ООО «ТриниДата»

РУКОВОДСТВО ПО ИСПОЛЬЗОВАНИЮ

системы АрхиГраф.MDM

г. Екатеринбург, 2015-2019

Оглавление

1. Назначение АрхиГраф.MDM

Программа АрхиГраф.MDM предназначена 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.

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, 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

Редактирование информационной модели и выполнение административных функций осуществляется через интерфейс Onto.pro. Работа с редактором (создание и редактирование классов объектов, атрибутов, элементов классификаторов и справочников) описано в Руководстве пользователя Onto.pro. Принципы создания онтологической информационной модели и управления ей описаны в методическом пособии «Введение в онтологическое моделирование».

Настройка прав доступа пользователей и внешних информационных систем к мастер-данным осуществляется через плагины интерфейса Onto.pro. Процедура управления правами описана в Руководстве администратора АрхиГраф.MDM.

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

2.1. Запросы на внесение изменений в модель и мастер-данные

Права доступа пользователей к элементам модели настраиваются в административном интерфейсе Onto.pro. Для каждой группы пользователей может быть установлен один из следующих уровней доступа к каждому классу модели:

- нет доступа;
- только просмотр;
- редактирование с подтверждением;
- редактирование.

Права доступа применяются к дереву классов рекурсивно. То есть, если для надкласса установлен один из перечисленных уровней доступа, а для подкласса уровень доступа для той же группы не определен, то применяется уровень доступа к надклассу.

На рис. 1 показан вид интерфейса настройки прав доступа в административном интерфейсе Onto.pro:

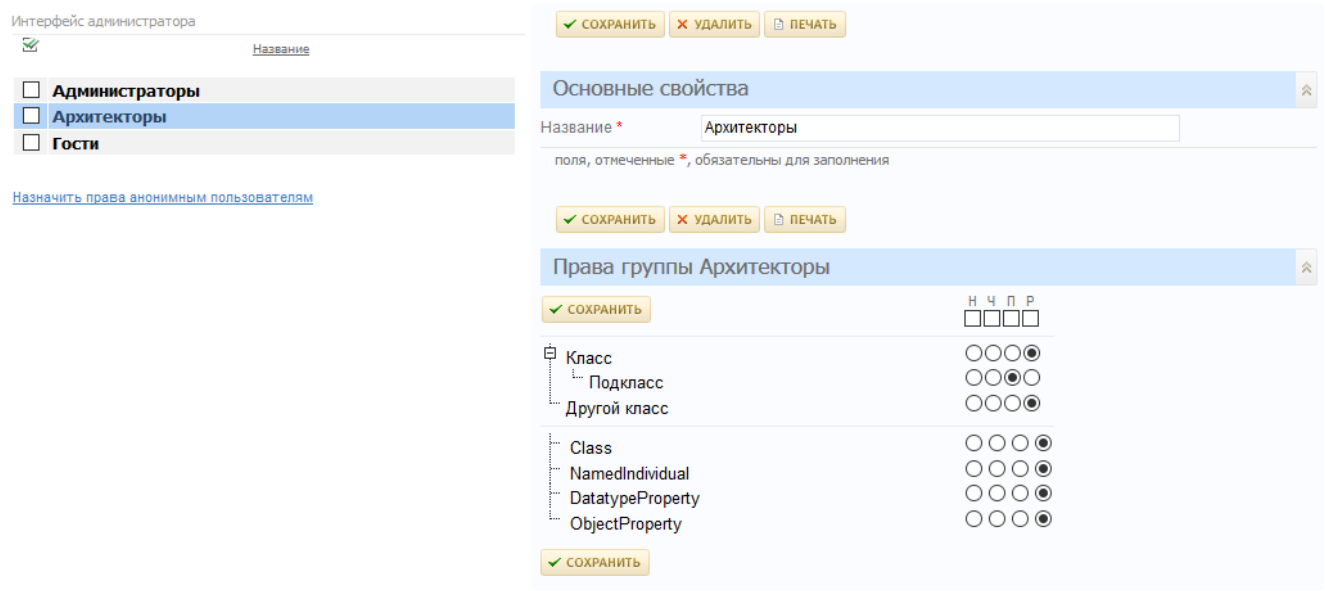

Рис. 1. Интерфейс настройки прав доступа групп пользователей к классам модели

Пользователь, имеющий к элементу модели права доступа «Редактирование с подтверждением», в форме редактирования элемента вместо кнопок «Создать», «Сохранить», «Удалить» видит кнопку «Создать запрос на изменение». При нажатии на эту кнопку функции редактирования разблокируются.

