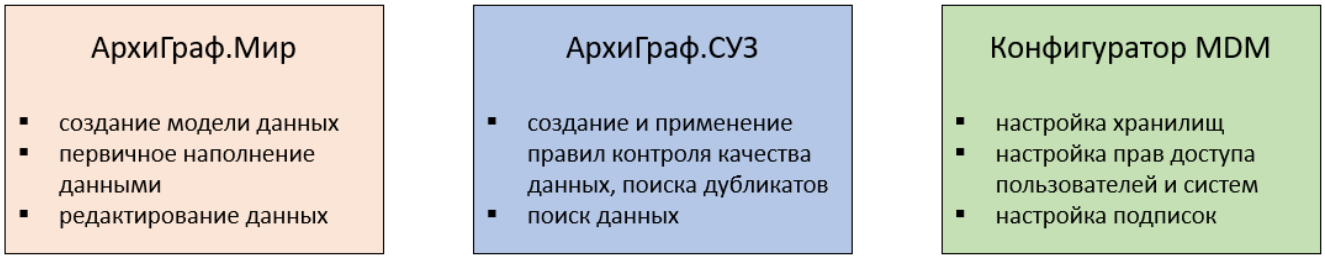

Рис. 2. Основные компоненты платформы АрхиГраф

Редактирование структуры и содержания НСИ и мастер-данных осуществляется через интерфейс редактора онтологий АрхиГраф.Мир, работа с которым описана в Руководстве пользователя АрхиГраф.Мир. Принципы создания онтологической информационной модели и управления ей описаны в методическом пособии «Введение в онтологическое моделирование».

Создание правил контроля целостности информационных объектов, правил поиска дубликатов и обогащения данных выполняется в среде АрхиГраф.СУЗ и описана в Руководстве пользователя АрхиГраф.СУЗ.

Настройка прав доступа пользователей и внешних информационных систем к мастер-данным, настройка хранилищ данных, создание типовых маршрутов согласования изменений осуществляется с помощью интерфейса Конфигуратора MDM или консольной утилиты mdmctl, работа с которыми описана в Руководстве администратора АрхиГраф.MDM.

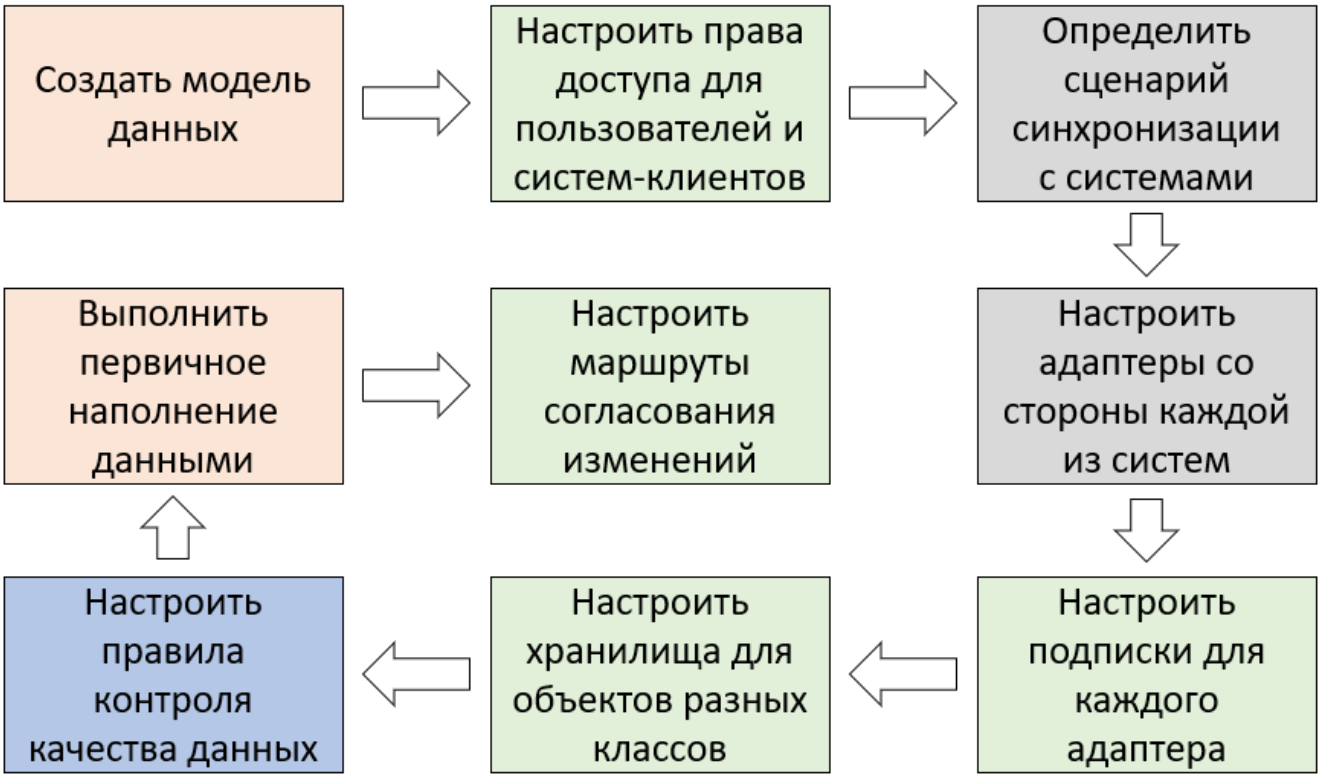

Рис. 3. Структура процесса внедрения АрхиГраф.MDM

Программный доступ к данным под управлением MDM-системы осуществляется через API платформы АрхиГраф.

Данное руководство является дополнением к перечисленным документам и освещает выполнение некоторых типичных задач управления мастер-данными.

На рис. 3 показана общая структура процесса внедрения АрхиГраф.MDM. Цвет блоков на диаграмме соответствует компонентам платформы (см. рис. 2), в которых выполняется каждый из шагов. Обзор общего процесса настройки платформы можно найти также в брошюре «Корпоративные автоматизированные системы на основе онтологических моделей: книга рецептов». Ниже приведен список документов и источников, в которых можно найти информацию по каждому шагу процесса внедрения.

