Платформа ActiveFrame 5

Описание архитектуры

Software Architecture Document

Утверждено:

. . . . . .

Согласовано:

. . . . . .

Содержание

1 Введение

1.1 Цель документа

В документе описывается архитектура платформы ActiveFrame 5 (далее AF5), на основе которой строятся новые версии системы электронного документооборота CompanyMedia, приводятся принципы разбиения на подсистемы и службы.

1.2 Целевая аудитория документа

Документ предназначен для системных архитекторов, системных аналитиков и разработчиков, участвующих в разработке архитектуры и дизайна системы CompanyMedia и платформы AF5, а также в реализации заказных бизнес-решений в предметной области электронного документооборота. Документ будет также полезен менеджерам проектов и другим заинтересованным лицам в проектах развития типового ПО Компании «ИнтерТраст» и проектах внедрения СЭД CompanyMedia, выполняемых компанией и ее партнерами.

1.3 Материалы

Документы, положенные в основу настоящего документа или используемые совместно с ним (дополняющие его):

1. CompanyMedia. Архитектура, выполняемые функции, принципы управления
2. ActiveFrame. Архитектура, выполняемые функции, принципы управления
3. Архитектура GUI – подсистемы конфигурируемых пользовательских интерфейсов
4. Интеграция CMJ – сервера с платформой AF5

1.4 Определения и сокращения

ActiveFrame 5 (AF5) - документоориентированная платформа хранения и обработки данных, основанная на Java Enterprise Edition, использует реляционную СУБД для хранения атрибутивных данных и репозиторий контента на основе сетевой файловой системы, включает конфигурируемый Web-клиент.

Информационная система на платформе AF5 (ИС или Система) – программный комплекс, состоящий из платформы и приложений, решающих различные прикладные задачи.

Система электронного документооборота (СЭД) CompanyMedia (CM) – прикладное решение на платформе AF5, реализующее функции автоматизации электронного документооборота.

Платформа – основной компонент системы, реализующий её базовые функции: управление конфигурацией системы, хранение служебных и доменных объектов, поиск по ним, управление процессами, разграничение доступа к объектам.

Ядро – серверная часть платформы.

Функциональный модуль (ФМ) – компонент системы, реализующий более или менее автономный набор прикладных функций. Например: базовый документооборот, организационно-штатная структура, электронная подпись, межведомственный документооборот и т.п. Модули используют сервисы, предоставляемые платформой, и не имеют прямого доступа к хранилищу объектов. Они также обычно включают типовую конфигурацию, предназначенную для облегчения настройки экземпляра системы у конкретного заказчика.

CMJ - подсистема CM, включающая прикладные модули CompanyMedia и REST-API для Web-клиента CM;

Слой сопряжения CMJ с AF5 «Сочи» – бизнес-модуль, обеспечивающий взаимодействие между клиентом CM4, с одной стороны, и ядром платформы. Поскольку CMJ рассчитан на определённую структуру бизнес-объектов, слой сопряжения содержит необходимую конфигурацию этих объектов, которая не должна изменяться при настройке экземпляров системы.

Конфигурация – набор текстовых (преимущественно XML) файлов, задающих правила работы конкретного экземпляра системы. Конфигурация включает настройки следующих объектов: доменные объекты; регламентированные процессы; группы пользователей; правила доступа; правила формирования интерфейса пользователя (папки, списки, формы, действия).

Графический интерфейс пользователя (ГИП) – пользовательский интерфейс, в котором элементы интерфейса (меню, кнопки, значки, списки и т. п.), представленные пользователю на дисплее, исполнены в виде графических изображений.

Web-клиент – компонент системы, реализующий её ГИП посредством Web-браузера. Содержит графические элементы, позволяющие отображать любые бизнес-объекты, их наборы (списки) и структуры (деревья), управлять исполнением бизнес-процессов, а также управлять пользователями и их ролями в системе. Клиент не зависит от функциональных модулей, которые, однако, могут при необходимости добавлять в него собственные графические элементы, и взаимодействует непосредственно с ядром системы.

Автоматизированное рабочее место (АРМ) – клиентское ПО системы, предназначенное для автоматизации деятельности определенного вида.

Доменный объект (ДО или ДОП) – структура данных, хранящая значения фиксированного набора именованных полей. Каждое поле объекта может хранить единственное значение. Множественные значения могут быть реализованы с использованием отдельных дочерних доменных объектов. Доменный объект имеет идентификатор и является объектом назначения прав доступа.

Статус – специальное поле карточкидоменного объекта. Используется для управления применимостью к ней нему правил доступа, форм, действий и т.п. Изменение статуса карточки ДО является специальной операцией и не может быть выполнено путём сохранения карточкиобъекта.

Регламентированный процесс – набор правил выполнения какого-либо сложного (составного), как правило, протяжённого во времени действия в рамках системы. Описывается в системе на языке BPMN 2.0.

Пользователь – лицо, работающее с системой. Внутри системы пользователи представляются доменными объектами.

Правило доступа – элемент конфигурации, определяющий возможность выполнения пользователем или группой пользователей какой-либо операции (создание, чтение, изменение, удаление,запуск процесса и т.п.) над объектом доступа (доменным объектом, действием,).

Группа пользователей – конфигурационная единица, предназначенная для разграничения прав доступа. Состав пользователей, входящих в группу, может либо назначаться вручную администратором (статическая группа), либо вычисляться системой по какому-либо алгоритму (динамическая группа). Динамические группы могут иметь зависимость от доменных объектов. В таком случае система создаёт по экземпляру динамической группы на каждый доменный объект заданного типа, а пользователи в эту группу включаются исходя из содержимого данного объекта.

Контекстная роль – набор групп пользователей, имеющих какое-либо отношение к доменному объекту. Предназначена для назначения прав доступа к этому или связанным объектам.

Коллекция –набор доменных объектов, динамически формируемый по правилам, задаваемым в конфигурации.

Форма – набор элементов ГИП, отображающих содержимое карточки – доменного объекта или их связанного набора. Набор визуальных элементов и их взаимное расположение, а также информация о соответствии элемента полю карточки/объекта хранится в конфигурации. При отображении элементы заполняются данными из конкретной карточки/ объекта.

Действие – процедура в системе, выполнение которой может быть инициировано пользователем через графический интерфейс (обычно – нажатием кнопки). Является объектом назначения прав доступа.

1.5 Архитектура системы CompanyMedia на платформе AF5 (в целом)

В основу создания данной версии СЭД CompanyMedia легли идеи о совмещении всех положительных сторон программного обеспечения, разработанного ранее (в рамках CompanyMedia 4.x) – CMJ-сервер и CMJ-WEB и перспективной разработки AF5, что в совокупности позволит получить систему, способную работать как по-старому (с использованием IBM Lotus Domino в качестве платформы), так и по-новому – как показано на схеме ниже. При этом в CompanyMedia, начиная с версии 5 платформа Lotus Domino заменена на новую платформу AF5 на основе реляционной базы данных, реализующую все необходимые сервисы для обработки данных.

1.5.1 Состав системы

Основные типы клиентских рабочих мест – web- и мобильное АРМ для всех пользователей, а также АРМ администратора взаимодействуют с сервером системы по интернет протоколу (HTTP/S).

Сервер системы документооборота реализован на платформе Java EE и интегрирован с платформой AF5.

Документоориентированная платформа хранения и обработки данных AF5, также основанная на Java Enterprise Edition, использует реляционную СУБД для хранения атрибутивных данных и репозиторий контента на основе сетевой файловой системы или подключаемой ECM-системы одного из широко известных производителей.

1.5.2 Архитектура сервера

Архитектура сервера CompanyMedia имеет несколько уровней (Рис.1):

- Уровень представления информации (интерфейсов пользователей), обеспечивающий взаимодействие web- и мобильных клиентов с сервером.
- Уровень предметной области, на котором в системе определяются:
    - бизнес-объекты: документы, поручения, задачи, кейсы, контакты и др.
    - жизненные циклы объектов, роли, события, операции и др.
    - коллекции: журналы, папки, представления, результаты поиска, персональные коллекции документов (ПКД) и др.
    - функциональные сервисы СЭД: регистрация, контроль, формирование дел (ФД), задачи, уведомления, обсуждения и др.
    - процессы обработки документов: подготовка, согласование и др.
    - кейсы
- Уровень системных сервисов:
    - Механизмы исполнения процессов с предопределенным порядком действий (BPMN 2.0) и без заранее заданного порядка (ДОУ, ACM)
    - Поиск
    - Центр отчетов
    - Управление правами доступа и др.
- Платформа защищенного хранения и обработки данных AF5 в свою очередь построена по «классической» трехуровневой архитектуре и включает:
    - Web-GUI - Собственный уровень представления информации (интерфейсов пользователей), на котором основано АРМ администратора, и могут быть реализованы дополнительные специализированные АРМы.
    - «Ядро» платформы, реализующее её основные сервисы и предоставляющее публичный API для работы с ними другим подсистемам (в частности серверу CompanyMedia) и интерфейсному слою платформы.

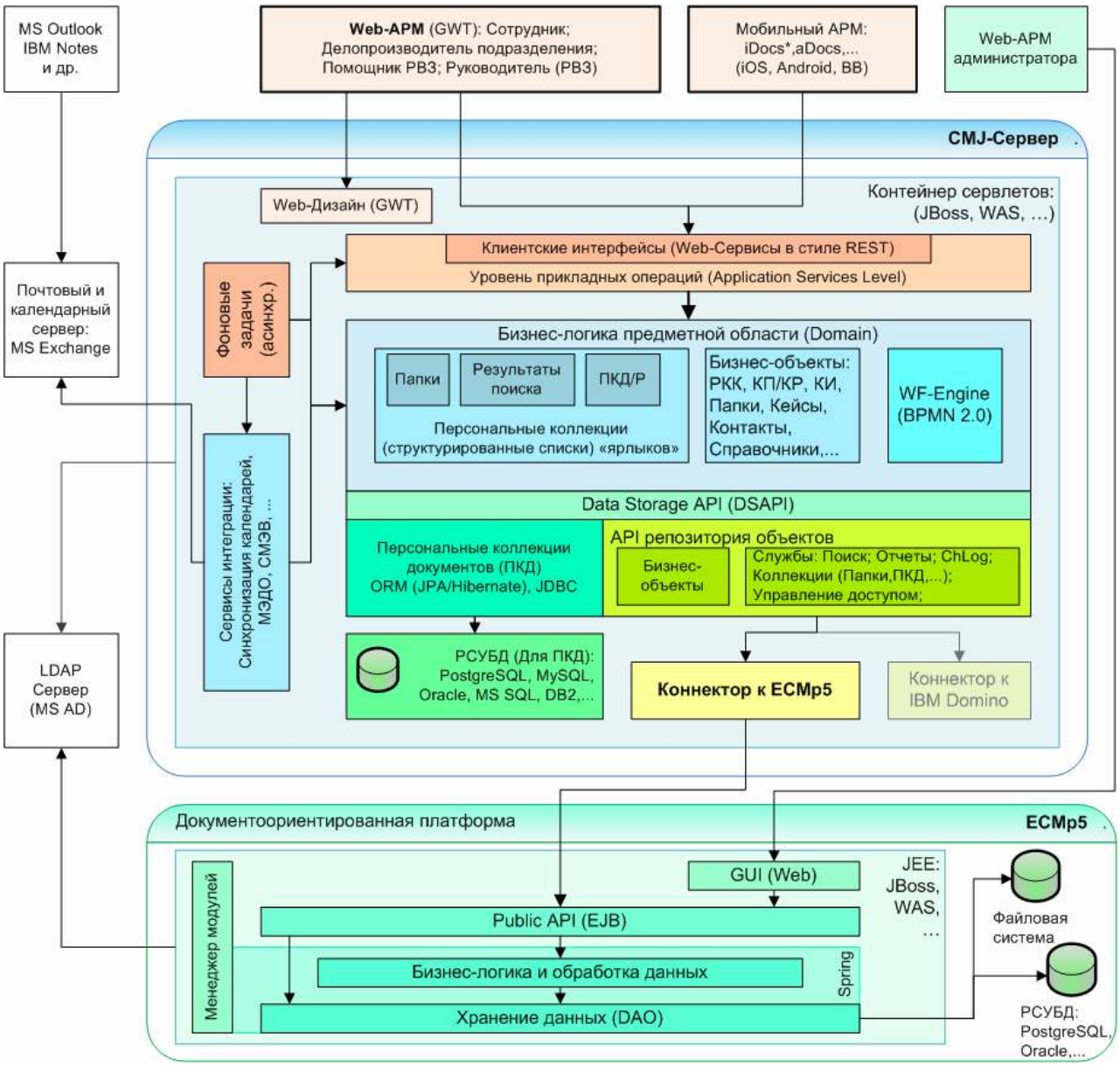


Рис.1 Архитектура СЭД ***CompanyMedia*** в целом

1.6 Архитектура ИС на платформе AF5

1.6.1 Состав системы

Информационная система на платформе AF5 представляет собой программный комплекс, состоящий из платформы и приложений – функциональных модулей, решающих различные прикладные задачи.