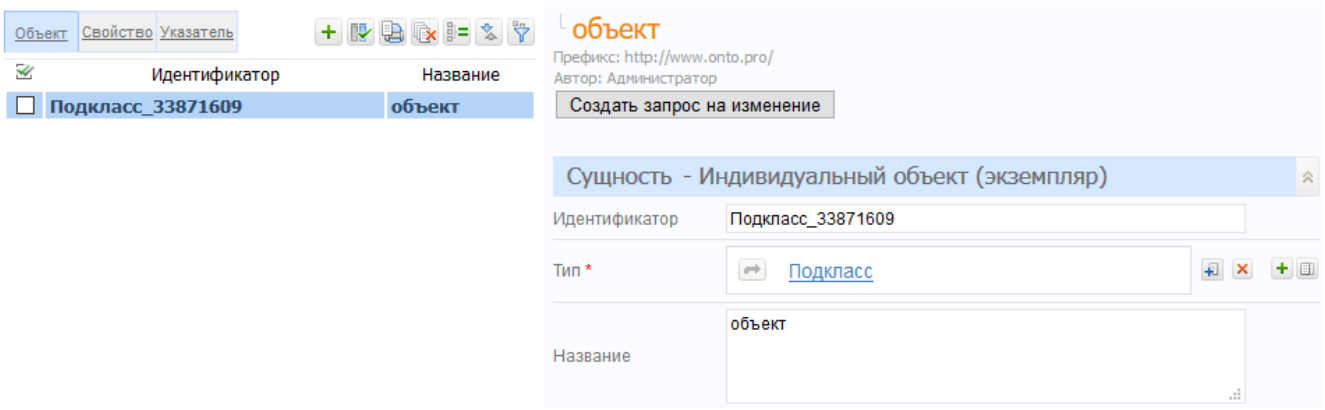

Рис. 2. Вид формы редактирования элемента модели для пользователя с правами доступа «Редактирование с подтверждением»

После того, как пользователь создаст, сохранит или удалит элемент модели, доступ к которому получил путем нажатия кнопки «Создать запрос на изменение», изменения не вносятся в модель, а сохраняются в журнале запросов на изменение.

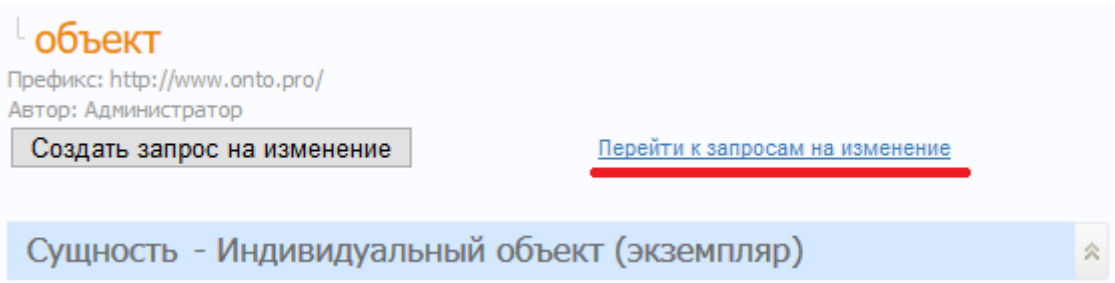

Рис 3. Вид формы редактирования элемента, для которого создан запрос на изменения

Ссылка на запрос на изменения отображается в правой верхней части формы на редактирование. Ссылку видят пользователи с правами «редактирование с подтверждением» и «редактирование».

Пользователь с уровнем прав «редактирование», перейдя по данной ссылке, попадает в журнал запросов на изменение, отфильтрованный по выбранному элементу. В этот журнал также можно попасть через пункт «Запросы на изменение» главного меню Onto.pro, тогда в нем будут отображаться все поступившие запросы.