№	Шаг	Источник
1.	Создать модель данных	Руководство по созданию промышленных онтологий Книга рецептов Руководство пользователя АрхиГраф.Мир
2.	Настроить права доступа для пользователей и систем-клиентов	Руководство администратора АрхиГраф.MDM Книга рецептов
3.	Определить сценарий синхронизации с системами	Данный документ
4.	Настроить адаптеры со стороны каждой из систем	Данный документ
5.	Настроить подписки для каждого адаптера	Руководство администратора АрхиГраф.MDM Описание API АрхиГраф.MDM
6.	Настроить хранилища для объектов разных классов	Руководство администратора АрхиГраф.MDM
7.	Настроить правила контроля качества данных	Руководство по созданию промышленных онтологий Руководство пользователя АрхиГраф.СУЗ
8.	Выполнить первичное наполнение данными	Руководство пользователя АрхиГраф.Мир Описание API АрхиГраф.MDM
9.	Настроить маршруты согласования изменений	Руководство администратора АрхиГраф.MDM

3. Основные функциональные особенности АрхиГраф.MDM

В этом разделе дается краткий обзор основных функциональных возможностей АрхиГраф.MDM, в основном сфокусированный на функциях, отличающих его от других продуктов. Подробную информацию по использованию

перечисленных здесь и других возможностей можно найти в описании API платформы АрхиГраф.

3.1. Программный доступ к данным

АрхиГраф.MDM обеспечивает доступ к выполнению операций чтения отдельных объектов (запрос GetObject), поиска множества объектов по комбинации условий (GetObjectsGroup), редактирования одного или нескольких объектов (UpdateObject, UpdateObjectsGroup), удаления объекта или группы объектов (DeleteObject, DeleteObjectsGroup).

Запрос отдельных объектов может выполняться по идентификатору объекта в MDM или в системе-источнике (см. раздел «Хранение в MDM кодов объектов в информационных системах-клиентах»). Запрос множества объектов осуществляется с указанием одного или нескольких классов объектов, а также условий отбора и сортировки.

Основные особенности онтологической модели данных, используемой АрхиГраф.MDM, состоят в следующем:

- Каждый информационный объект («запись») может одновременно относиться более чем к одному классу (типу сущностей).
- Классы образуют иерархию (надкласс-подкласс). Каждый объект подкласса одновременно является объектом всех его надклассов, даже если не включен в них явно.
- Свойства существуют независимо от классов, объекты разных классов могут обладать значениями одного и того же свойства.
- Полный набор свойств, применимых к информационному объекту, складывается из набора свойств, применимых ко всем классам, в которые он входит, и их надклассам.
- Каждый объект может обладать множеством значений каждого свойства, если на возможное количество значений этого свойства не наложены ограничения («кардинальность»).

Отметим несколько важных особенностей запроса на изменение объекта – UpdateObject, или набора объектов – UpdateObjectsGroup:

- В тегах Attribute, в которых передаются устанавливаемые значения свойств объекта(-ов), можно использовать флаг Empty, чтобы очистить значения свойства.
- Флаги AddValue и DelValue позволяют явно указать, что необходимо сохранить или удалить одно из значений свойства, не изменяя другие его значения.

- Флаг ExistingOnly позволяет установить новое значение свойства только в том случае, если ранее объект уже имел какое-либо значение этого свойства.
- Параметр Copy позволяет присвоить одному атрибуту значение другого атрибута – например, скопировать «Название» в «Читаемое название». Эта возможность полезна при рефакторинге модели, когда вместо свойства со старым идентификатором создается новое свойство, и необходимо перенести в него значения старого свойства.

При выполнении запросов на удаление объекта(-ов) по умолчанию не происходит очистка ссылок на удаляемые объекты со стороны других объектов. Чтобы такая очистка была выполнена, необходимо указать в запросе флаг DeleteReference. Флаг VerifyReference позволяет выполнить проверку на наличие ссылок на удаляемый объект и отменить операцию удаления, если хотя бы одна такая ссылка будет найдена.

Все запросы к API MDM обрабатываются с учетом прав доступа обращающейся системы-клиента. Для идентификации системы-клиента в каждом запросе должно указываться значение атрибута Originator. Аутентификация может производиться на основании значения дополнительного атрибута Token, который содержит постоянный токен для каждой системы-отправителя (в доверенной среде), или с помощью механизма JWT – JSON web tokens. Список систем-клиентов и права доступа для них настраивается в конфигураторе MDM.

3.2. Программный доступ к модели данных

API АрхиГраф.MDM предоставляет и доступ к модели данных – определениям классов и свойств. Для считывания всей модели данных можно воспользоваться запросом GetDataSchema или GetDataSchemaCompact. Оба запроса возвращают полный список классов и свойств модели, доступных системе-клиенту MDM, отправившей запрос – если только не указан параметр StartElement, ограничивающий получаемое описание определенной ветвью модели.

Описание модели состоит из перечня классов и свойств, которыми могут обладать объекты этих классов. Запрос GetDataSchema возвращает список классов, в котором для каждого класса указан список свойств, применимых к его элементам – в результате каждое свойство может быть упомянуто много раз, если оно применимо к объектам более чем одного класса. Запрос GetDataSchemaCompact возвращает отдельно список свойств и список классов, причем с каждого класса имеются ссылки на применимые к его объектам свойства и на те свойства, значениями которых могут являться объекты каждого класса.

Можно получить модель данных и по-другому – в виде описания отдельных ее элементов. Для этого нужно выполнить запросы GetObjectsGroup на получение

объектов классов owl:Class, owl:DataProperty и owl:ObjectProperty, которые представляют соответственно классы, литеральные и ссылочные свойства согласно спецификации OWL консорциума W3C. Для каждого свойства будут указаны значения свойств rdfs:domain и rdfs:range. Первое из них указывает на класс(ы), которые могут обладать значением свойства, второе – на типы XSD (для литералов) или другие классы (для ссылок), которые являются его диапазоном значений.

Спецификация OWL подразумевает возможность создания сложных конфигураций классов для описания областей применения и диапазонов значений свойств, например: «Значениями свойства «Имеет код ОКВЭД» могут обладать объекты классов «Юридическое лицо» и «Физическое лицо», за исключением тех, которые не имеют значения булевского свойства «Зарегистрирован как индивидуальный предприниматель», равного true». АрхиГраф.MDM поддерживает только простые способы комбинирования классов с помощью логических операторов И, ИЛИ. В ответе на запрос GetDataSchema способ комбинирования классов в области применения и диапазоне значений указывается с помощью атрибутов DomainRange и RangeIntersection. По умолчанию, если значения этих атрибутов не указано, областью применения или диапазоном значений является объединение классов (ИЛИ), указанных во вложенных тегах Target в описании свойства. Если же значение этих атрибутов равно true, то областью применения или диапазоном значений является пересечение классов (И).