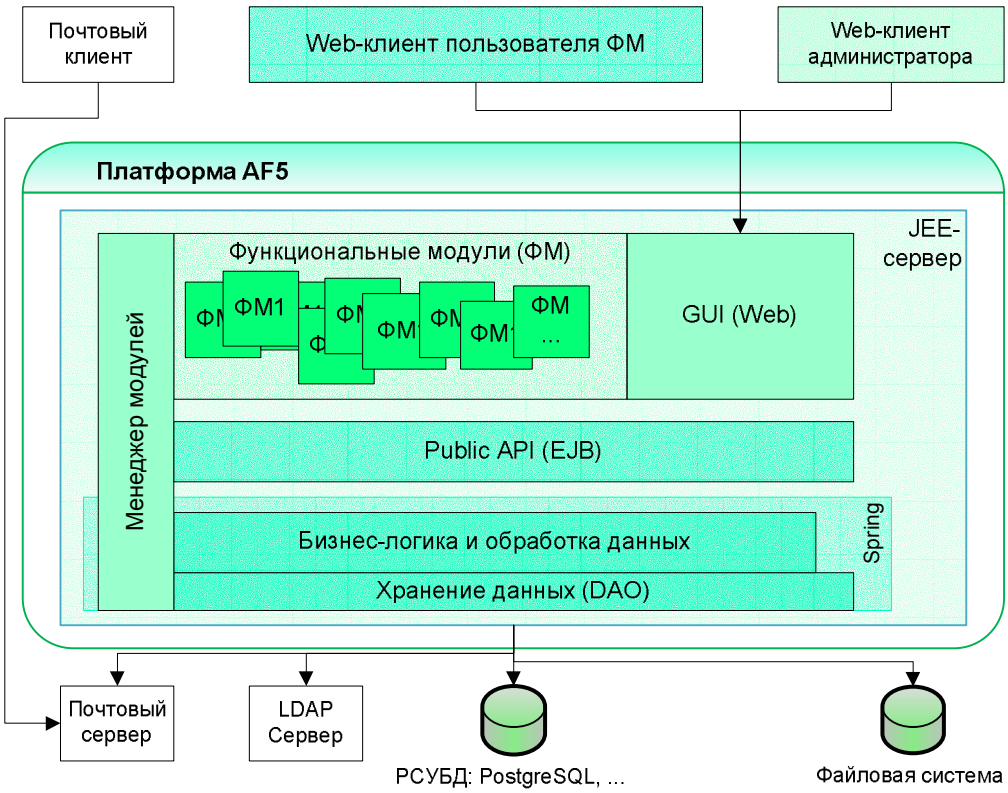
Документоориентированная платформа хранения и обработки данных AF5, основанная на Java Enterprise Edition, использует реляционную СУБД для хранения атрибутивных данных и репозиторий контента на основе сетевой файловой системы.

Функциональные модули – компоненты системы, реализующие как самостоятельные, так и «связанные» прикладные функции.

Основные тип клиентского рабочего места – web-АРМ для всех пользователей, а также АРМ администратора – взаимодействуют с сервером системы по интернет-протоколу (HTTP/S).

1.6.2 Архитектура системы

Архитектура АИС на платформе AF5 имеет несколько уровней:



- Уровень представления информации (интерфейсов пользователей), обеспечивающий взаимодействие web-клиентов с сервером.
- Уровень предметной области - функциональные модули, решающие прикладные задачи системы.
- Уровень системных сервисов (платформа AF5), реализующий её базовые функции: управление конфигурацией системы, хранение служебных и функциональных объектов, поиск по ним, управление процессами, разграничение доступа к объектам.

1.6.3 Функциональные модули

Функциональный модуль – компонент системы, реализующий более или менее автономный набор прикладных функций.

Примерами функциональных модулей могут быть: базовый документооборот («делопроизводство»), организационно-штатная структура, электронная подпись, межведомственный документооборот, и т.п. Функциональные модули используют сервисы, предоставляемые ядром, и не имеют прямого доступа к хранилищу объектов. Они также обычно включают типовую конфигурацию, предназначенную для облегчения настройки экземпляра системы у конкретного заказчика.

Сам функциональный модуль, как правило, содержит компоненты, охватывающие все уровни системы, см. Рис.3.

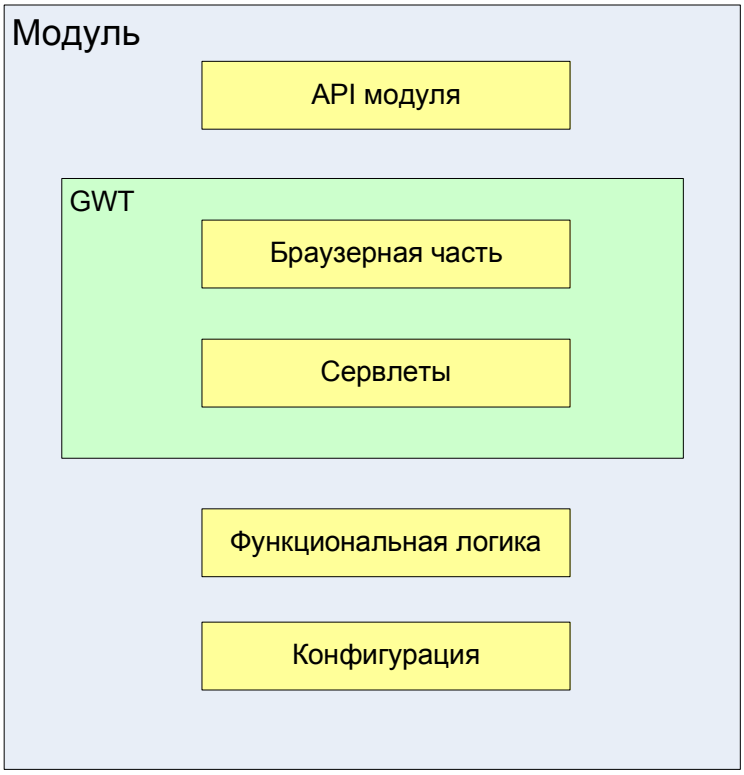


Рис.3 Архитектура типового функционального модуля

Так, в конфигурации практически всегда присутствует описание дополнительных доменных объектов, отражающих сущности предметной области в процессе автоматизации, и связанные с ними правила расчета матриц доступа для подсистемы управления доступом.

Там же задаются правила настроек индексации для выполнения функций расширенного поиска и форм задания критериев отбора для работы с ним.

Регламенты для автоматизации процессов предметной области (шаблоны в нотации BPMN 2.0 для исполнительного механизма BPM на основе Activiti) также входят в состав конфигурации функционального модуля.

Решение по автоматизации предполагает как полностью автоматизированные участки обработки, где формализованные алгоритмы реализованы в виде программных компонентов (java) и относятся к слою функциональной логики, так и участки предполагающие интерактив – некоторые действия пользователя в рамках интерфейса системы. Для последних предназначены дополнительные разделы конфигурации, описывающие интерактивные формы, списки (представления), правила работы элементов управления графического интерфейса пользователя (ГИП) и т.д.

В слое конфигурации также могут указываться шаблоны отчетов. Они также могут подгружаться в систему динамически через ГИП администратора.

Помимо возможности использования стандартных виджетов (заранее разработанных элементов ГИП) функциональный модуль может содержать и расширенный состав специально разработанных элементов управления для решения задач навигации, форматирования, вывода и представления данных и т.д. При этом любой виджет содержит как серверную часть, отвечающую за обработку массива данных (программные компоненты, реализованные отдельными классами java и взаимодействующие с клиентами – как правило, ГИП – посредством принципа запрос-ответ), так и клиентскую, реализованную на java-script и интерпретируемую уже самим браузером при формировании страницы (окна ГИП системы).

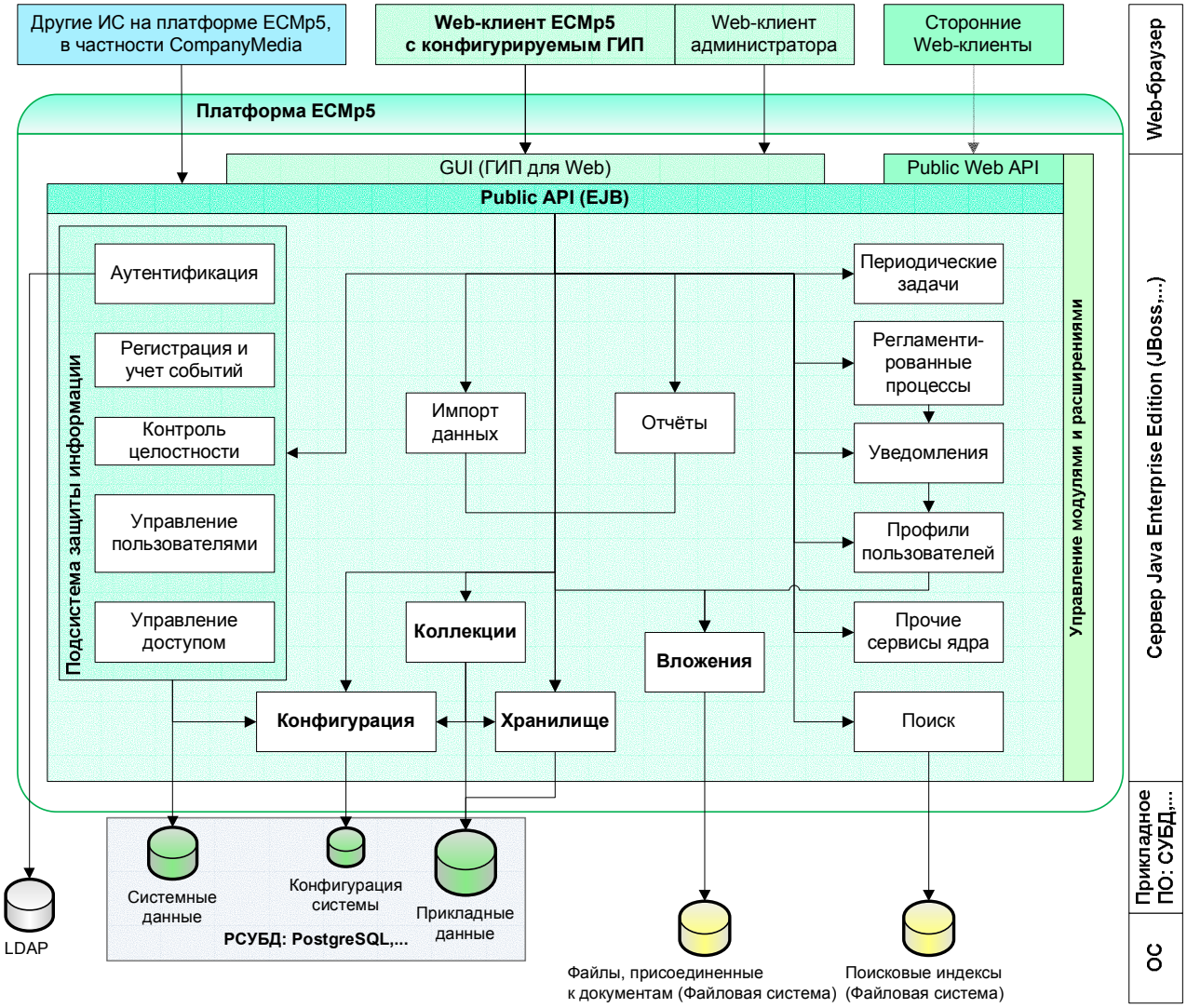
Возможность создания программного интерфейса взаимодействия с модулем делает применение функциональных модулей универсальным механизмом для построения крупных информационных систем.