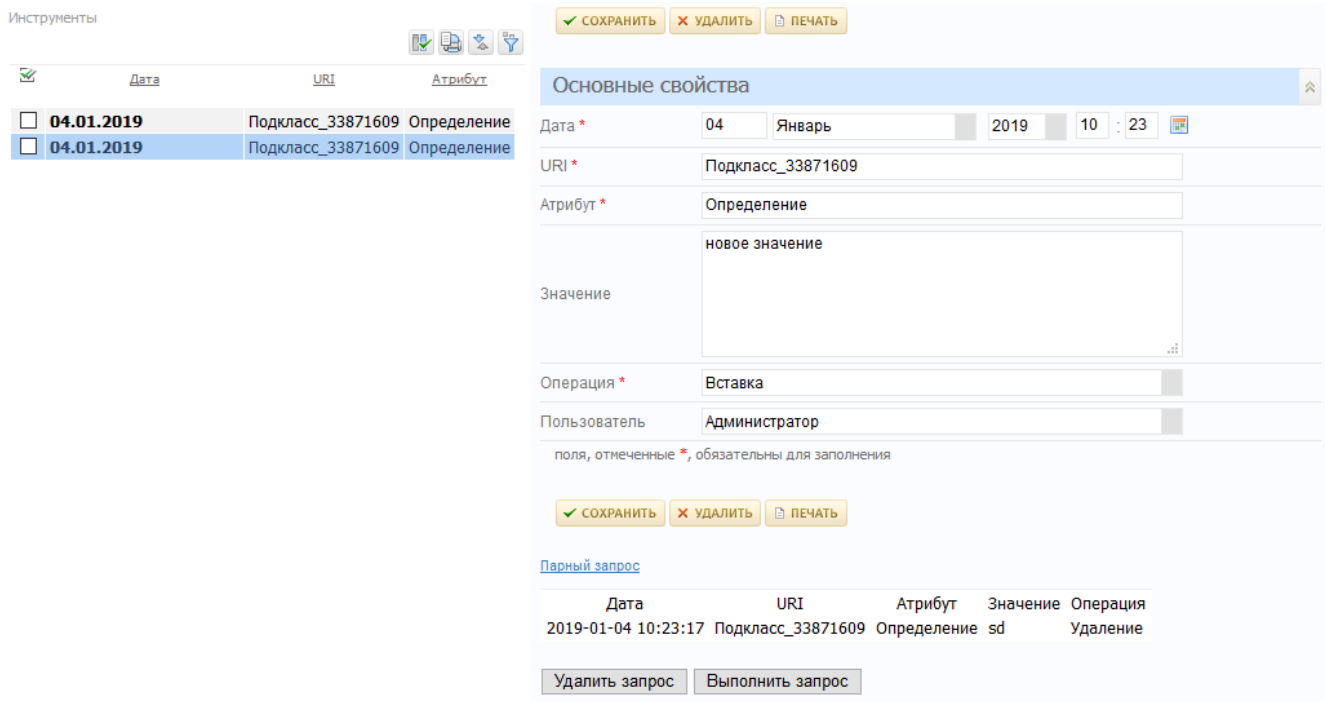

Рис. 4. Просмотр записи журнала запросов на изменения

В журнале отображается отдельная запись для каждого созданного/удаленного значения атрибута элемента (или созданного/удаленного элемента). Если пользователь изменял существующее значение атрибута, то этой операции будет соответствовать две записи в журнале: одна – на удаление старого значения, вторая – на вставку нового.

В форме выбранного запроса отображается ссылка на парный запрос на создание/удаление значения данного атрибута в рамках той же операции. Ели пользователь изменял значения нескольких атрибутов одновременно, то будет также показана таблица с запросами на изменение значений других атрибутов.