Существует возможность изменять структуру модели данных с помощью запросов UpdateObject на изменение объектов классов owl:Class, owl:DataProperty и owl:ObjectProperty. Это позволяет создавать новые классы и свойства или изменять существующие непосредственно во время работы системы. Изменения, внесенные в модель, по умолчанию немедленно вступают в силу. Существует возможность запланировать вступление изменений в силу на какую-либо дату в будущем (см. ниже раздел «Поддержка темпоральности»).

Отметим, что при изменении модели MDM не очищает физически значения свойств объектов, которые были удалены из модели или перестали быть применимы к объектам определенных классов. Можно получить значения таких свойств, если указать в запросе GetObject или GetObjectsGroup флаг AllAttributes="1".

3.3. Хранение в MDM кодов объектов в информационных системах-клиентах

Для идентификации объектов в MDM используется идентификатор MDM, передаваемый в атрибуте Code в запросах. Этот идентификатор имеет форму URI (в запросах может указываться как полностью, так и в сокращенном формате с префиксом). Идентификатор MDM может быть указан явно системой-клиентом в момент создания объекта. При этом система-клиент несет ответственность за

проверку того, что объект с таким идентификатором не существует; если такой объект уже существует в MDM, вместо создания нового объекта произойдет изменение имеющегося. Если идентификатор не указать в запросе на создание объекта, то его сгенерирует сама MDM, которая при этом гарантирует его уникальность.

Для того, чтобы иметь возможность впоследствии найти объект, не сохраняя на своей стороне его код MDM, система-отправитель может при создании объекта передать его локальный идентификатор в специальном атрибуте LocalCode. Идентификатор системы-источника, по запросу которой был создан объект, сохраняется в специальном атрибуте SourceSystem объекта. Значения этих атрибутов доступны среди обычных свойств объекта и в онтологической модели имеют идентификаторы http://trinidata.ru/archigraph-mdm/LocalCode и http://trinidata.ru/archigraph-mdm/SourceSystem. Каждый из них может иметь только одно значение для каждого объекта. Эти свойства можно использовать как условия отбора объектов при выполнении запроса GetObjectsGroup. Можно извлечь объект и конкретный по значению LocalCode запросом GetObject, указав значение локального кода в атрибуте LocalCode тега GetObject.

MDM поддерживает особую ситуацию, когда система-клиент, создавшая объект, имеет несколько экземпляров. Это могут быть, например, отдельные базы 1С, или несколько экземпляров прикладной системы, развернутые для разных территориальных подразделений организации. В этом случае при создании и поиске объекта по локальному идентификатору можно передать значение еще одного специального атрибута – SystemInstance, который хранит идентификатор конкретного экземпляра системы-клиента.

MDM не контролирует значения атрибута SystemInstance – в отличие от кода системы-клиента, это просто строка, которая может принимать любые значения. Каждый объект может иметь множество локальных идентификаторов, каждый из которых связан с отдельным SystemInstance. При запросе свойств объекта через API MDM в этом случае будет возвращено несколько значений служебного свойства http://trinidata.ru/archigraph-mdm/LocalCode, каждое из которых аннотировано собственным значением атрибута SystemInstance. В ответе MDM это будет выглядеть так:

```
<Attribute AttributeId="http://trinidata.ru/archigraph-
mdm/LocalCode" SystemInstance="Система1" Type="Literal" Value="1"/>

<Attribute AttributeId="http://trinidata.ru/archigraph-
mdm/LocalCode" SystemInstance="Система2" Type="Literal" Value="2"/>
```

Изменение идентификаторов MDM и локальных идентификаторов объектов невозможно. Можно только удалить объект и создать новый с таким же

идентификатором. При этом надо иметь в виду, что все ссылки, существовавшие на идентификатор MDM, продолжат указывать на новый объект.

3.4. Права доступа

Для каждой системы-клиента могут быть установлены права доступа к объектам определенных классов и значениям определенных свойств онтологической модели. По умолчанию любая система-клиент имеет полный доступ ко всем данным и структуре модели. Доступ можно ограничить, установив к определенным классам и/или свойствам уровень доступа «нет», «только чтение» или «редактирование с подтверждением». В последнем случае при попытке выполнения запроса на редактирование или удаление объекта со стороны системы-клиента будет создана особая информационная сущность – запрос на изменения, который должен пройти маршрут согласования в соответствии с шаблоном, настраиваемым в АрхиГраф.СУЗ для объектов каждого класса.

Права доступа, установленные для надкласса, действуют и на его подклассы. Если на какой-либо объект действуют сразу несколько правил доступа (например потому, что он входит одновременно в несколько классов, для которых существуют разные правила доступа), выбирается наиболее строгое из всех применимых правил. Флаг ReturnRights в запросе к объекту позволяет получить в явном виде описание прав доступа, действующих для данного объекта и системы-клиента.

Под «системой-клиентом» в контексте разграничения прав может пониматься как реальная сторонняя информационная система – клиент MDM, так и определенная группа пользователей. АрхиГраф.Мир и АрхиГраф.СУЗ выполняют аутентификацию пользователей с помощью стороннего провайдера авторизации – KeyCloak, который в свою очередь может быть интегрирован с Active Directory (может быть настроена и прозрачная авторизация Kerberos). На уровне провайдера аутентификации пользователь может быть включен в одну или несколько групп, а на уровне АрхиГраф.Мир и АрхиГраф.СУЗ может быть настроено сопоставление этих групп с идентификаторами систем-клиентов MDM. Таким образом, АрхиГраф.Мир и АрхиГраф.СУЗ при обращении к API MDM передают идентификатор той системы-клиента, которая соответствует группе пользователя. Подобный способ аутентификации и авторизации рекомендуем использовать при создании любых прикладных программных компонентов, работающих с АрхиГраф.MDM.

3.5. Поддержка темпоральности – работа с разными версиями модели и данных

Большинство информационных систем хранит в виде объектов данных только описание текущего состояния бизнес-объектов. Сведения об истории изменения свойств объектов данных если и фиксируются, то только в метаданных,

таких как журналы изменений объектов. Это значит, что большинство платформ хранения данных не дает возможности напрямую получить состояние информационного объекта на какой-то момент времени.