1.7 Архитектура платформы AF5

Платформа защищенного хранения и обработки данных AF5 включает:

- Собственный уровень представления информации (интерфейсов пользователей), на котором основано АРМ администратора, и могут быть реализованы дополнительные специализированные АРМы, включает
    - Составное приложение «список/редактор системных объектов»
    - Сервис конфигурируемых списков
    - Сервис конфигурируемых форм
    - GUI компоненты отображения конфигурируемых форм
    - Основные виджеты, отображающие атрибуты объектов системы
- Уровень "ядра" платформы, реализующий её основные сервисы и предоставляющий публичный API для работы с ними другим подсистемам (в частности серверу CompanyMedia) и интерфейсному слою платформы:
    - Сервис конфигурации – обеспечивает хранение и контроль целостности расширяемой конфигурации системы, а также доступ других компонентов как к конфигурации в целом, так и к отдельным её компонентам только на чтение (без возможности изменения).
    - Сервис хранения доменных объектов – выполняет основные операции (чтение, сохранение, удаление) над конфигурируемыми структурами данных в системе – доменными объектами.
    - Подсистема хранения вложений – обеспечивает хранение неструктурированных данных – файлов (вложений), а также доступ к ним с возможностью подключения адаптеров к различным сервисам хранения файлов (файловая система).
    - Сервис точек расширения – предоставляет возможность расширения функциональности системы путём внедрения дополнительного кода в основные операции системы.
    - Сервис коллекций, предоставляющий конфигурируемые выборки (списки) по множеству доменных объектов;
    - Подсистема поиска – выполняет высокоскоростной поиск текста и другой информации в больших объёмах структурированных и неструктурированных данных.
    - Подсистема уведомлений – выполняет информирование пользователей (с помощью e-mail, всплывающих сообщений в ГИП, e-mail и другими способами) о различных событиях в системе.
    - Подсистема регламентов (регламентированных процессов) – управляет выполнением регламентированных процессов обработки данных, задаваемых на языке BPMN.
    - Подсистема планирования и выполнения периодических задач.
    - Подсистема управления доступом – обеспечивает авторизацию пользователей для выполнения любых операций в системе.

- Сервис регистрации и учёта событий – сохраняет в системе информацию о событиях, связанных с действиями пользователей и администраторов, а также обеспечивает доступ к этим данным.
- Сервис профилей пользователей – обеспечивает хранение индивидуальных настроек пользователей, а также доступ к ним других компонентов системы.
- Административные сервисы: управление конфигурацией (обновление и проверка корректности конфигурации), управление данными о пользователях (создание, удаление), назначение ролей.
- Подсистема отчетов - выполняет формирование отчётов по доменным объектам с использованием конфигурируемых шаблонов.
- Сервис импорта/экспорта объектов системы из/в xml.



Архитектура платформы **AF5**

2 Архитектурные цели и ограничения

2.1 Концептуальные требования к архитектуре

Система разбивается на подсистемы согласно требуемым функциям. Каждая подсистема выполняет обработку замкнутого набора событий.

Подсистемы разбиваются на компоненты. Каждый компонент реализует один или несколько сервисов.

Взаимодействие между отдельными компонентами и модулями осуществляется только через общую системную среду. Прямые вызовы между подсистемами отсутствуют.

Разрабатываемая Система должна удовлетворять основным принципам построения современных информационных систем:

1. **Защищенность**. Отдельные компоненты системы имеют минимально необходимый доступ к защищенной информации.
2. **Масштабируемость.** Способность системы к росту числа обрабатываемых запросов определяется только характеристиками аппаратного обеспечения, на котором функционирует данная система, и каналов передачи данных, используемых для обмена информацией.
3. **Распределенность**. Система позволяет распределять информационные ресурсы и процессы их обработки по нескольким серверам, функционирующим в её составе.
4. **Модульность**. В связи с разнородностью решаемых задач Система состоит из отдельных интегрируемых модулей, разделенных на несколько взаимодействующих слоев.
5. **Открытость**. Система обладает открытым программным API интерфейсом. Подсистемы предоставляют сервисы для обращения сторонних систем к своим функциям.
6. **Гибкость**. Данный принцип определяет возможность добавления новых функций в систему без нарушения её функционирования.
7. **Комплексность**. При декомпозиции должны быть установлены такие связи между структурными элементами системы (подсистемами, компонентами подсистем и комплексами ПО), которые обеспечивают цельность системы и возможность его сопряжения с другими подсистемами.
8. **Соответствие промышленным стандартам.** Под этим понимается следующие принципы, применяемые при разработке системы:
   - использование для реализации протоколов и алгоритмов обработки различных стандартов. Таким образом, при наличии промышленного стандарта не должна использоваться самостоятельная разработка;
   - использование для реализации функций, требуемых для работы системы продуктов третьих фирм, если они соответствуют требованиям проекта;
   - использование для решения задач масштабирования и обеспечения отказоустойчивости прежде всего свойств и возможностей промышленных продуктов третьих фирм, если они соответствуют требованиям проекта.

3 Архитектурные механизмы и пакеты

3.1 Обзор

В данном разделе приводится описание основных механизмов, из которых состоит система.

(подчеркнуты еще не описанные пункты)

Конфигурируемые доменные объекты (ДО)

Иерархии доменных объектов

Подсистема управления доступом

Работа с вложениями

Подсистема поиска

Взаимодействие с Activiti

Использование Activiti концепция

Уведомления

Интеграция с JasperReports

Подсистема импорта данных из CSV файла

Подсистема периодических заданий

Подсистема AuditLog

Настройка файлов конфигурации

Точки расширения

Коллекции

Язык запросов к данным (ДО) - (DOEL)

Локализация

Аутентификация

3.2 Доменные объекты

Доменные объекты – основные хранимые элементы данных в системе. Каждый доменный объект состоит из некоторого набора именованных значений (полей). Набор полей в конкретном доменном объекте определяется его типом. Типы доменных объектов фиксированы для конкретной информационной системы и определяются в её конфигурации.

Каждое поле может хранить данные одного конкретного типа. Список доступных типов полей приведён в табл.

Тип поля	**Описание**
boolean	Булево значение
string	Строка ограниченной длины

text	Текст без ограничения длины
date-time	Дата и время
dateTimeWithTimeZone	Локальная дата и время (с указанием часового пояса)
timelessDate	Дата (без времени)
reference	Ссылка на доменный объект
long	Целое число
decimal	Десятичная дробь

Поля типа reference позволяют организовывать связи между доменными объектами. Такие поля являются, как правило, типизированными, т.е. могут содержать только ссылки на объекты одного конкретного типа. Возможно также создание нетипизированных ссылочных полей, но она должна быть использована с осторожностью, т.к. такие поля не поддерживают сохранение ссылочной целостности данных. Кроме того, может ухудшаться производительность групповых операций с объектами, содержащими нетипизированные ссылочные поля.

Платформа не поддерживает типы полей, хранящих множественные значения. Вместо таких полей могут использоваться отдельные «маленькие» типы доменных объектов, содержащие по 2 поля, одно из которых предназначено для хранения значения, а второе содержит ссылку на «главный» доменный объект.

3.2.1.1 Идентификация

Каждый доменный объект имеет уникальный идентификатор, по которому возможно получение, сохранение или изменение объекта. Идентификатор объекту назначается платформой при его создании. Идентификаторы доменных объектов используются также для создания связей между объектами.

3.2.1.2 Статус

Каждый доменный объект содержит специальное поле – статус. Оно предназначено для отслеживания состояния объекта в его жизненном цикле. От статуса объекта зависят правила доступа пользователей к нему.

Поле статуса хранит ссылку на специальный доменный объект Status, предопределённый в платформе AF5. Это поле не может изменяться при обычных операциях сохранения доменного объекта. Для изменения статуса объекта предусмотрена отдельная операция.

3.2.1.3 Наследование

Платформа ActiveFrame5 поддерживает парадигму наследования типов доменных объектов. Наследование означает форму отношения двух типов доменных объектов, при которой:

- один тип («дочерний») получает («наследует») все поля, определённые в другом типе («родительском»);
- дочерний тип может использоваться везде, где может использоваться родительский тип.

Платформа поддерживает только одиночное наследование, т.е. у любого типа может быть только один родительский тип (но не наоборот).

3.2.2 Конфигурация

Конфигурация информационной системы содержит определения всех типов доменных объектов, существующих в ней. Конфигурация ядра платформы содержит определения нескольких типов объектов, необходимых для функционирования самой платформы (такие, как группа пользователей, статус). Другие типы объектов определяются в функциональных модулях систем.

Определение типа доменного объекта содержит:

- его (типа) имя;
- ссылку на родительский тип;
- список полей;
- определения ключей уникальности;
- определения индексов;
- список типов вложений;
- имя начального статуса объекта;
- признак хранения истории изменений объектов.

Единственным обязательным элементом определения типа является его имя. Имя типа объекта должно быть уникальным в границах системы (т.е. не совпадать с именем любого другого типа доменного объекта, включая определённые в конфигурации ядра платформы), состоять из букв латинского алфавита, цифр и знака подчёркивания и не превышать в длину 25 символов.

Если используется наследование (задан родительский тип), этот тип также должен быть определён в конфигурации системы. Это определение не обязано находиться том же файле конфигурации, что и тип-наследник, а если и находится, то может быть в любом месте (выше или ниже).

Список полей, хотя и может быть пустым, как правило, содержит определения нескольких полей, предназначенных для хранения данных. Платформа AF5 не накладывает ограничений на количество полей, которые могут быть определены в одном объекте, однако, такие ограничения могут накладываться используемой СУБД.

Ключ позволяет наложить ограничение уникальности на содержимое поля или комбинации полей. Такой ключ не позволит сохранить объект, набор значений полей которого, заданных в ключе, в точности совпадает с таким же набором значений полей другого объекта.

Индексы позволяют увеличить производительность системы при операциях выборки доменных объектов с фильтрацией по значениям полей. Индексы могут строиться как по отдельным полям, так и по их комбинациям. Также для построения индексов могут использоваться SQL-выражения.

Конфигурация типа доменного объекта может содержать объявление одного или нескольких типов вложений (attachments), которые могут присоединяться к этому объекту. Каждый тип вложения становится отдельным типом ДО. Структура этих объектов описана ниже.

Начальный статус объекта позволяет задать имя статуса, который получит объект данного типа при создании. Если он не задан, поле статуса остаётся пустым. Это часто используется в случаях, когда жизненный цикл объекта является вырожденным.

При установленном признаке хранения истории изменений объектов платформа осуществляет сохранение копии объекта при каждом его изменении. Существует также

специальный API для получения как истории изменений конкретного объекта, так и состояния объекта на конкретный момент времени (версии объекта).

3.2.2.1 Группы полей

Для упрощения конфигурирования доменных объектов предусмотрена возможность объявления отдельных именованных групп полей. Эти группы затем могут включаться в различные типы доменных объектов. При таком включении в тип доменного объекта добавляются все поля, входящие в группу – как если бы они были добавлены поодиночке. Группа полей как отдельная сущность в доменном объекте или его типе не существует.

Вложение групп полей друг в друга не предусмотрено.

3.2.3 Структура данных

Для каждого типа доменного объекта создаётся отдельная таблица в БД. Эта таблица имеет ряд служебных столбцов, а также столбцы для хранения данных сконфигурированных полей. В качестве имени таблицы используется имя доменного объекта, заданное в конфигурации.

Набор служебных столбцов зависит от того, является ли данный тип ДО наследником другого типа. Список служебных столбцов для типа, не наследующего другому (корневого), приведён в табл.

Имя столбца	**Тип столбца**	**Назначение**
id	bigint	Числовой идентификатор объекта
id_type	integer	Идентификатор типа объекта
created_date	timestamp	Временная метка создания объекта
updated_date	timestamp	Временная метка обновления объекта
created_by	bigint	Ссылка на пользователя, создавшего объект
created_by_type	integer	
updated_by	bigint	Ссылка на пользователя, изменившего объект
updated_by_type	integer	
status	bigint	Ссылка на объект статуса данного объекта
status_type	integer	
access_object_id	bigint	Идентификатор объекта, используемого для вычисления прав доступа к данному объекту

Столбец id используется для формирования первичного ключа (primary key) таблицы.

Представление всех ссылок в служебных полях совпадает с представлением сконфигурированного ссылочного поля (см. ниже), что позволяет использовать эти поля как обычные ссылочные.

Таблица для некорневого типа содержит из перечисленных только столбцы id и id_type. Кроме того, для этих столбцов в таких типах формируется внешний ключ (foreign key), ссылающийся на такие же поля в таблице родительского типа ДО.