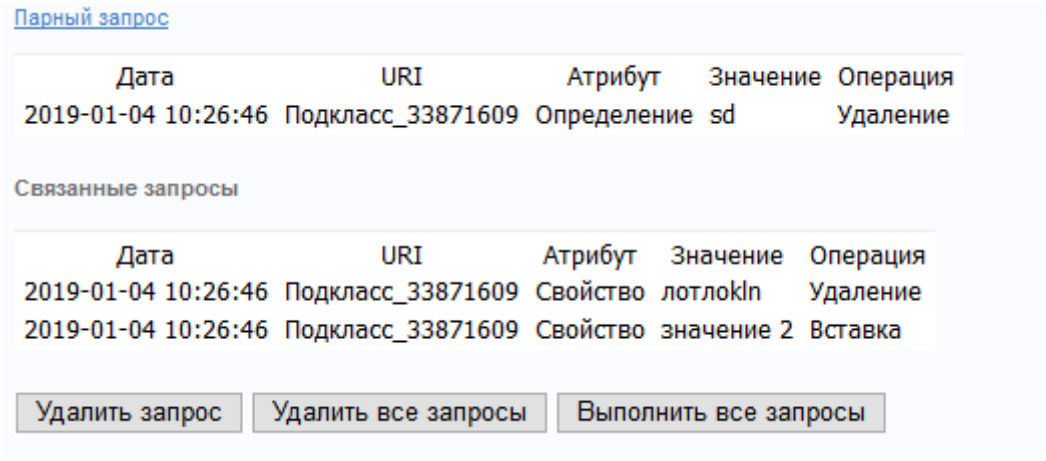

Рис. 5. Просмотр связанных запросов

Пользователь может подтвердить (выполнить) отдельный запрос или группу запросов, в этом случае изменения немедленно вступят в силу, а в истории изменения свойств элемента модели появятся соответствующие записи. Пользователь может удалить запрос или группу запросов, в этом случае изменения в модель внесены не будут.

2.2. Поиск и исключение дубликатов

Поиск и исключение дубликатов выполняется средствами системы АрхиГраф.СУЗ. Процесс выполняется следующим образом:

1. Создаются правила определения дубликатов, применимые к объектам разных классов. Например, контрагенты являются вероятными дубликатами, если у них совпадает ИНН.
2. Правила определения дубликатов автоматически или вручную применяются к содержимому MDM. В ходе применения правил записи-дубликаты получают специальную пометку.
3. При помощи интерфейса работы с дубликатами ответственный пользователь просматривает помеченные записи и принимает решение по каждой паре дубликатов. Решение может заключаться в слиянии записей, или в признании записей самостоятельными.