В API платформы АрхиГраф реализован другой подход к работе со временем (темпоральностью). В любом запросе на чтение или изменение данных (GetObject, GetObjectsGroup, UpdateObject, UpdateObjectsGroup, DeleteObject, DeleteObjectsGroup) можно указать параметр Time – метку времени. В ответ на запросы о получении информации об объекте будут возвращены те значения свойств объекта, которые были актуальны на указанную дату и время. Платформа хранит даты начала и окончания существования объектов, т.е. и сам набор получаемых объектов в ответ на запросы с одинаковыми условиями фильтров, но разной меткой времени, может быть разным. Например, на 1 февраля к категории «Активные» мог относиться один набор клиентов, а на 1 мая – другой. Заметим, что запросы на извлечение объектов с указанием метки времени работают существенно медленнее, чем запросы текущего состояния объектов. Есть также возможность получить полную историю изменения состояний объекта, указав в запросе на извлечение объектов параметр WithHistory="1".

Если метку времени указать в запросе на создание, изменение или удаление объекта, то изменения будут внесены указанной датой. Можно изменить объект «задним числом» – при этом текущие свойства объекта могут остаться неизменными, если между указанной датой и текущим моментом был выполнен другой запрос на изменения. Например, если текущая дата – 01.02.2022, изменения в объект вносятся датой 15.01.2022, но те же свойства были изменены запросом от 22.01.2022, то выполнение запроса на изменение приведет только к записи новых исторических значений. Если в такой ситуации вносить изменения в объект датой 25.01.2022, то изменятся и текущие значения свойств объекта.

Можно и запланировать изменения на будущее. Планирование изменений может пригодиться, например, при работе с тарифами: можно задать новые значения свойств информационных объектов с 1 января нового года. Эти изменения попадут в запланированные и не повлияют на текущее состояние объекта, но когда наступит указанная дата – они автоматически вступят в силу, изменив текущие значения свойств объекта. При этом получить запланированные значения свойств можно будет и до их вступления в силу, если указать в запросе метку времени, будущую по сравнению с датой планируемых изменений.

Существует возможность восстановить объект по состоянию на определенный момент времени («откатить» сделанные изменения) с помощью запроса RestoreObjects к API MDM.

Работа с темпоральностью модели организована немного по-другому. Существует специальный запрос CreateEndpoint, который позволяет создать

«виртуальную точку доступа», содержащую версию модели данных на дату, указанную в параметре Date. Виртуальная точка доступа создается физически как отдельный набор данных в графовой СУБД. Клиент MDM может отправлять запросы к этой точке доступа. При этом MDM будет использовать для ответа текущие данные (если только не указана особая дата и в запросе к данным), но представлять их в соответствии с той структурой модели, которая существовала на дату, указанную при создании виртуальной точки доступа. Это позволяет смотреть на текущие данные «сквозь призму» исторического состояния модели.

По окончании работы с виртуальной точкой доступа ее необходимо удалить запросом DeleteEndpoint.

Описанные механизмы работы с темпоральностью модели и данных позволяют работать с их плавным версионированием, избавляя приложение-клиент от необходимости хранить на своей стороне сведения об изменении структуры модели и свойств информационных объектов.

3.6. Многоязычность данных

Спецификации онтологического моделирования предусматривают, что каждое строковое свойство одного объекта может иметь множество значений, каждое из которых аннотировано принадлежностью определенному языку. Например, свойство «Название» (rdfs:label) объекта КомпанияАльфа может иметь значение «ООО "Альфа"» на русском языке и «Alpha, LLC» на английском (а также любое количество других значений на других языках).

В платформе АрхиГраф.MDM один из языков выбирается в качестве языка по умолчанию – это делается при помощи раздела «Константы» Конфигуратора MDM, константа «Язык данных по умолчанию» (data_default_language), ее значение – ISO-код языка (например, ru). Полный список языков, доступных в платформе, хранится в таблице languages базы данных настроек MDM.

При работе с API MDM при записи в базу значений свойств строкового типа можно указать атрибут Lang="[ISO-код языка]" в каждом из тегов Attribute. Если язык не указан явно, то считается, что значение задано на языке по умолчанию. Тот же атрибут можно использовать в теге Filter при указании условий извлечения группы объектов из MDM, и в ряде других запросов.

Если значение атрибута Lang не указано на уровне корневого тега запроса на извлечение объекта, то MDM вернет значения свойств объекта только на указанном языке. В противном случае MDM возвращает значения строковых свойств объекта на всех языках, на которых имеются заданные значения. В этом случае в ответе может быть несколько тегов Attribute для каждого свойства, которые отличаются между собой значением атрибута Lang, например:

```
    <Attribute AttributeId="http://www.w3.org/2000/01/rdf-
schema#label" Type="Literal" Value="Персона" Lang="RU"/>

    <Attribute AttributeId="http://www.w3.org/2000/01/rdf-
schema#label" Type="Literal" Value="Person" Lang="EN"/>
```

Параметр MessageLang в запросах к MDM позволяет указать язык, на котором MDM должна вернуть сообщение о результате выполнения операции.

3.7. Работа с пространственными данными

АрхиГраф.MDM поддерживает работу с несколькими типами пространственных данных:

- archigraph:PointType
- archigraph:LineType
- archigraph:PolygonType
- archigraph:MultiPolygonType

Значения свойств, относящихся к этим типам, должны сохраняться в виде строки в формате GeoJSON (даже при работе с XML-форматом запросов).

Сохранять значения свойств объектов, имеющих такой тип, можно независимо от того, в каком хранилище физически находятся объекты этого класса. Однако для использования специфической функциональности работы с пространственными данными, например – поиска объектов, координаты которых входят в определенный многоугольник, необходимо, чтобы хранилище поддерживало работу с пространственными данными (например, Postgres с расширением PostGIS). Для использования этой функциональности нужно указать тег Geometry в запросе на извлечение объектов по условиям GetObjectsGroup (детальное описание см. в документации на API АрхиГраф.MDM).

3.8. Использование правил контроля целостности данных, правил логического вывода и правил поиска дубликатов

Правила контроля целостности данных (форматно-логического контроля) создаются в интерфейсе АрхиГраф.СУЗ. Работа с ними описана в Руководстве пользователя АрхиГраф.СУЗ. Правила позволяют описывать как условия на связность объектов, так и на соответствие значений свойств объектов определенным условиям (с помощью регулярных выражений).

Для применения правил нужно указать атрибут Check="1" в корневом теге запроса на создание или извлечение объекта. Если объект не соответствует условиям правила, то в данных MDM будет сформирован отдельный информационный объект класса shacl:ValidationResult, свойства которого указывают на объект, не прошедший валидацию, и правило, условия которого были

нарушены. MDM также вернет сообщения о нарушенных правилах в атрибуте Message тега Item (возвращаются названия нарушенных правил).