Для каждого сконфигурированного поля добавляется один или несколько столбцов, в зависимости от типа поля. В качестве имени столбца используется имя поля, заданное в конфигурации. Некоторые типы полей хранят данные в двух столбцах. В таком случае к имени одного из них добавляется фиксированный суффикс. Соответствие типов полей типам столбцов приведено в табл.

Тип поля	**Тип столбца**	**Суффикс**	**Примечание**
Boolean	smallint		0 или 1
string	character varying		Длина задаётся в конфигурации
text	text		
date-time	timestamp without time zone		
dateTimeWithTimeZone	timestamp without time zone		
	character varying	_tz	
timelessDate	timestamp without time zone		
reference	bigint		Идентификатор целевого объекта ссылки
	integer	_type	Идентификатор типа целевого объекта ссылки
long	bigint		
decimal	numeric		

Если тип объекта является наследником другого типа, то его таблица содержит только поля, объявленные непосредственно в этом типе. Для хранения значений унаследованных полей создаются записи в таблицах родительских типов.

Для полей типа reference, за исключением нетипизированных, создаётся внешний ключ, ссылающийся на таблицу с данными объектов целевого типа, объявленного в конфигурации поля. Это позволяет поддерживать ссылочную целостность БД.

Ещё один внешний ключ создаётся для пары столбцов id + id_type в случае, когда используется наследование типов. Этот ключ ссылается на такую же пару столбцов в таблице, соответствующей родительскому типу.

Каждая запись в таблице представляет один доменный объект. Данные одного доменного объекта, с учётом наследования типов, могут храниться в нескольких таблицах – по одной записи в каждой таблице, входящих в цепочку наследования типа этого объекта.

3.2.3.1 Исторические данные

Для хранения исторических данных по каждому типу доменных объектов создаётся ещё одна таблица в БД. Имя этой таблицы формируется из имени типа ДО с добавлением к нему суффикса _al (audit log). Структура таблицы совпадает со структурой основной таблицы для типа, но с добавлением нескольких служебных столбцов:

Имя столбца	**Тип**	**Назначение**

	столбца	
operation	integer	Код операции с объектом: 1 – создание, 2 – изменение, 3 - удаление
domain_object_id	bigint	Идентификатор объекта
domain_object_id_type	integer	Идентификатор типа объекта

У этой таблицы не создаются внешние ключи для ссылочных полей, т.к. ссылки в исторических данных не должны препятствовать удалению объектов. По этой же причине не создаётся внешний ключ для пары столбцов domain_object_id + domain_object_id_type, хотя они, по сути, являются ссылкой на исходный доменный объект.

Когда для некоторого типа доменных объектов установлен признак хранения исторических данных, то при каждом сохранении доменного объекта этого типа производится также добавление новой записи (версии) в его таблицу истории. Новый объект в истории получает собственный идентификатор, а идентификатор оригинального объекта сохраняется в паре столбцов domain_object_id + domain_object_id_type.

Версии объектов также являются доменными объектами с точки зрения любых операций чтения. Операции создания, изменения и удаления для этих объектов не поддерживаются.

3.2.4 Встроенные типы ДО

Платформа содержит несколько встроенных (предварительно сконфигурированных) типов доменных объектов, предназначенных для обеспечения её функционирования. Часть из них, предназначенная для реализации основных механизмов платформы, описана ниже. Другие типы объектов, специфические для конкретных подсистем, описаны в документации по этим подсистемам.

3.2.4.1 Status

Объекты типа Status хранят все статусы, в которых могут находиться другие доменные объекты в системе.

Список полей объекта Status приведён в табл.

Поле	**Тип**	**Описание**
Name	string	Уникальный идентификатор (имя) статуса
Description	string	Словесное описание статуса

По полю Name создаётся ключ уникальности, т.к. разные статусы с одинаковым идентификатором существовать не могут.

3.2.4.2 Person

Тип Person представляет пользователя системы. Объекты этого типа используются для авторизации операций.

Список полей объекта Person приведён в табл.

Поле	**Тип**	**Описание**
Login	string	Уникальный идентификатор пользователя в системе

FirstName	string	Имя пользователя
LastName	string	Фамилия пользователя
EMail	string	Адрес электронной почты пользователя (используется для отправки уведомлений)
profile	reference	Ссылка на профиль пользователя – набор его персональных настроек

На поле Login установлен ключ уникальности, т.к. оно используется для определения пользователя по данным аутентификации.

3.2.4.3 Authentication_Info

Тип Authentication_Info предназначен для хранения аутентификационных данных пользователей. Этот тип используется только в том случае, когда система настроена на внутреннюю аутентификацию пользователей (без использования внешнего каталога пользователей).

Список полей объекта Authentication_Info приведён в табл.

Поле	**Тип**	**Описание**
User_Uid	string	Уникальный идентификатор (логин) пользователя
Password	string	Хэш-код (MD5) пароля пользователя

Для поля User_Uid определён ключ уникальности, т.к. это поле используется для идентификации пользователя при входе в систему.

3.2.4.4 Объекты вложений

Типы для хранения объектов вложений не определяются явно в платформе, но создаются при объявлении типа вложения в конфигурации типа ДО по определённому шаблону. Список полей объектов вложений приведён в табл.

Поле	**Тип**	**Описание**
Name	string	Оригинальное имя файла вложения
Path	string	Путь к файлу в хранилище вложений
MimeType	string	Тип данных файла в формате MIME (RFC 1521)
Description	string	Текстовое описание вложения
ContentLength	long	Размер файла в байтах
<Имя типа исходного>	reference	Ссылка на исходный объект

Как видно из этой таблицы, во всех типах объектов вложений совпадают все поля, кроме одного – ссылки на исходный объект. Имя этого поля совпадает с именем типа исходного объекта.

3.3 Иерархии доменных объектов

Поскольку доменные объекты платформы AF5 слишком просты и близки к записям в БД (в частности, не поддерживают хранения множественных значений полей), для удобства моделирования предметной области полезна инфраструктура иерархий доменных объектов. Практически любой бизнес-объект, например, системы электронного документооборота может быть представлен в виде иерархии доменных объектов. Поэтому, по сути, такая инфраструктура предназначена для бизнес-объектов.

Однако, такие средства **не реализованы в платформе AF5**, но активно используются в СМ (их использует Слой сопряжения CMJ с AF5 «Сочи»).

Для каждой сущности, с которой работает CMJ-сервер, существует ровно 1 «корневой» ДОП (КДОП) и N «обратно-связанных с корневым» ДОП-ов (ОДОП), реализующих «многозначность» CMJ-полей. Таким образом, «однозначное» CMJ-поле реализуется как поле КДОП, а «многозначное» - как поле ОДОП, обратно-связанного с КДОП.

Более того, в СМ несколько типов КДОП могут представлять собой по смыслу «иерархию» объектов. Например, РКК (родительский КДОП) + Резолюции и Отчеты (дочерние КДОП). Все КДОП, входящие в такую иерархию, должны получить одинаковый доступ. Для этого у дочерних КДОП указывается «ссылочный доступ» — он берется «по» родительскому КДОП. А в родительском КДОП (его матрице доступа) должны учитываться дочерние.

3.4 Подсистема управления доступом

Подсистема управления доступом (ПУД) AF5 обеспечивает разграничение доступа пользователей к доменным объектам в системе. ПУД реализует дискреционный принцип доступа. Пользователь получает доступ к конкретному объекту только в том случае, если существует правило, предоставляющее пользователю разрешение на доступ данного типа к этому объекту. Помимо разграничения доступа к доменным объектам, ПУД позволяет ограничивать доступ к общесистемным операциям (наподобие создания доменного объекта). Для обобщения подходов объектом доступа в этом случае считается система, а типом доступа – выполнение соответствующей операции.

Правила выдачи разрешений (матрица доступа) определяются в конфигурации системы. Для общесистемных (глобальных) операций всё просто: правила определяют группы пользователей, которые получат разрешение на выполнение данной операции. О группах речь пойдёт ниже, пока только необходимо подчеркнуть, что разрешения выдаются не отдельным пользователям, а группам.

Для контекстных операций вводится понятие контекстной роли. Под ней мы понимаем набор групп (опять групп!) пользователей, имеющих отношение к данному объекту. При этом отношение может быть совершенно любым, способ его определения отдаётся на откуп настройщику или разработчику: из содержимого полей данного доменного объекта или любых связанных с ним, с помощью SQL-запроса или даже собственного Java-класса. Как бы это ни происходило, контекстная роль, получая параметром доменный объект (контекст), раскрывается в набор групп, которым могут быть предоставлены права на тот или иной тип доступа к этому объекту. Таким образом, доступ к объекту может зависеть от его содержимого.

Теперь, наконец, о группах пользователей. Они бывают двух видов: статические и динамические. Состав статических групп определяет администратор через свой интерфейс: он может добавить пользователя в группу или удалить из неё. (Следует отметить, что он не может создавать или удалять сами группы, так как они используются

в правилах доступа, определяемых в конфигурации.) Состав динамических групп вычисляется системой. И вновь у настройщика/разработчика имеется большой простор в выборе способа вычисления: SQL-запрос, Java-класс. Но самое интересное в динамических группах то, что они также могут быть контекстными, т.е. зависеть от доменного объекта! Фактически, при определении в конфигурации одной контекстной динамической группы пользователей ПУД создаёт множество реальных групп – столько, сколько есть подходящих доменных объектов. И состав каждой вычисляется исходя из содержимого своего контекстного объекта, т.е. можно снова использовать его поля или связанные объекты.

Важно отметить, что разрешения на конкретные типы доступа выдаются конкретным группам пользователей. И алгоритм, вычисляющий состав контекстной роли, должен определять для контекстной динамической группы её реальный экземпляр (контекст). Чаще всего, именно те доменные объекты, на которые ссылается контекстный или связанный с ним объект, будут, в свою очередь, становиться контекстом для динамических контекстных групп пользователей. Правила доступа для глобальных операций могут ссылаться на контекстные группы пользователей, но тоже должны указывать какой-либо метод определения контекста для такой группы.

3.4.1 Примеры

Примеры статических групп: Администраторы, Делопроизводители. Живой человек (администратор) при необходимости вручную (через административный интерфейс) добавляет и удаляет пользователей в эти группы. (См. также «О ПРОБЛЕМЕ КУРИЦЫ И ЯЙЦА»).

Примеры динамических групп без контекста:

- Все пользователи – система автоматически добавляет туда каждого пользователя (доменный объект «Пользователь»), заведённого в системе.
- Делопроизводители – альтернативный вариант реализации группы, добавляющий пользователя в неё на основании, например, некоторого флага (поля) в его профиле (доменном объекте).

Примеры динамических контекстных групп:

- Пользователь и его заместители – по одной группе на каждый доменный объект «Пользователь», в которую, помимо самого этого пользователя, добавляются также временно или постоянно замещающие его пользователи на основе доменных объектов «Замещение». Алгоритм вычисления группы может учитывать и увольнение сотрудника, удаляя его самого из его же группы, при этом его заместители смогут выполнять его функции.
- Сотрудники подразделения – по одной группе на каждый доменный объект «Подразделение», вычисляемой на основании ссылок на подразделение в доменных объектах «Сотрудник».
- Сотрудники подразделения пользователя – вычисляется аналогично предыдущей, но по одной на каждый доменный объект «Сотрудник».

Примеры контекстных ролей:

- Автор документа (для доменного объекта «Документ») – включает группу «Пользователь и его заместители», контекстом для которой является пользователь из поля «Автор».

- Исполнители поручений (для доменного объекта «Документ») – включает экземпляры группы «Пользователь и его заместители» для всех пользователей, содержащихся в полях «Исполнитель» каждого дочернего объекта «Поручение».
- Подписант документа-основания (для доменного объекта «Поручение») – включает группу «Пользователь и его заместители» с контекстом пользователя из поля «Подписант» дочернего объекта «Подпись» родительского объекта (типа «Документ»).

3.4.2 О типах доступа

Матрица доступа к доменным объектам определяется с точностью до его типа и статуса. Определены следующие типы доступа к доменным объектам:

- чтение
- изменение (запись)
- удаление
- создание дочернего объекта заданного типа
- выполнение заданного конфигурируемого действия

Типы доступа для общесистемных операций перечислены ниже:

- создание доменного объекта заданного типа
- выполнение заданного конфигурируемого действия

Из перечисления видно, что под большинством перечисленных элементов скрывается множество действительных типов доступа, отличающихся конкретными типами объектов или действиями. Права доступа на каждую для каждого такого типа (например, «Создание дочернего объекта „Поручение“», «Создание дочернего объекта „Подпись“» и т.д.) назначаются независимо друг от друга.