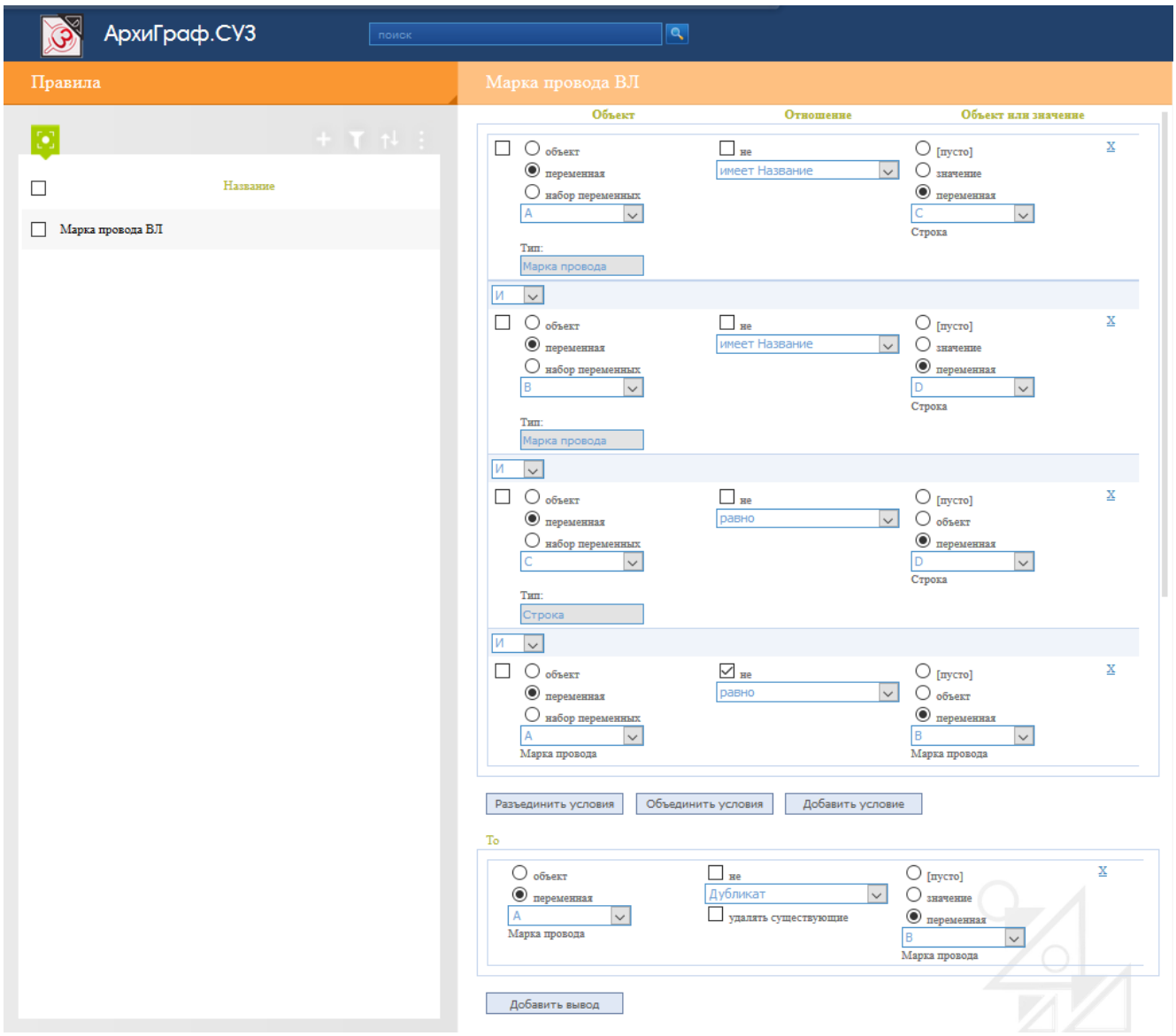

Рис. 6. Настройка правила определения дубликатов

Правила определения дубликатов создаются при помощи конструктора правил, функциональность которого описана в Руководстве пользователя АрхиГраф.СУЗ. Правило представляет собой набор логических условий, каждое из которых сравнивает определенные свойства двух записей, обозначаемых переменными A и B. Вывод, который применяется в случае выполнения условий правила, состоит в том, что объекты A и B являются вероятными дубликатами.

После того, как правила настроены, необходимо запустить их применение (запуск может выполняться и автоматически, по расписанию). На специальной странице АрхиГраф.СУЗ отображается процесс и результат применения правил.

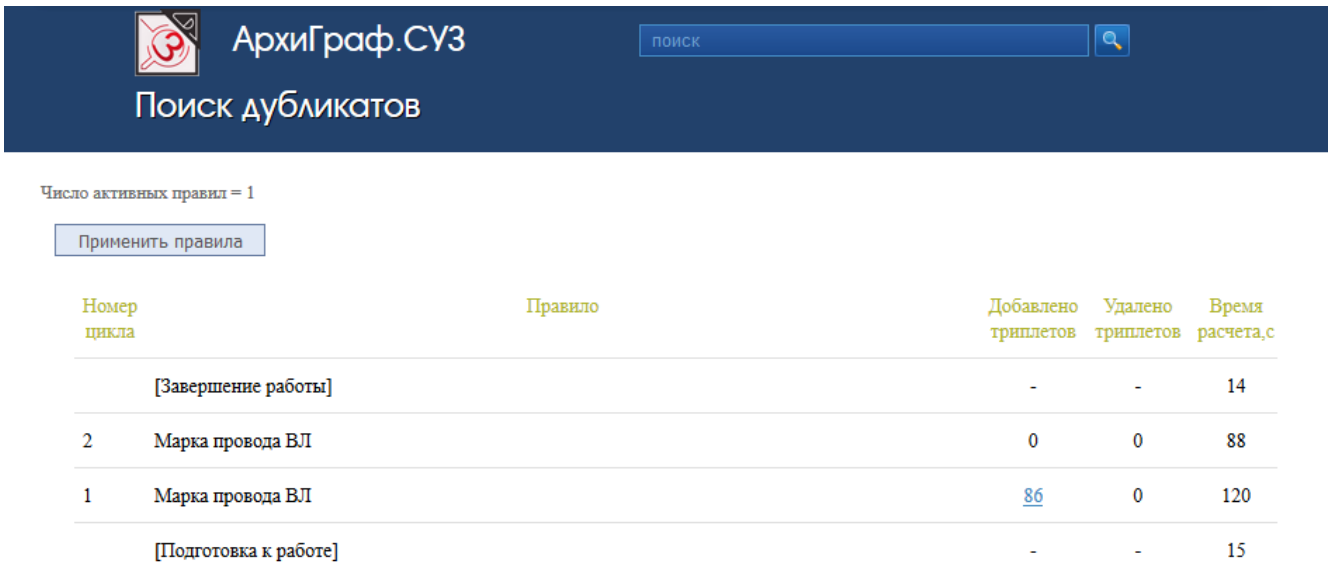

Рис. 7. Страница результатов применения правил поиска дубликатов

Еще одна страница, «Просмотр дубликатов», отображает список вероятных дубликатов элементов модели, обнаруженных каждым правилом.