В свойствах правила можно указать, что оно является обязательным для выполнения. В этом случае при попытке создания или редактирования объекта, нарушающего правило, MDM откажет в выполнении операции.

Правила логического вывода позволяют автоматически дополнять данные MDM новой информацией в случае, если выполняются предпосылки, описанные в условии правила. Для применения правил логического вывода нужно указать атрибут Logic="1" в запросе на создание или редактирование объекта.

При работе в редакторе АрхиГраф.Мир можно включить или отключить применение правил обоих типов при помощи переключателей на странице «Настройки».

Правила поиска дубликатов являются частным случаем правил логического вывода. Их задача состоит в обнаружении пар информационных объектов, которые, возможно, описывают один и тот же реальный объект. Результат применения правила в зависимости от выбранных при его создании опций может состоять в пометке двух объектов как возможных дубликатов, или в их автоматическом объединении. Для работы с помеченными объектами предназначен интерфейс «Поиск дубликатов» в АрхиГраф.СУЗ, описанный в Руководстве пользователя этого продукта.

На уровне API MDM для объединения возможных дубликатов предназначен запрос MergeObject, описанный в Руководстве по API АрхиГраф.MDM. Этот запрос, как и соответствующая функция в интерфейсе АрхиГраф.СУЗ, присоединяет один объект к другому, а также заменяет все существующие ссылки на присоединяемый объект на тот объект, к которому он был присоединен. В опциях этого запроса можно указать политику объединения значений атрибутов, отличающихся у двух объектов.

3.9. Инструменты диагностики и отладки

В запросах к MDM можно указать опции, позволяющие получить отладочную информацию о выполнении запросов.

Опция ReturnTime позволяет получить время выполнения запроса на стороне MDM (исключая время передачи запроса и ответа).

TimeLog позволяет записать детальный журнал процедуры исполнения запроса в лог-файл на сервере MDM.

ReturnTimeLog позволяет получить этот журнал в ответе на запрос. Журнал представляет собой набор записей, каждая из которых соответствует вызову одной

из функций внутри MDM. Для каждой функции указывается количество ее вызовов, суммарное время выполнения всех вызовов этой функции и среднее время одного вызова. Время выполнения указывается накопительным итогом, т.е. время вызова функции верхнего уровня включает время вызова всех функций, которые из нее вызываются.

Опция StorageRequestLog позволяет записать в лог-файл на сервере MDM журнал запросов, которые MDM выполняет к хранилищам данных.

3.10. Агрегирующие операции

Для выполнения агрегирующих операций в запросах на извлечение групп объектов предназначен тег Aggregate. Внутри этого тега указывается операция (суммирование, поиск наибольшего значения и др.), атрибут, к которому она применима, а также атрибут, по которому происходит группировка записей (не обязателен). Пример:

```
<Aggregate Operation="SUM">
    <Attribute Name="Total" Code="Стоимость"/>
    <GroupBy Code="Регион"/>
</Aggregate>
```

В данном примере произойдет суммирование значений атрибута «Стоимость» для всех извлекаемых объектов с группировкой по свойству «Регион». Значение будет возвращено под именем Total, указанным в запросе, в таком формате:

```
<Aggregate Name="Total" Value="112234"/>
```

Ответ на запрос GetObjectsGroup с указанной внутри операцией Aggregate отличается по формату от обычного. Тег Items не содержит вложенных тегов Item описывающих конкретные записи, а содержит только теги Aggregate, в которых содержится результат выполнения агрегирующих операцией. Число тегов соответствует числу агрегирующих операций, указанных в запросе.

3.11. Проверки корректности запросов к MDM и их отключение

По умолчанию MDM выполняет проверки корректности запросов, в том числе запросов на извлечение, по следующим критериям:

- корректность синтаксиса запроса
- наличие в модели данных всех классов, указанных в условии запроса
- наличие в модели данных всех свойств, указанных в условии запроса
- соответствие переданных значений свойств диапазонам значений этих свойств

- наличие в данных MDM всех объектов, указанных в качестве значений свойств – как в фильтрах, так и при присвоении значений.

Поскольку эти проверки могут занимать длительное время, предусмотрена возможность отключения некоторых из них. При этом приложение, выполняющее запрос, само ответственно за его корректность.

Флаг WithoutRefChecking (по умолчанию = 0, проверка включена) на уровне корневого тега запроса позволяет отключить проверку наличия в данных MDM всех объектов, указанных в качестве значений свойств в фильтрах.

Флаг IgnoreIfNotCorrect (по умолчанию = 0, проверка включена) на уровне атрибута позволяет отключить проверку соответствия запроса модели данных: наличие в модели данных всех свойств, указанных в условии запроса, и соответствие переданных значений свойств диапазонам значений этих свойств. Отключение этой проверки может использоваться с целью ускорения выполнения запроса на извлечение объекта.

Одна из особенностей платформы состоит в том, что при изменении состава свойств, применимых к объектам определенных классов, значения этих свойств, уже присвоенные объектам, не удаляются из базы данных. С помощью флага AllValues (по умолчанию = 0, извлечение не соответствующих модели значений свойств отключено) можно получить значения этих свойств.

Дополнительно, флаг Check (по умолчанию = 0, проверка отключена) позволяет включить проверку соответствия объектов, затрагиваемых запросом, на соответствие логическим условиям, которые пользователь может сконфигурировать в АрхиГраф.СУЗ.

3.12. Подписка на изменения объектов модели и данных в MDM

Одна из важнейших возможностей АрхиГраф.MDM состоит в предоставлении интерфейса подписки на изменения объектов определенных классов и/или модели данных. Подписка функционирует следующим образом: система-клиент MDM (Originator) при помощи запроса UpdateSubscription к API MDM устанавливает подписки на интересующие ее классы модели (можно подписаться и на всю модель целиком, указав специальное значение __root__ в атрибуте Code тега ObjectType). В запросе на создание подписки передаются реквизиты подключения к брокеру RabbitMQ или Kafka, а также имя очереди (топика), через которую приложение готово получать уведомления от MDM.

В момент создания, изменения или удаления объектов тех классов, на которые подписалась система-клиент, MDM помещает в указанную очередь пакеты с информацией о происшедших изменениях. При этом подключение к брокеру выполняется с теми реквизитами, которые были указаны при создании подписки.