Также необходимо отметить, что в списке отсутствует важнейшая контекстная операция «Изменение статуса». Причина заключается в том, что основным механизмом управления жизненным циклом доменного объекта в системе должны быть бизнес-процессы. Они являются (важной) частью бизнес-логики и работают от имени системы. Ручное изменение статуса пользователем (или иным клиентом) системы рассматривается в качестве исключения, и при наличии такой необходимости оно может быть выставлено в качестве конфигурируемого контекстного действия, операция выполнения которого уже является объектом назначения прав доступа. Конечно, при этом подходе операция изменения статуса доменного объекта не может быть вынесена в публичный API системы.

Та же логика может быть применена и к глобальному действию «Создание доменного объекта заданного вида», но отсутствие этой операции в публичном API видится более нелогичным, поэтому она всё же включена в список доступных для назначения прав операций. Тем не менее, это решение таит в себе ряд неудобств. В интерфейсе системы создание недочерних доменных объектов должно быть конфигурируемой функцией (предусмотрено много различных способов их создания в различных местах), для чего весьма удобную концепцию предоставляют конфигурируемые действия. Однако, их выполнение также является операцией, управляемой ПУД, и мы должны либо требовать от настройщика дублировать правила доступа для этих операций, либо предусматривать способ их логического связывания (и автоматического распространения правил с одной на другую).

Наконец, тот же вопрос стоит и в отношении контекстной операции «Назначение (запуск) заданного бизнес-процесса» (для доменного объекта). Если функции управления бизнес-процессами выносятся в публичный API, то они должны быть под управлением ПУД, а список дополнен соответствующими операциями. Или же эти функции генерализуются через интерфейс конфигурируемых действий, и тогда дополнительные операции в ПУД не нужны.

3.4.3 Разница между контекстными группами и контекстными ролями

Собственно, зачем же в ПРД нужны контекстные роли? Они очень похожи на контекстные группы пользователей. Внешняя разница между ними заключается лишь в том, что контекстные группы включают в себя пользователей, а контекстные роли – группы пользователей (не обязательно контекстные, но в первую очередь именно их). Любые правила можно технически реализовать с использованием только контекстных групп.

Важнейшее различие между этими объектами заключается в их внутренней реализации. Контекстные группы пользователей существуют физически – в виде записей в БД, по одной на каждый контекстный объект для каждой группы. Хранится также состав каждой такой группы – список пользователей, входящих в неё. При каждом изменении контекста (доменного объекта или связанных с ним, влияющих на состав группы) производится повторное вычисление состава этой группы.

Контекстные роли, с другой стороны, существуют только в конфигурации. Их состав не хранится в БД, а вычисляется непосредственно при вычислении списка доступа к доменному объекту.

Этими свойствами групп и ролей обуславливается разница в их использовании. Контекстные группы пользователей предназначены для использования с относительно редко изменяющимися объектами в качестве контекста, однако позволяют задавать относительно сложные правила их формирования. Для СЭД такими объектами являются пользователи и связанные с ними сущности (профили), организационно-штатная структура и т.п. Контекстные роли связываются с динамичными объектами системы (документы, поручения и т.п.), существующими в большом количестве, но должны иметь максимально лёгкие и быстрые алгоритмы для их вычисления.

3.4.4 Списки доступа

Списки доступа (Access control list, ACL) – внутренний объект ПУД, предназначенный для оптимизации запросов определения доступа.

Списки доступа для контекстных операций хранятся в таблицах БД – по одной таблице на каждый тип доменных объектов. Списки доступа хранят *актуальные* разрешения (permissions) на доступ различного типа к каждому доменному объекту. Субъектами разрешений выступают группы пользователей. Актуальность означает хранение только тех разрешений, которые действительны для текущего статуса доменного объекта. При изменении статуса объекта производится повторное вычисление списка доступа к нему.

ПУД также отслеживает все изменения доменных объектов и другие события, которые могут привести к изменению списков доступа, и вносит соответствующие изменения в списки.

Наличие списков доступа позволяет реализовать проверку прав пользователя на выполнение той или иной операции (включая операции над множеством однотипных

объектов, например, чтение списка) через обращение лишь к двум таблицам БД: списку доступа (частному для этого типа объектов) и составу групп пользователей.

С учётом тотального преобладания количества разрешений на чтение над количеством разрешений на любые другие операции над объектом, а также количества запросов прав чтения над всеми прочими, списки доступа разбиваются на две отдельные таблицы (для каждого типа доменных объектов, конечно): списки чтения и списки доступа (остальные операции).

3.4.5 Архитектура

ПУД состоит из нескольких компонентов, включённых в ядро системы. Ключевой из них – служба контроля доступа (СКД), обеспечивающая проверку прав. Она основана на схеме маркеров доступа (access tokens). Все сервисы DAO-слоя (базовые сервисы), осуществляющие непосредственный доступ к защищаемым объектам, требуют от вызывающего их кода предоставления специального объекта – маркера доступа. Это объект инкапсулирует данные о субъекте, объекте и типе доступа (операции). Они обращаются к СКД для проверки подлинности и соответствия маркера доступа запрошенной операции, и только в случае положительного ответа производят нужный доступ. Для получения маркера доступа код бизнес-слоя системы также должен обратиться к СКД, которая при этом производит проверку прав (авторизацию) субъекта доступа. Такой подход блокирует возможность доступа к доменным объектам в обход СКД.

Следует отметить, что субъектом доступа может быть не только пользователь. Любой код, выполняющийся в бизнес-слое, может запросить маркер доступа как от имени пользователя (любого), так и от имени системы, указав при этом идентификатор процесса. Права доступа системы к любым объектам не ограничены, но идентификатор процесса, явно указанный в его коде, сохраняется в системных журналах, протоколирующих доступ к защищаемым объектам. Это позволяет не давать пользователям лишних прав на изменение тех объектов, которые должны изменяться только бизнес-правилами при внесении изменений пользователем в другие объекты (связанные), упрощая конфигурацию прав и защищая объекты от несогласованных «ручных» изменений. Кроме того, протоколирование источников изменений с точностью до процесса упрощает отладку системы и проведение расследований по обращениям заказчиков.

В СКД также реализуется возможность делегирования базовым сервисам функций по проверке прав доступа. Такие сервисы получают возможность встраивания запросов проверки прав в основные запросы, осуществляющие доступ к объектам. Эта возможность используется лишь для некоторых (доверенных) сервисов и для ограниченного набора типов доступа, в основном – для чтения отдельных доменных объектов и коллекций. Принимая на себя такую ответственность, доверенные сервисы становятся, по сути, частью ПУД.

СКД и доверенные базовые сервисы осуществляют проверки прав с использованием списков доступа и таблиц членства пользователей в группах.

Другая важная часть ПУД – компоненты, обеспечивающие актуальность списков доступа. В их число входят классы, реализующие различные алгоритмы формирования контекстных ролей и динамических групп пользователей. Помимо собственно вычисления их состава, эти компоненты реагируют на различные события, происходящие в системе, которые могут повлиять на состав вычисляемых ролей или групп (прежде всего – изменение доменных объектов), осуществляя немедленный пересчёт. И снова некоторая доля ответственности ложится на базовые сервисы, которые обязаны информировать компоненты ПУД о важных для них событиях.

3.4.6 Распространение прав доступа

Согласно требованиям по защите информации, система должна содержать средства, ограничивающие распространение прав на доступ. Под распространением мы понимаем неявное предоставление прав доступа к объектам вследствие изменения их содержания. Поясню на примере: пользователю выдано поручение по документу, вследствие чего, по соответствующему правилу («Все исполнители поручений могут читать документ-основание») он имеет доступ к документу на чтение. Предположим, что он перепоручает своё задание другому лицу, или выдаёт связанное поручение. Это лицо автоматически попадает в контекстную роль «Все исполнители поручений» и получает доступ на чтение к тому же документу. В соответствии с бизнес-задачей, всё так и должно быть. Но вот соответствует ли это требованиям безопасности? Имеет ли право пользователь предоставлять таким образом доступ к документу кому угодно? Или круг лиц, кого он может вовлечь, должен быть ограничен? Или так могут делать пользователи, но не все?

Частично защита от неконтролируемого распространения прав обеспечивается строгим подходом к конфигурированию правил доступа. В некоторых случаях может понадобиться дополнительный защищаемый тип доступа – привязка доменного объекта к другому (контекстному). Также возможно создание дополнительного сервиса со своим набором правил, описывающих разрешения на распространение прав (кто – кому). Этот сервис должен контролировать все изменения списков доступа и блокировать запрещённые. (Единственно надёжным вариантом здесь представляется откат транзакций, пытающихся выполнить несанкционированные изменения, т.к. просто удалённые разрешения будут выданы позже при изменении объекта от имени системы.)

Перед окончательным выбором решения необходимо выработать единый подход к ней со стороны специалиста по защите информации и бизнес-аналитика системы.

3.4.7 Доступ к полям объектов

В некоторых случаях выдвигаются требования дополнительного разграничения доступа к отдельным полям объектов (как на чтение, так и на изменение). Непосредственная реализация этих требований может привести к чрезмерному росту как объёма списков доступа, и без того немалого, так и количества вычислений этих списков.

Для удовлетворения подобных требований можно предложить использовать декомпозицию доменных объектов на такие части, что правила доступа ко всем полям одной части будут одинаковы, и соединение этих доменных объектов в иерархию.

3.4.8 О проблеме курицы и яйца

Уже упоминалось, что управление составом групп пользователей осуществляет администратор. Но сам администратор – это не кто иной, как пользователь системы, включённый в группу «Администраторы». Но откуда же возьмётся самый первый администратор? (Подобная проблема, кстати, встаёт и с пользователями вообще – управление пользователями обычно осуществляет также администратор, но он сам является пользователем, и кто-то должен добавить его самого.)

Для решения этой проблемы при старте системы производится специальная проверка. Если группа «Администраторы» пуста (не содержит ни одного пользователя), создаётся специальный пользователь system_administrator (если он ещё не существует) и добавляется в группу «Администраторы». Это не только решает проблему первого старта, но и защищает систему от случайного удаления всех администраторов. Конечно, доменные объекты «Пользователь», «Группа пользователей», а также правила,

предоставляющие права на изменение этих объектов группе «Администраторы» задаются в системной конфигурации (недоступной для изменения настройщиком).

3.4.9 Дополнительные функции

Требуется реализовать возможность включения одних групп пользователей в другие. При этом включение может быть вложенным (на любую глубину). Такая функция была реализована в прототипе системы. Детали конфигурирования и реализации будут изложены позже.

С учётом наличия в системе большого количества служебных доменных объектов (в частности, для реализации полей с множественными значениями) необходимо реализовать возможность «наследования» списков доступа дочерними объектами от родительских. Т.е. для конкретного типа доменного объекта в конфигурации может быть указано использование родительских списков доступа; в таком случае вместо собственных таблиц для него создаётся отдельная таблица (или поле) с указателем на имеющего список доступа родителя (т.к. наследование может быть многоуровневым), а запросы определения прав должны использовать списки доступа этого родителя.

Для следующих версий системы рассматривается возможность дополнительной реализации мандатного принципа доступа. При этом для выполнения операции пользователь должен будет иметь разрешения по обоим принципам: и дискреционному (нынешнему), и мандатному.

3.4.10 Служба контроля доступа

Служба контроля доступа (СКД) является частью подсистемы управления доступом (ПУД). Она контролирует доступ к доменным объектам и глобальным операциям в системе. Кроме того, она осуществляет протоколирование этих событий.

В основе реализации системы лежит принцип передачи маркеров доступа (authorization или access tokens). Маркер доступа содержит в себе информацию о субъекте, объекте и типе доступа. Он формируется только в том случае, если данный тип доступа к данному объекту разрешён данному субъекту. (Это пуристский подход, который в целях оптимизации быстродействия следует несколько усложнить – см. далее.) Для выполнения любой базовой операции с объектом (чтение, запись, удаление) требуется наличие маркера доступа на соответствующий тип доступа к этому объекту. Создаваться маркеры доступа могут только через СКД по запросу бизнес- или (реже) DAO-слоя системы. Хорошей идеей выглядит хранение маркеров доступа в контексте текущего запроса (транзакции) вместо явной передачи их через параметры методов.