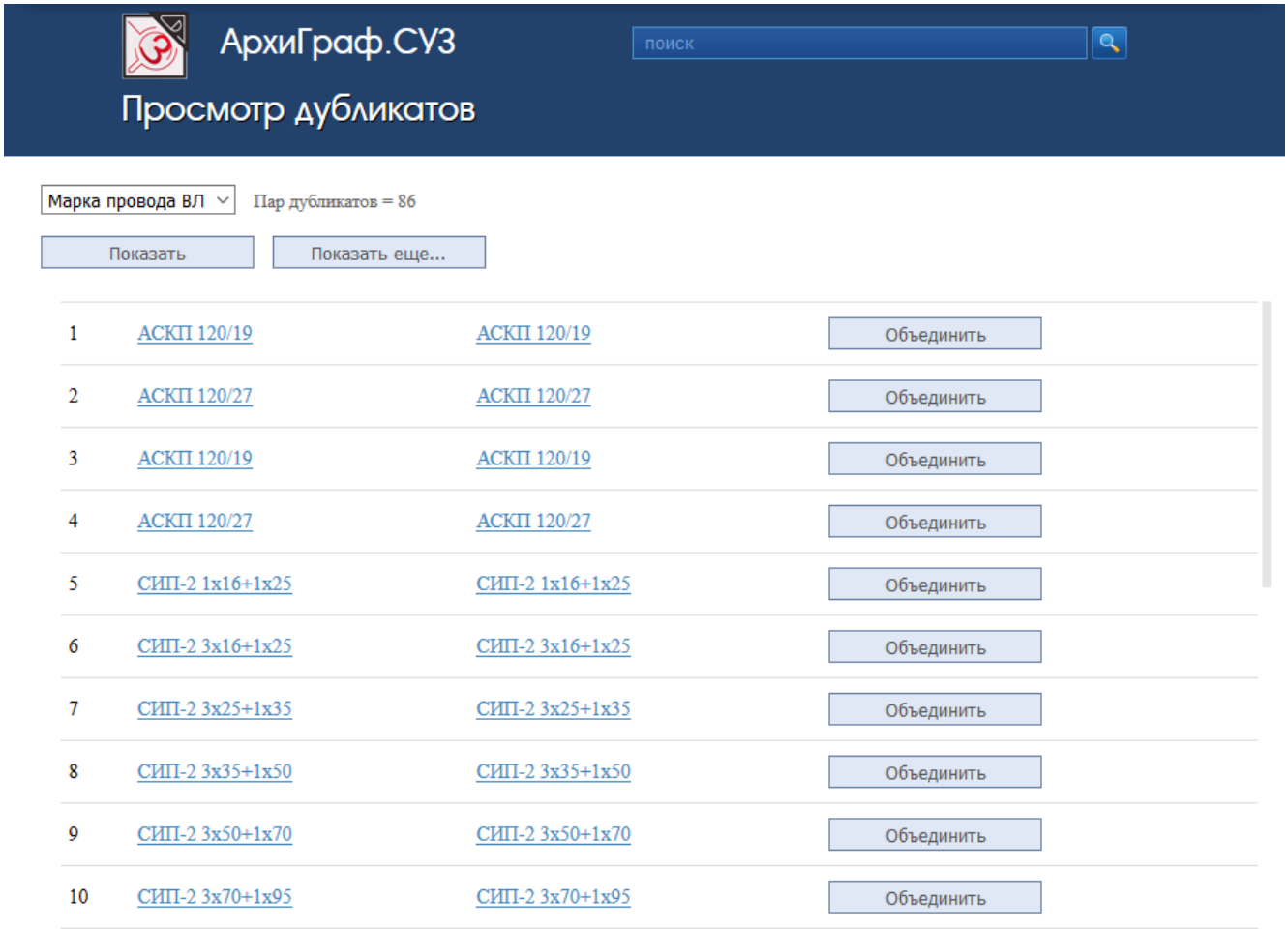

Рис 8. Список обнаруженных дубликатов

Ответственный пользователь последовательно просматривает список, при этом он имеет возможность перейти к просмотру свойств каждого из найденных возможных дубликатов. Нажатие кнопки «Объединить» открывает интерфейс работы с выбранной парой записей.