Приложение с помощью функций API может проверить наличие подписки (GetSubscription), изменить ее свойства (UpdateSubscription) или удалить подписку (DeleteSubscription). Каждое приложение может создать любое количество подписок. Управлять подписками можно и вручную со стороны Конфигуратора MDM.

Важно отметить, что MDM имеет два механизма работы с брокерами сообщений: описанный здесь инструмент работы с подписками, в котором инициатором отправки сообщений выступает MDM, и асинхронный режим обработки запросов основного API (в том числе запросов на управление подписками), в котором инициатором выступает система-клиент. При использовании асинхронного режима обработки запросов основного API с помощью Конфигуратора MDM настраиваются пары очередей, одна из которых служит для отправки запросов в MDM, а другая – для получения ответов.

3.13. Поддержка работы MDM на нескольких площадках

Один из вариантов развертывания АрхиГраф.MDM предусматривает создание нескольких экземпляров (или кластеров) платформы. Каждый экземпляр может соответствовать территориальному или функциональному подразделению организации, использующей платформу. При этом существует возможность установить взаимную подписку нескольких кластеров на объекты и модель данных друг друга. Взаимная подписка настраивается с помощью Конфигуратора MDM как обмен данными между каждой парой копий АрхиГраф.MDM. Для каждого экземпляра платформы используется собственный идентификатор системы-отправителя, с которым она обращается к другим экземплярам системы.

Подписка может быть как полной (с односторонней или двусторонней синхронизацией), так и частичной. При полной двусторонней синхронизации модель и данные на обеих площадках будут идентичны, независимо от того, со стороны какой площадки вносились изменения. При частичной синхронизации появляется возможность иметь как наборы данных, общие для всех площадок, так и специфичные для каждой.

При развертывании одного кластера MDM также может использоваться понятие нескольких площадок, которое в этом случае имеет другой смысл. Различные площадки могут использоваться для развертывания экземпляров хранилищ, подключенных к АрхиГраф.MDM. Для выполнения запросов только в хранилищах, относящихся к определенной площадке, можно указать в корневом теге любого запроса специальный атрибут UsePlace. Его значение – строковое название площадки, которое указывается в свойствах хранилища с помощью Конфигуратора MDM.

4. Синхронизация модели и мастер-данных с приложениями

4.1. Архитектура обмена

Подробное описание программного взаимодействия клиентских приложений с данными под управлением MDM см. в документе API платформы АрхиГраф.

Описываемый ниже механизм обмена предназначен, в первую очередь, для обмена основными данными между прикладным ПО и MDM-системой АрхиГраф.MDM. Аналогичный механизм может использоваться и для обмена информацией напрямую между прикладными программными компонентами: в этом случае может потребоваться дополнительная настройка маршрутизации сообщений на уровне шины, которая основывается на типах их содержимого. В любом случае, принцип обмена базируется на использовании не зависящего от структуры данных, моделе-ориентированного формата. Для формирования и разбора такого формата со стороны каждого приложения необходимо анализировать модель данных, описание которой также предоставляется MDM-системой при помощи ее программного интерфейса.

Принцип работы программного интерфейса АрхиГраф.MDM основан на обмене XML-пакетами с другими программными системами. Транспорт пакетов может осуществляться несколькими способами:

- при помощи очередей MQ (предпочтительный способ, и единственный возможный в высоконагруженных и высоконадежных архитектурах),
- с помощью HTTP-запросов в основном синтаксисе MDM,
- с помощью REST-интерфейса.

Форматы пакетов и последовательность обмена ими не зависят от способа передачи информации. В любом случае, запросы и ответы представляют собой XML-пакеты одинакового формата, для доставки которых могут использоваться различные транспорты.

Таким образом, в процессе обмена каждое приложение, желающее отправить запрос к АрхиГраф.MDM, должно сформировать XML-сообщение, и отправить его в MDM при помощи соответствующего транспорта. MDM-система возвратит ответ при помощи того же транспортного протокола.

В рекомендуемой конфигурации, когда обмен сообщениями происходит через MQ, для MDM и каждого приложения создается по две очереди: входящих и исходящих сообщений. В качестве менеджеров очередей могут использоваться как коммерческие, так и OpenSource решения. Мы рекомендуем использовать Apache Kafka или RabbitMQ. Преимущество данной конфигурации состоит в том, что MDM-система и приложения – ее клиенты обрабатывают сообщения в

асинхронном режиме. Благодаря этому не может возникнуть ситуации перегрузки того или иного сервера или приложения, повышается надежность обмена.

В сложных интеграционных архитектурах маршрутизацию сообщений соответствующим получателям может обеспечивать корпоративная сервисная шина (ESB), отвечающая за перемещение сообщений из исходящей очереди отправителя во входящие очереди получателей. Для выбора подходящего маршрута шина может использовать информацию об отправителе, и анализировать содержимое пакета. Можно и напрямую включить в пакет сведения об адресате. В качестве ESB, работающей в связке с АрхиГраф.MDM, могут использоваться решения различных производителей, включая Open Source-продукты (Apache Synapse, WSO2 Message Broker).

Архитектура программных компонентов, участвующих в обмене, при реализации описанного принципа будет выглядеть таким образом:

Рис. 4. Архитектура обмена с АрхиГраф.MDM

Типичный сценарий синхронизации НСИ и мастер-данных между бизнес-приложениями с использованием MDM имеет такую структуру:

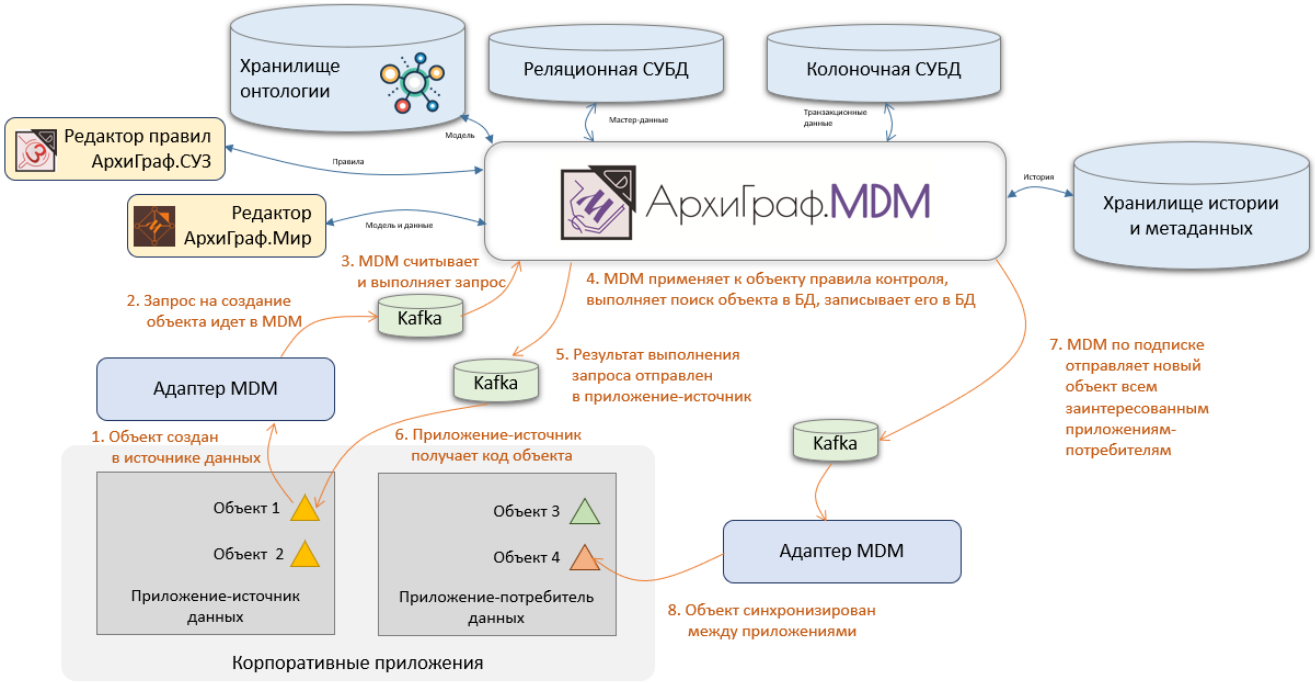

Рис. 5. Типовой сценарий синхронизации мастер-данных между бизнес-приложениями с помощью MDM

5. Реализация обмена с MDM со стороны приложений

Для обработки и формирования сообщений со стороны приложений-клиентов должны существовать адаптеры информационного обмена, работающие по стандартному алгоритму, который изложен ниже. Имеется стандартная библиотека таких адаптеров для различных платформ; для обсуждения возможности их использования можно обратиться к разработчику MDM-системы, компанию «ТриниДата».

1. В первую очередь, на стороне адаптера должен быть реализован парсер модели данных. Необходимо сохранить содержимое пакета DataSchema в собственную базу данных. Как минимум, нужно выделить оттуда типы сущностей (классы), и атрибуты, присущие каждому из них.
2. Необходимо реализовать механизм настройки мэппинга (сопоставления) сущностей информационной модели с элементами локального хранилища данных. Такими элементами могут выступать таблицы базы данных и их столбцы, или объекты программной платформы (например, Справочники для 1С), и их атрибуты. В любом случае, интерфейс настройки мэппинга должен обладать следующими возможностями:
   - Сопоставление классов информационной модели таблицам/объектам платформы;
   - Сопоставление атрибутов информационной модели столбцам таблицы/атрибутам объектов платформы;

- Задание нестандартной процедуры-обработчика на уровне атрибута информационной модели;
- Задание нестандартной процедуры-обработчика на уровне класса (опционально).

В частности, для атрибутов, которые являются ссылками на объекты других типов, желательно предусмотреть два варианта хранения связи: в атрибуте объекта (если они связываются один к одному), или в таблице-связке (если объекты связываются один ко многим). При использовании связей один ко многим в информационной модели будет задано, что атрибут, выражающий связь, может иметь множество значений. Подробности реализации этой схемы приведены далее. В любом случае, при настройке связей между объектами, необходимо также указать имена полей, хранящих код MDM и локальный код в таблице, на которую ссылаются связи (эти поля могут называться стандартным образом во всех таблицах – например, id и mdm_code).

Настройки мэппинга, которые администратор системы выполняет в этом интерфейсе, сохраняются в локальном хранилище данных приложения, и используются разборе входящих и формировании исходящих сообщений.

3. Блок обработки входящих сообщений Package должен реализовывать следующую логику для каждого элемента Item, содержащегося в пакете (в случае стандартной процедуры обработки, т.е. если не задана процедура специального обработчика):
   - Определить элемент локального хранилища, куда должны сохраняться объекты данного класса. Если объект является членом нескольких классов, вся обработка выполняется для него несколько раз – по одной итерации цикла для каждого класса. Проверить наличие прав доступа на изменение объектов такого типа.
   - Выполнить поиск объекта в хранилище по глобальному идентификатору. Глобальный идентификатор должен содержаться в свойстве, которое одинаково называется для всех элементов локального хранилища – например, mdm_code. Если объект найден, происходит переход к процедуре его обновления, если не найден – к процедуре создания.
   - В случае создания объекта, необходимо добавить в хранилище новую запись, присвоить ей значение mdm_code, и затем перейти к процедуре обновления.
   - Процедура обновления проходит по всем атрибутам объекта, поступившим в составе пакета, и для каждого из них формирует часть запроса на обновление. Эта операция выполняется только для тех

атрибутов, для которых найден соответствующий элемент локального хранилища данных – в противном случае, значение атрибута игнорируется. Если в пакете отсутствует значение для какого-либо атрибута, имеющего мэппинг, и у этого атрибута уже есть значение в локальной системе – оно остается без изменений. Очищаются значения только тех атрибутов, для которых пришло пустое значение (при этом в базах данных необходимо присваивать полям записи NULL, если значение атрибута пусто).

- При сохранении изменений должны отключаться обработчики, регистрирующие журнал изменений для отправки (см. следующий пункт), или журнал должен очищаться после их срабатывания. Это необходимо для исключения отправки эхо-сообщений в адрес MDM.
- Каждую итерацию цикла, связанную с сохранением какой-либо сущности, необходимо заключать в конструкцию try … catch. При возникновении исключения, система должна записать информацию о нем в свой журнал событий, приложив исходный XML-код пакета, и продолжить обработку.
- Если при сохранении изменений обнаружена ссылка на другой объект, который еще не известен системе, она должна выполнить следующие действия: определить тип объекта, на который сделана ссылка, по модели данных; создать пустой объект такого типа, и присвоить ему mdm_code, равный значению ссылки; отправить запрос GetObject с кодом этого объекта. В результате, через некоторое время система получит информацию об этом объекте, и целостность данных восстановится. Данная возможность предназначена для автоматического восстановления после сбоев обмена.
- При запуске задания обработки входящих сообщений по расписанию следует принять меры для того, чтобы не возникла ситуация, когда два экземпляра этого задания работают одновременно. Для этого можно ввести флаг блокировки в локальном хранилище данных – при его обнаружении, вновь запущенный экземпляр задания должен прекратить работу. Флаг блокировки должен автоматически сбрасываться по истечении разумного промежутка времени, чтобы исключить его «зависание» в случае возникновения не обработанного исключения в коде адаптера.
- В общем случае, необходимо обрабатывать ситуацию, когда для одного атрибута приходит несколько значений. Приложению разрешается использовать только первое значение, если его модель данных не позволяет хранить несколько значений для атрибутов, но в этом случае приложение не сможет отправлять запросы на изменение значения атрибутов (см. далее). В то же время, множества значений могут иметь