Маркеры доступа на выполнение базовых операций являются простыми. Кроме них, СКД умеет формировать универсальные маркеры, которые могут выступать заместителями (proxies) простых маркеров при проверке прав доступа. Эти маркеры используются, например, для выполнения операций от имени системы.

В рамках предложенной схемы понятие субъектов доступа должно быть расширено по сравнению с описанием настройки прав доступа. Субъектами доступа для СКД являются не только пользователи, но и компоненты системы, осуществляющие изменения объектов, в том числе конфигурируемые действия и бизнес-процессы.

Реализация такой службы создаст единый компонент, контролирующий доступ к защищаемым объектам системы. Доступ к объектам в обход компонента будет невозможен; для выполнения каких-либо операций (даже чтения объектов) от имени системы код должен будет получить универсальный маркер доступа, обратившись в СКД. Теоретически, СКД в соответствии с какими-либо своими настройками может и не

предоставить такой маркер, т.е. запретить доступ. Можно также рассмотреть возможность дескриптивного назначения маркера доступа классам или методам (через аннотации или beans.xml).

Ещё одним преимуществом предложенной архитектуры является простота реализации протоколирования событий. Маркер доступа агрегирует всю информацию, необходимую для внесения записи в протокол работы. Протоколирование может осуществляться, например, одновременно с проверкой маркера доступа перед выполнением базовой операции.

3.4.10.1 Оптимизация производительности

Существенным недостатком предложенной схемы является необходимость выполнения проверок прав отдельными запросами в БД (при создании маркера доступа), что может негативно сказаться на времени отклика системы. Схема БД при этом построена таким образом, чтобы позволять формирование эффективных запросов самих объектов одновременно с проверкой прав доступа к ним. Таким образом, необходимо отказаться от жёсткого следования принципу проведения проверки в момент формирования маркера доступа.

Итак, некоторые простые маркеры доступа могут формироваться без проверки соответствующих разрешений пользователя на доступ – отложенные (deferred) маркеры. Получив такой маркер, базовый сервис (DAO) обязан позаботиться о включении в запрос соответствующих проверок. Отложенные маркера наиболее важны для эффективной фильтрации коллекций, которая осуществляется путём встраивания фильтров по правам в запросы к БД. Возможно использование отложенных маркеров и для других операций. Это означает, что часть важнейшей функции СКД по проверке прав пользователей эта служба делегирует базовым сервисам управления доменными объектами и коллекциями.

Ещё одним способом оптимизации производительности (а зачастую, и увеличения удобства использования) являются групповые маркеры доступа. Они содержат список объектов доступа (их идентификаторов) либо список типов доступа. Такие списочные проверки легко реализуются единым запросом в БД, существенно сокращая количество обращений к ней, а бизнес-сервисы получают возможность ранней проверки прав доступа к множеству объектов, которые понадобятся для выполнения одного запроса.

3.4.10.2 Посмотрим на примерах

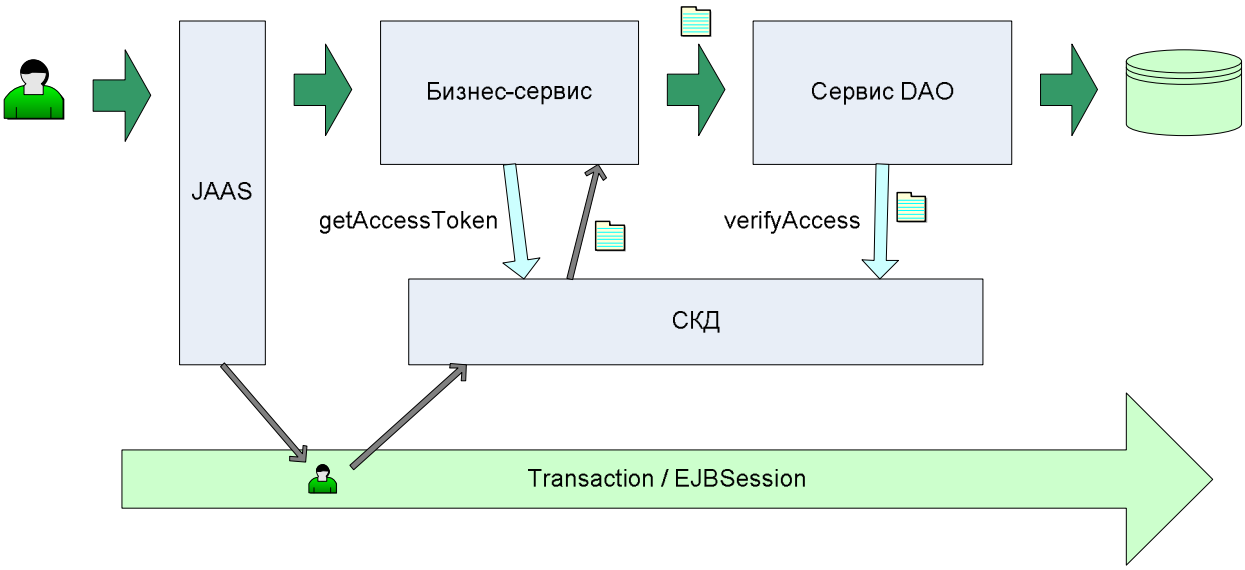
- DomainObjectService получает запрос на чтение доменного объекта.
  Он обращается в СКД, запрашивая маркер доступа на чтение этого объекта (без указания пользователя; СКД может взять его из контекста EJB), затем вызывает DomainObjectDao. (Следует заметить, что при пуристском подходе проверка прав доступа произойдёт при формировании маркера, в случае неудачи будет выброшено исключение, и обработка запроса завершится. В оптимизированном варианте проверка на этом этапе не производится.) DomainObjectDao проверяет полученный маркер доступа на применимость к своей операции и, исходя из того, что маркер является отложенным, формирует SQL для чтение объекта в БД, добавляя в него проверки на права доступа пользователя. Если подходящей записи не нашлось, то либо идентификатор неправильный, либо доступ запрещён, и тогда уже бросается исключение.
- ActionService (сервис обработки конфигурируемых действий) получает запрос на выполнение действия «Создать новый документ».
  Он обращается к СКД с запросом маркера доступа на выполнение действия. Этот маркер – не отложенный (проверка прав осуществляется сразу), но универсальный:

он замещает любые другие маркеры доступа. Субъектом доступа в нём является действие, а не пользователь, вызвавший оное. Получив маркер, сервис передаёт управление классу, указанному в конфигурации действия. Этот класс в соответствии с реализованной в нём бизнес-логикой вызывает DomainObjectDao, который запрашивает маркер доступа на системное действие. Поскольку имеющийся у нас сложный маркер замещает любое действие, требование DomainObjectDao удовлетворено, и тот создаёт доменный объект в БД.

- DomainObjectService получает запрос на сохранение доменного объекта «Поручение». При этом в объекте «Документ» у нас сконфигурировано вычисляемое поле «Срок исполнения», которое вычисляется специальным классом как максимальное значение поля «Срок исполнения» из всех дочерних доменных объектов «Поручение». Разумеется, вычисление этого поля в родительском документе должно производиться при изменении объекта «Поручение».
  Итак, вначале всё идёт так же, как при чтении объекта. Вопрос о том, является ли маркер доступа на запись объекта отложенным, здесь не важен. После сохранения в БД объекта «Поручение» вызывается код нашего специального класса для вычисления поля (через точку расширения, видимо). И поскольку он будет читать множество объектов, и даже менять один (документ), а вовсе не факт, что у сохранившего поручение пользователя есть права на все эти операции, класс должен будет обратиться к СКД за универсальным маркером доступа. При наличии такого маркера класс сможет производить любые манипуляции с объектами системы (действовать от её имени), но именно он становится субъектом этих операций, и это будет отражаться в протоколе.
- Activiti исполняет service task бизнес-процесса.
  Поскольку бизнес-процесс действует от имени системы, инфраструктура взаимодействия с Activiti в ходе подготовки среды исполнения для класса, реализующего этот task (JavaDelegate), запрашивает универсальный маркер доступа. Субъектом доступа провозглашается данный бизнес-процесс. Любой базовый сервис, проверяя соответствие маркера своей операции, получает положительный ответ и может выполнять запрошенные действия.
- CollectionService получает запрос на чтение списка (коллекции) объектов.
  Он обращается в СКД, запрашивая доступ на чтение коллекции объектов (отдельный тип доступа!) Предоставленный маркер является отложенным, значит, исключение по нехватке прав здесь невозможно. CollectionServiceDao, получив маркер и определив, что он отложенный, встраивает в запросы проверки прав доступа для субъекта (пользователя), указанного в маркере доступа.

3.4.10.3 А теперь – слайды!

Схема взаимодействия СКД с различными компонентами системы:



3.4.11 Вхождение группы в группу

Для реализации вхождения группы в группу необходимо исходить от следующих условий:

- ресурсоемкость запросов на получение доменных объектов доступных на чтение должна возрасти незначительно;
- сложность запросов на получение доменных объектов доступных на чтение не должна зависеть от уровня вложенности групп друг в друга;
- не должно быть разницы для включения статических или динамических групп друг в друга

Для реализации данных требований целесообразно хранить развернутую структуру вхождения групп в группы в дополнение к иерархической структурой и обновлять развернутую структуру при изменениях, вносимых в иерархическую структуру. Под развернутой структурой подразумевается сущность, в которой хранится вхождение групп в группы с учетом иерархии. При создание новой группы в сущности хранящей развернутую структуру создается первая запись для новой группы, со ссылкой на саму себя. Это позволяет не менять запросы на получение прав доступа в зависимости от наличия или отсутствия дочерних групп. При получение прав доступа используется только развернутая структура.

3.4.11.1 Техническое решение

Создаются две сущности:

1. GROUP_GROUP_SETTINGS – конфигурация вхождения групп в группы. Хранятся ссылки только на непосредственно включенные группы. Используется при администрирование и получение непосредственного состава группы. Изменяется с помощью Public API. Доступно для изменения администраторам.

Поля:

PARENT_GROUP_ID – родительская группа, ссылка на сущность USER_GROUP.

CHILD_GROUP_ID – дочерняя группа, ссылка на сущность USER_GROUP.

2. GROUP_GROUP – развернутая структура вхождения групп в группы. Хранятся ссылки на саму себя, на непосредственно включенные группы и на группы

включенные в дочерние группы. Используется при получение всех дочерних групп по родителю. Заполняется по событиям изменения сущности GROUP_GROUP_SETTINGS. Запрещается редактирование с помощью public API, недоступно администраторам для редактирования.

Поля:

PARENT_GROUP_ID – родительская группа, ссылка на сущность USER_GROUP.

CHILD_GROUP_ID – дочерняя группа, ссылка на сущность USER_GROUP.

Упрощенная модель данных отображена на рис. 1.