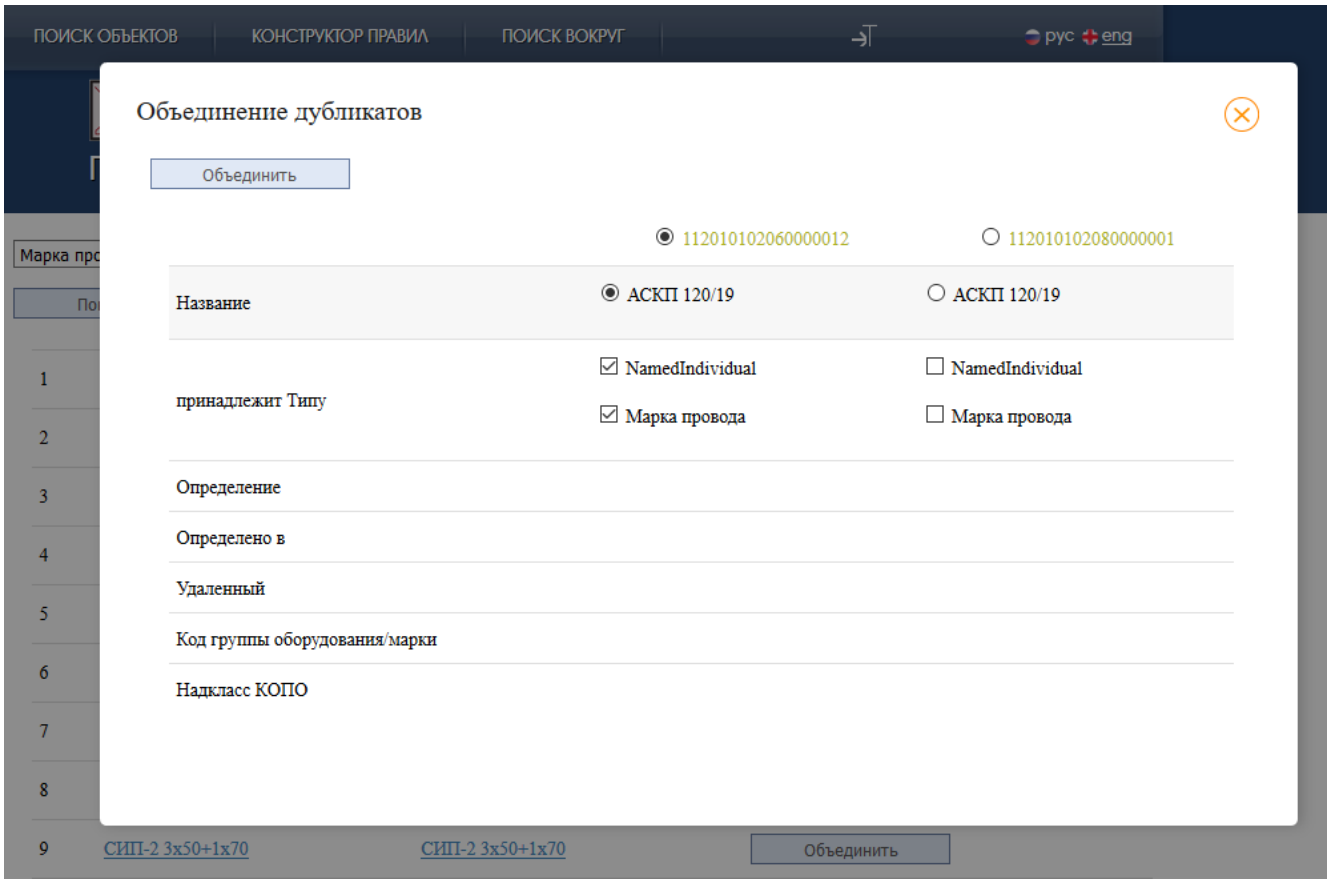

Рис. 9. Работа с выбранной парой дубликатов

В диалоговом окне отображается таблица с двумя столбцами, каждый из которых содержит свойства (принадлежность классам, набор атрибутов) одной из сравниваемых записей. Пользователь имеет возможность:

- Для атрибутов, которые могут иметь только одно значение – отметить переключателем то значение, которое получит объединенная запись.
- Для атрибутов, которые могут иметь несколько значений – можно отметить переключателями любое количество значений. Объединенная запись получит все отмеченные значения.
- Для набора классов, к которым относится запись – можно отметить переключателями любое количество классов. Объединенная запись получит все отмеченные значения атрибутов.