мэппинг на подчиненную таблицу в приложении. В этом случае идентификаторы значений в подчиненной таблице не играют никакой роли. Чаще всего такая ситуация используется на практике для связывания объектов разных типов, т.е. для атрибутов, чьи значения имеют тип Reference. В этом случае приложение может хранить таблицу-связку для каждой пары типов объектов, или – лучше – иметь одну универсальную таблицу, которая описывает связи любых объектов между собой.

4. Блок формирования исходящих сообщений Items должен реализовывать следующую логику:
   - При помощи механизма отслеживания изменений (триггеры на таблицах БД, процедуры-обработчики для платформенных решений) формировать журнал изменений в записях тех контейнеров локального хранилища данных, которые имеют мэппинг с информационной моделью.
   - С определенной периодичностью (раз в несколько минут) выполнять обработку журнала, формируя исходящие пакеты UpdateObject с информацией об изменениях. Сущности одного типа, или нескольких взаимосвязанных типов – например, клиент и его адреса – могут отправляться одним пакетом. Сущности разных типов желательно отправлять разными пакетами, чтобы упростить диагностику возможных проблем, а также потому, что на типе содержимого могут быть основаны правила маршрутизации.
   - Для формирования пакета выполняется преобразование, обратное описанному в предыдущем пункте. При этом, если передаваемый элемент не имеет присвоенного значения mdm_code – он идентифицируется локальным кодом, значение которого помещается в атрибут LocalCode тега Item. Ссылки на такой элемент возможны только в рамках того же пакета UpdateObject, и осуществляются путем указания локального кода объекта в значении атрибута тега Attribute. Таким образом, например, пакет для создания двух сущностей – клиента и его адреса – будет выглядеть так:

```
<?xml version="1.0" encoding="UTF-8"?>
<UpdateObject Originator="1C">
  <Item LocalCode="abc123" Name="ООО \"Альфа\"">
    <Type TypeId="Компания"/>
  </Item>
  <Item LocalCode="def456" Name="Ленина 10-59">
    <Type TypeId="Адрес"/>
    <Attribute Type="LocalCodeReference"
AttributeId="ПринадлежитКомпании" Value="abc123"/>
  </Item>
</Package>
```

Если по какой-то причине элемент, на который имеется ссылка с отправляемой сущности, еще не имеет MDM-кода, и эти сущности нельзя поместить в один пакет – нужно задержать отправку сущности до тех пор, пока код элемента не станет известен. Для этого можно поместить запись об элементе в конец журнала изменений, чтобы она обработалась при следующем запуске процедуры отправки изменений.

Повторная отправка одной и той же сущности, не имеющей MDM-кода – например, если сущность была отредактирована сразу после создания, до получения постоянного кода – является штатной ситуацией, и корректно обрабатывается MDM-системой.

При редактировании тех же сущностей, которым уже присвоены коды MDM, пакет может выглядеть так:

```
<?xml version="1.0" encoding="UTF-8"?>
<UpdateObject Originator="1C">
  <Item Code="prefix:afebb4394" Name="ООО \"Альфа\"">
    <Type TypeId="Компания"/>
  </Item>
  <Item Code="prefix:fbdc54ad5" Name="Ленина 10-59">
    <Type TypeId="Адрес"/>
    <Attribute Type="Reference" AttributeId="ПринадлежитКомпании"
Value="prefix:afebb4394"/>
  </Item>
</Package>
```

Возможна и смешанная ситуация, когда одна сущность в пакете создается, другая – редактируется.

- В пакете необходимо указать значения всех атрибутов отправляемой сущности, для которых есть мэппинг в данной системе. Если какой-либо атрибут имеет пустое значение, необходимо передать и его. MDM-система при обработке запроса на обновления изменит значения только пришедших атрибутов; то есть, если в MDM определены и другие атрибуты той же сущности, для которых не установлен мэппинг в данном приложении, эти атрибуты останутся неизменными.
- Если какой-либо атрибут имеет множественное значение согласно схеме, а в локальной системе используется только первое из них – приложение не может изменять значения этого атрибута. При отправке нужно поместить в соответствующий тег Attribute флаг Ignore="true". Если же в системе определен способ хранения множественных значений – нужно отправить их все. MDM-система при выполнении операции сначала удалит все имеющиеся значения этого атрибута, а затем создаст новые.
- Операция удаления сущностей выполняется путем присвоения им специального атрибута, имя которого определяется конкретной

моделью данных (например, archive). Получив запись с таким атрибутом, система должна удалить ее, или пометить на удаление. Аналогично, при удалении записи в локальной системе, необходимо отправить запрос UpdateObject, в котором удаленной (или помеченной на удаление) записи присвоен атрибут archive.

- Сформировав пакет, необходимо поместить его во внутреннюю очередь отправки, и после этого удалить обрабатываемую запись из журнала изменений.
- Очередь отправки также должна обрабатываться с определенной периодичностью. Обработка заключается в помещении каждого очередного сообщения в исходящую очередь, или вызове программного интерфейса MDM другим способом. Эта очередь необходима для того, чтобы исходящие сообщения не были потеряны при временном разрыве связи с MDM.

5. В ответ на каждый пакет UpdateObject поступает пакет OperationResults. Этот пакет необходимо обработать следующим образом:
   - Если в очередном теге OperationResult не содержится сообщения об ошибке – нужно найти элемент заданного типа по его LocalCode, и сохранить в его свойствах значение постоянного кода – mdm_code.
   - Если поступило сообщение об ошибке, необходимо поместить запись об этом в журнал событий, с указанием кода ошибки, а также вида и кода исходной операции. Желательно сохранить и сам исходный пакет – если адаптер ведет лог исходящих пакетов.
6. При поступлении ответа InvalidPackage, необходимо проинформировать администратора системы.