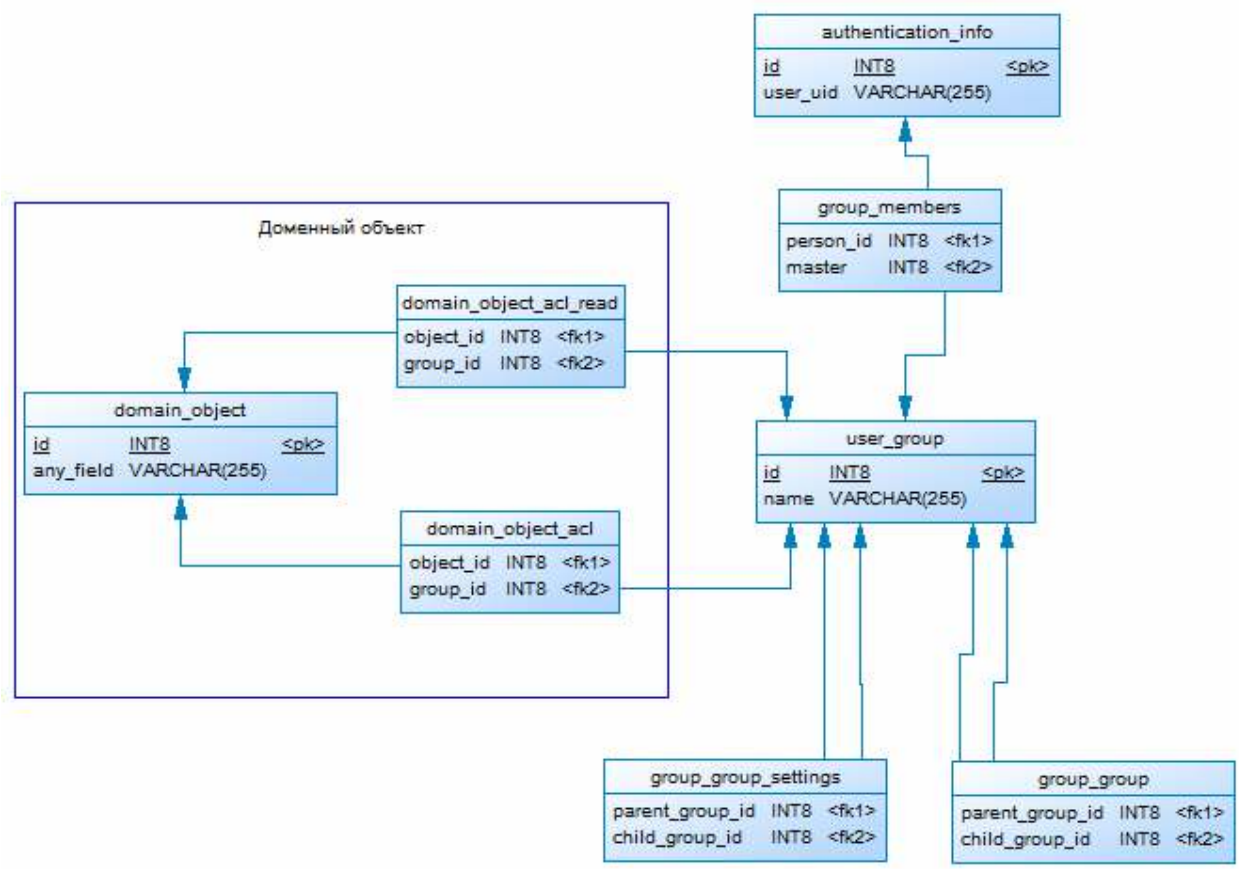


Рис. 1. Упрощенная модель данных, при реализации включения группы в группу.

3.4.11.2 Описание алгоритма работы

Необходимо создать сервис для работы с группами (возможно стоит использовать существующий сервис) и метод reclacGroupMembers(groupId), который пересчитывал бы записи в сущности GROUP_GROUP по идентификатору группы.

При добавление группы в группу необходимо:

- Добавляется запись в GROUP_GROUP_SETTINGS
- Для родительской группы и для всех групп куда входит родительская группа в качестве дочерней пересчитываются записи в сущности GROUP_GROUP вызовом метода reclacGroupMembers

При удаление дочерней группы из родительской следует:

- Удалить запись в GROUP_GROUP_SETTINGS

- Для родительской группы и для всех групп куда входит родительская группа в качестве дочерней пересчитываются записи в сущности GROUP_GROUP вызовом метода reclacGroupMembers

При получение актуальных прав пользователя на доменный объект необходимо модифицировать существующий запросы, приведя их к подобному виду:

Получение acl для доменного объекта:

```
Select acl.* from domain_object_acl acl
Inner join group_group gg on (acl.group_id = gg.parent_group_id)
Inner join group_members gm on (gg.child_group_id = gm.master)
Where gm.person_id = :user_id
And acl.object_id = :object_id
```

, где

user_id – идентификатор пользователя из сущности authentication_info

object_id – идентификатор доменного обьекта

Получение списка доменных объектов:

```
Select domain_object.* from domain_object d
Where exists (select acl.object_id from domain_object_acl_read acl
Inner join group_group gg on (acl.group_id = gg.parent_group_id)
Inner join group_members gm on (gg.child_group_id = gm.master)
Where gm.person_id = :user_id
And acl.object_id = d.id
)
```

, где

user_id – идентификатор пользователя из сущности authentication_info

3.4.12 Сервис управления пользователями и группами

Для более удобной работы с доменными объектами пользователи и группы создан spring сервис ru.intertrust.cm.core.dao.api.PersonManagementServiceDao и соответствующий ejb ru.intertrust.cm.core.business.api.PersonManagementService.

Описание методов доступны в общем javadoc и продублированы ниже.

3.4.12.1.1 Method Summary

Methods	
Modifier and Type	**Method and Description**
`void`	**`addGroupToGroup`**`(`**`Id`** `parent, `**`Id`** `child)` Добавление группы в группу
`void`	**`addPersonToGroup`**`(`**`Id`** `group, `**`Id`** `person)` Добавление персоны в группу
`List<DomainObject>`	**`getAllChildGroups`**`(`**`Id`** `parent)` Получение групп входящих в группу с учетом вхождения группы в группу
`List<DomainObject>`	**`getAllParentGroup`**`(`**`Id`** `child)` Получение всех родительских групп для группы, с учетом вхождения группы в группу
`List<DomainObject>`	**`getAllPersonsInGroup`**`(`**`Id`** `groupId)`

	Получение списка персон входящих в группу, с учетом вхождения группы в группу
`List<DomainObject>`	**`getChildGroups`**`(Id parent)` Получение групп, непосредственно входящие в группу
`Id`	**`getGroupId`**`(String groupName)` Получение идентификатора группы по имени
`List<DomainObject>`	**`getPersonGroups`**`(Id personId)` Получения списка групп, в которые входит персона с учетом наследования
`Id`	**`getPersonId`**`(String login)` Получение идентификатора персоны по его логину
`List<DomainObject>`	**`getPersonsInGroup`**`(Id groupId)` Получение списка персон, входящих в группу
`boolean`	**`isGroupInGroup`**`(Id parent, Id child, boolean recursive)` Проверка вхождения группы в группу.
`boolean`	**`isPersonInGroup`**`(Id groupId, Id personId)` Проверка входит ли персона в группу с учетом вхождения группы в группу
`void`	**`remoteGroupFromGroup`**`(Id parent, Id child)` Удаление группы из группы
`void`	**`remotePersonFromGroup`**`(Id group, Id person)` Удаление персоны из группы

3.5 Работа с вложениями

3.5.1 Общие положения

1. С вложениями предполагается работать как с обычными доменными объектам. Для каждого загруженного вложения создается доменный объект Вложение.

Для каждого типа доменного объекта создается свой тип доменного объекта Вложение.

2. Для указания, что у доменного объекта могут быть Вложения, а заодно и названия таблицы для Вложений используется тег `attachment-type` внутри конфигурации `domain-object-type`

Причем у одного ДО может быть несколько типов Вложений (например, если нужно задать разные права для разных типов Вложений). Пример указания всех типов Вложений для ДО.

```
        <attachment-types>
            <attachment-type name="Outgoing_Document_Atch"/>
        </attachment-types>
```

Атрибут `name` указывает название таблицы вложений.

Поддержку элементов `attachment-types` и attachment-type нужно добавить в DomainObjectTypeConfig.

Состав полей объектов Вложения конфигурируется в доменном объекте Attachment

```
    <domain-object-type name="Attachment" template="true">
        <fields>
            <string name="Name" length="128" />
            <string name="Description" length="128" />
        </fields>
```

```
    </domain-object-type>
```

Флагом того, что это не обычный доменный объект и для него не надо создвавать таблицу, является атрибут `template="true".` Это шаблонный объект, который используется при построении других доменных объектов. Шаблонные объекты ДО Вложений должны находится в системном конфигурационном файле. Сейчас системный конфигурационный файл **domain-objects.xml** включает и определение бизнес объектов, которые должны будут уйти из него.

Поддержку атрибута `template` нужно добавить в DomainObjectTypeConfig.

3.Так как доменные объекты Вложения не указываются явно в конфигурационном файле, то для получения этих объектов нужно изменить реализацию `ConfigurationExplorerImpl.` Этот класс хранит закешированные доменные объекты в отдельной хеш таблице. Поэтому при добавленни ДО в кеш, можно проверять имеет ли этот доменный объект Вложения и если так, создавать ДО Вложения на основе шаблонна и добавлять их в кеш. Такой механизм позволит получить ДО Вложения через `ConfigurationExplorerImpl.getDomainObjectTypeConfig()` и `ConfigurationExplorerImpl.getDomainObjectConfigs().`

Сами Вложения (содержимое загруженых файлов, далее контент Вложений) предлагается хранить на файловой системе.

3.5.2 Хранение вложений

Для каждого типа доменного объекта создается соответсвующая таблица для Вложений, название которой указывается в конфигурации,

с полями

id,

name, – первоначальное имя атачмента

path, – относительный путь к атачменту. Путь указывается относительно корневой директории хранения, которая настраивается в конфигурации.

mimeType, - mime тип атачмента

parent – уникальный идентификатор доменного объекта, для которого загружен атачмент. Ссылается на поле id соответсвующей таблицы доменного объекта.

3.5.3 Сервисы

Сервис для работы с доменными объектами Вложение и их контентом AttachmentService.

Этот сервис предоставляет Public API для работы с Вложениями.

С ДО Вложения можно работать, как с обычными ДО объектами, используя CrudService. Например, для поиска ДО Вложений по идентификатору и др.

Но все специфические операции работы с Вложениями вынесены в AttachmentService.

```
/**
 * Работа с доменными объектами Вложения. Вынесен в отдельный сервис, так как
нужны функции работы с контентом Вложений.
 * @author atsvetkov
 *
 */
public interface AttachmentService {
```

```
     /**
      * Удаленный интерфейс для EJB
      * @author atsvetkov
      *
      */
     public interface Remote extends AttachmentService {
     }

       /**
      * Создает доменный объект (ДО) Вложение на основе его типа, не сохраняя
его в СУБД.
      * @param objectId доменный объект, для которого создается ДО Вложение
      * @param attachmentType тип Вложения
      * @return пустой ДО Вложение
      */
     DomainObject createAttachmentDomainObjectFor(Id objectId, String
attachmentType);

  /**
      * Сохраняет доменный объект Вложение и его контент. Если id ДО Вложение
пустое, то создает новый объект. Иначе,
      * замещает файл вложения для существующего ДО Вложение.
      * @param inputStream обертка для для java.io.InputStream, используется
для перемещения файлов в потоке по сети
* @param attachmentDomainObject доменный объект Вложение
* @return сохраненный доменный объект Вложение
      */

     DomainObject saveAttachment(RemoteInputStream  inputStream, DomainObject
attachmentDomainObject);

     /**
      * Загружает контент для доменного объекта Вложение.
      * @param attachmentDomainObject ДО Вложение
      * @return обертка для для java.io.InputStream, используется для
перемещения файлов в потоке по сети
      */

     RemoteInputStream loadAttachment(DomainObject attachmentDomainObject);
     /**
      * Удаление ДО Вложение и его контента.
      * @param path id в БД или абсолютный путь на жёстком диске
      */
     void deleteAttachment(DomainObject attachmentDomainObject);

     /**
      * Получает список ДО Вложений для переданного ДО.
      * @param domainObject ДО, для которого находятся вложения
      * @return список ДО Вложений
      */
     List<DomainObject> getAttachmentDomainObjectsFor(DomainObject
domainObject);
}
```

Для передачи потоков по сети (в случае вызова данного сервиса как remote EJB) используется библиотека RMIIO (http://openhms.sourceforge.net/rmiio/). Библиотека доступна в стандартном мавен репозитарии:

http://repo1.maven.org/maven2/com/healthmarketscience/rmiio/rmiio/2.0.3/

```
    <dependency>
        <groupId>com.healthmarketscience.rmiio</groupId>
```

```
        <artifactId>rmiio</artifactId>
        <version>2.0.3</version>
    </dependency>
```

Объект RemoteInputStream обертывает java.io.InputStream и выполняет передачу контента по сети по частям.

Для работы непосредственно с контентом Вложений предлагается Dao сервис **`AttachmentContentDao`**.

```
/**
 * Предоставляет операции для сохранения/загрузки/удаления Вложений на
файловой системе.
 * @author atsvetkov
 *
 */
public interface AttachmentContentDao {

    /**
     * Сохраняет Вложение в хранилище на файловой системе. Путь к хранилищу
указывается в настройках разворачивания приложения.
     * @param inputStream поток с Вложением
     * @param attachmentDomainObject доменный объект Вложение
     * @return относительный путь к сохраненному Вложению
     */
    String saveContent(InputStream  inputStream);

    /**
     * Загружает Вложение по относительному пути в хранилище.
     * @param domainObject ДО Вложение
     * @return поток с Вложением
     */
    InputStream loadContent(DomainObject domainObject);

    /**
     * Удаление Вложения по относительному пути в хранилище.
     * @param domainObject ДО Вложение
     */
    void deleteContent(DomainObject domainObject);

}
```

Замечу, что этот сервис использует уже обычный `java.io.InputStream` для передачи контента Вложений, т. к. предполагается, что он будет находится на одном сервере с вызывающим его EJB.