После подтверждения операции объединения будет создана объединенная запись, которая будет принадлежать всем отмеченным классам и получит все отмеченные значения атрибутов. Переключатель около идентификатора записи в верхней части окна показывает, какая из записей будет выбрана основной. Все

ссылки на вторую (присоединяемую) запись будут автоматически преобразованы в ссылки на основную запись.

3. Синхронизация модели и мастер-данных с приложениями

3.1. 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),
- SOAP-сервисов, или
- прямых HTTP-запросов.

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 ActiveMQ. Преимущество данной конфигурации состоит в том, что MDM-система,

как и все остальные приложения, обрабатывает сообщения в асинхронном режиме. Благодаря этому не может возникнуть ситуации перегрузки того или иного сервера или приложения, повышается надежность обмена. Кроме того, такая схема облегчает кластеризацию MDM-системы.

Маршрутизацию сообщений соответствующим получателям обеспечивает корпоративная сервисная шина (ESB), или брокер сообщений (Message Broker). Эти компоненты отвечают за перемещение сообщений из исходящей очереди отправителя во входящие очереди получателей. Для выбора подходящего маршрута они используют информацию об отправителе, и анализируют 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:

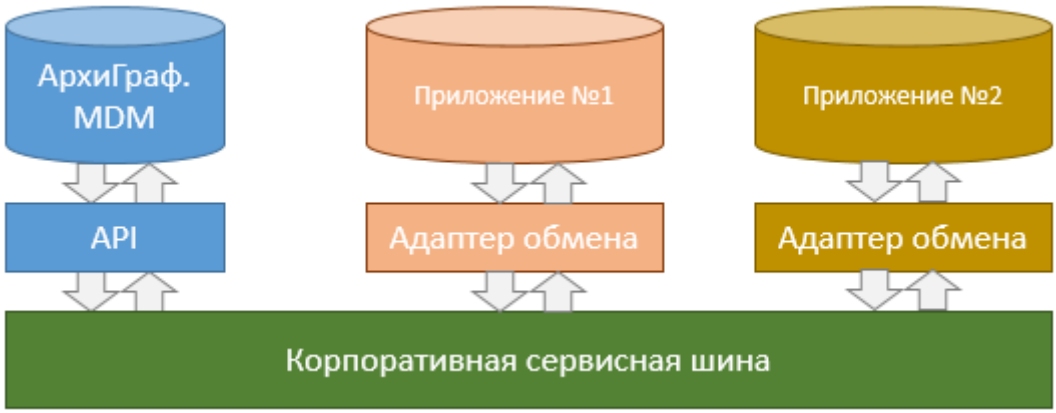

Рис. 10. Архитектура обмена с АрхиГраф.MDM

Последовательность прохождения каждого сообщения через программные компоненты архитектуры обмена выглядит так:

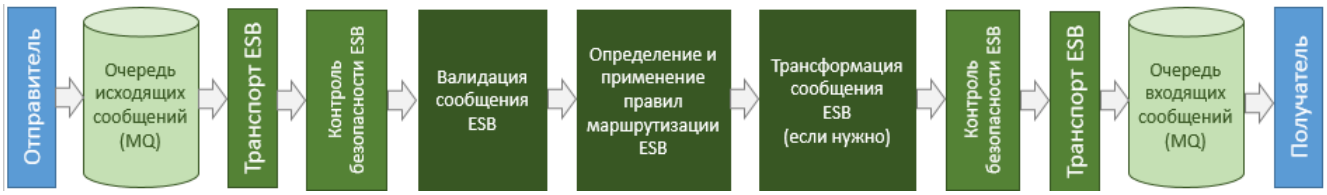

Рис. 11. Последовательность обработки сообщения программными компонентами при передаче между очередями

4. 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 уложить содержимое пакета DataModel 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 имена могут иметь и стандартные значения – 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 EntityRequest 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 Package 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 ChangesRequest 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 ChangesRequest, 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"?>
<ChangesRequest 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"?>
<ChangesRequest 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 ChangesRequest, 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 ChangesRequest 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.