3.5.4 Реализация сервиса

Реализация сервиса будет выполнена в виде EJB - AttachmentServiceImpl

Функции работы с ДО Вложения должны быть делегированы `DomainObjectDao`

а функции работы с файлами Вложений — `AttachmentContentDao.`

Реализация AttachmentContentDao для файловой системы — **`FileSystemAttachmentContentDaoImpl.`**

Реализация метода saveAttachment будет сохранять контент Вложения, сохранять доменный объект Вложение. В случае отката транзакции нужно удалять сохраненный контент вложения на файловой системе. Т.е. этот случай нужно отдельно обрабатывать.

Так как операция загрузки файла может быть небыстрой (зависит от пропускной способности сети и размера файла), то нужно предотвратить возможность изменеия Вложения, которое в данный момент скачивается другим пользователем.

Поэтому предлагается следующая логика работы с файлами Вложений.

При апдейте существующего вложения создается новый файл в хранилище.

В пределах транзакции будет вестись журнал созданных файлов и файлов, которые следует удалить по успешному завершению транзакции.

При откате транзакции новые созданные файлы должны быть удалены, а старые нетронуты.

При успешном выполнении транзакции старые файлы, вместо которых были созданы новые, удаляются.

Реализация перехватчика конца транзакции и журнала файлов на удаление будет выполнена в рамках другой задачи (https://jira.inttrust.ru:8443/browse/CMFIVE-4)

Если удалить старый файл не удастся (т. к. он в данный момент читается другим пользователем), то этот файл так и останнется не удаленным, причем ссылка на него в доменном объектке пропадет. Для удаления таких файлов можно создать фоновую периодическую задачу.

Директория, куда будут сохраняться вложения, настраивается в конфигурационном фале deploy.properties. В этой корневой директории вложения будут группирваться по подпапкам год/месяц/день. Эта структура папок нужна, чтобы ускорить чтение/запись фалов на файловой системе. (Сейчас конфигурационные параметры специфичные для окружения приложения хранятся в файле build.properties. Этот файл нужно переименовать в deploy.properties).

При сохранени Вложения оригинальное название файла будет сохраняться в ДО Вложение, но на фаловой системе сохраненный файл должен иметь уникальное закодированное имя. Предлагается использовать генератор java.util.UUID.randomUUID().

3.6 Подсистема поиска

3.6.1 Область применения

К понятию «поиска» можно отнести несколько различных функций системы AF5:

1. поиск по полю (фильтрация) в списке документов;
2. простой поиск (по строке);
3. расширенный поиск (по атрибутам).

3.6.1.1 Поиск по полю в списке

В интерфейсе системы списки документов представляются таблицами. При этом в заголовках тех полей (столбцов), по которым поддерживается поиск, присутствует специальная кнопка, открывающая поле для ввода фильтра. При вводе значения в это поле осуществляется повторный запрос списка документов, учитывающий – дополнительно к основным условиям отбора – введённый фильтр.

Списки документов в интерфейсе системы настраиваются с помощью специальных конфигурируемых сущностей – коллекций. Существует 2 варианта настройки коллекций:

(1 – generic) указанием специального Java-класса, осуществляющего выборку объектов и (2 – specific) написанием запросов на псевдо-SQL (DOSQL), преобразуемым в настоящий SQL ядром системы. В обоих случаях поддерживается встраивание в запросы фильтров, которые могут модифицировать результирующий запрос. Эта возможность может быть использована для организации фильтров списка по значениям полей. Правда, в случае конфигурации коллекции через DOSQL написание запроса и всех фильтров к нему может быть сложной и трудоёмкой задачей.

Описанный выше подход оставляет фильтрацию списков за пределами ответственности подсистемы поиска. Его, однако, нужно будет пересмотреть, если для фильтрации необходимы функции, реализуемые исключительно подсистемой поиска – например, учёт морфологии русского языка.

3.6.1.2 Простой поиск

Строка для поиска документов присутствует на каждой странице системы – в шапке интерфейса. Тем не менее, при поиске учитывается настраиваемый параметр контекстной страницы – область поиска.

Область поиска определяет множество документов – доменных объектов на основании их типа, статуса и, возможно, других полей, а также набор полей (включая поля связанных объектов), в которых производится поиск заданной строки. Одним из важных требований к простому поиску является возможность поиска текста во вложенных файлах различных форматов (MS Word, Open Office, plain text и др.). Следует отметить, что результатом поиска является список документов – ключевых (корневых) объектов иерархии, олицетворяющих собой бизнес-сущности, а не доменных объектов, непосредственно содержащих удовлетворяющее поисковому запросу поле.

Области поиска также должно быть сопоставлено представление коллекции (collection-view), которое позволит отобразить результаты поиска в виде таблицы.

3.6.1.3 Расширенный поиск

Расширенный поиск во многом схож с простым, но добавляет некоторые возможности. Ссылка на форму расширенного поиска присутствует рядом со строкой простого поиска.

Во-первых, расширенный поиск позволяет выбрать любую область поиска независимо от контекста (хотя именно контекст определяет выбор по умолчанию), и даже несколько таковых одновременно. Набор полей формы расширенного поиска зависит от выбранной(ых) пользователем области(ей). В случае выбора нескольких областей поиска форма содержит только поля, присутствующие во всех выбранных областях. Возможность выбора несовместимых областей поиска блокируется.

Каждое поле формы поиска позволяет наложить ограничение на искомые документы по содержанию их конкретных полей. Тип фильтра при этом зависит от типа и семантики поля документа: для строковых полей фильтр будет содержать также строку для поиска по словам или по вхождению, для полей дат – диапазон дат, для полей, ссылающихся на справочники – набор подходящих значений этого же справочника. Также, как правило, присутствует поле для поиска текста в файлах вложений. Кроме того, форма расширенного поиска содержит поле «Искать везде» – аналог строки в простом поиске.

Наконец, форма расширенного поиска позволяет выбрать одно из нескольких возможных представлений коллекции найденных документов и ограничить размер выборки.

3.6.1.4 Области поиска

В форме расширенного поиска CM 4.x присутствует группа флажков под названием «Область поиска». Однако, реальные области поиска определяются сочетанием выбранных значений из этой группы со значением переключателя «Искомый объект». Именно это сочетание задаёт набор полей, по которым осуществляется поиск. Оно же определяет состав полей, возвращаемых для искомых объектов. Разумеется, не каждое сочетание области поиска и искомого объекта соответствует реальной области поиска, и выбор таких сочетаний блокируется пользовательским интерфейсом.

Поскольку переключатель «Искомый объект» позволяет выбрать только один вариант, это гарантирует, что возвращаемые поиском документы (доменные объекты) являются однотипными, независимо от того, сколько выбрано разных областей поиска. Даже для поиска собственно документов в различных областях (например, «Входящие документы», «ОРД») искомым объектом является «РКК» – единый для разных типов документов. Это будет работать и в CM, где РКК является базовым (родительским) типом для различных типов доменных объектов, представляющих входящие, исходящие и др. документы.

3.6.2 Архитектура

3.6.2.1 Apache Solr

Для реализации поиска в системе используется мощная поисковая платформа с открытым кодом Apache Solr. Она позволяет организовать эффективные поисковые индексы для быстрого выполнения поисковых запросов по большим массивам данных, поиск текста с учётом морфологии русского языка, индексирование файлов в различных форматах (включая MS Word, Open Office, Adobe PDF, HTML и др.). Solr может быть встроен в ядро системы, либо запускаться как отдельное приложение на том же сервере приложений или на другом, причём этот сервер может как на том же компьютере, так и на отдельном (выделенном) или даже кластере – решение остаётся за заказчиком и зависит от его потребностей и возможностей.

Подсистема поиска является, фактически, мостом (bridge) между Apache Solr и остальной частью платформы AF5. Она обеспечивает поставку данных для индексации в Solr, формирование поисковых запросов к этой платформе и интерпретацию результатов поиска в коллекции объектов. Для взаимодействия с Solr используется Java-библиотека SolrJ.

При разработке подсистемы, помимо её кода и необходимой конфигурации, должны быть разработаны требования и рекомендации по настройке Apache Solr. В этих рекомендациях должна быть особым образом отмечена необходимость ограничения доступа к Solr извне, так как он имеет доступ к значительной части данных системы, но не обеспечивает собственной аутентификации или авторизации пользователей.

3.6.2.2 Организация поисковых индексов

Для упрощения процедуры обновления поисковых индексов они организуются в соответствии со структурой доменных объектов системы: одному документу Solr соответствует один доменный объект системы. Это особенно важно для доменных объектов вложений, индексация которых может требовать передачи большого объёма данных. В специальном поле документа Solr сохраняется идентификатор доменного объекта системы. Кроме того, ещё в одном поле должен быть сохранён идентификатор документа системы, т.е. главного (корневого) доменного объекта иерархии. Это поле – ключевое для построения коллекции в ответ на поисковый запрос. Для эффективной

организации поиска по областям идентификатор области поиска также должен сохраняться в специальном поле документа Solr.

Помимо перечисленных служебных полей, в поисковые индексы Solr включаются поля доменных объектов, по которым может осуществляться поиск. Также могут быть сконфигурированы индексы (поля Solr), вычисляемые по содержимому полей связанных объектов (например, фамилия подписанта документа). Однако, автоматическое отслеживание изменений этих объектов для обновления индекса исходного объекта в настоящий момент не предусмотрено – в частности, потому, что это может потребовать слишком большого объёма вычислений. Поэтому такие поля должны использоваться только для редко изменяемых объектов (связанных) – таких, как объекты организационно-штатной структуры. Переиндексация

Следует отметить, что Solr может быть настроен на хранение либо только индексов полей, либо также их содержимого. Второй вариант может существенно увеличить объём данных, хранимых Solr, но позволит отображать в коллекции контекст найденных слов. Выбор варианта конфигурации может быть оставлен на усмотрение администратора системы.

3.6.2.3 Конфигурация

Базовым элементом конфигурации подсистемы поиска является определение областей поиска. Для каждой области поиска указываются типы доменных объектов, которые в неё попадают, а также набор полей, подлежащих индексированию.

Кроме того, необходима привязка области поиска к отображаемой в интерфейсе странице.

3.6.2.4 Графический интерфейс пользователя

Архитектура компонентов GUI, необходимых для организации поиска, не рассматривается в данном документе. Тем не менее, необходимо упомянуть несколько моментов, важных для организации взаимодействия подсистемы поиска с web-клиентом системы.

Важной особенностью GUI платформы AF5 является способность отображения динамических форм, состав полей которых определяется не кодом системы, а внешними данными – конфигурацией. Механизм динамических форм, использующийся для просмотра и изменения доменных объектов системы, может быть использован и для формирования поисковых запросов (расширенный поиск). С точки зрения архитектуры подсистемы поиска, это означает необходимость обеспечения интерфейсной общности между возможными подлежащими объектами динамических форм – доменным объектом и поисковым запросом, – чтобы любой из них мог использоваться как источник и приёмник данных формы.

3.6.3 Компоненты

3.6.3.1 Сервис поиска

Сервис поиска является EJB и входит в публичный API системы. Он содержит методы для выполнения простых и расширенных поисковых запросов. Все методы возвращают коллекцию IdentifiableObjects, каждый из которых представляет документ системы. Идентификатор объекта – это идентификатор главного доменного объекта,

соответствующего документу, и он содержит все поля, определённые в конфигурации области поиска. Нужно подчеркнуть, что независимо от того, где был найден искомый текст, в результатах поиска будут содержаться только идентификаторы главных объектов, а вот поля могут заполняться как из главного, так и из связанных с ним объектов. Если запросу поиска удовлетворяют несколько доменных объектов, связанных с одним главным, то он попадёт в результирующую коллекцию только один раз.

Сервис поиска формирует поисковый запрос, передаёт его Apache Solr, получает от него список найденных документов Solr, группирует их в соответствии с описанием области поиска, делает выборку коллекции из доменных объектов (фильтруя их по правам пользователя) и передаёт её клиенту.

Все методы сервиса поиска позволяют ограничить размер возвращаемой коллекции объектов. Поиск выполняется в синхронном режиме.

3.6.3.2 Агент обновления поисковых индексов

Компонент, отвечающий за актуальность поисковых индексов Solr, реагирует на события изменения (включая создание и удаление) доменных объектов системы. Он подключает свой код через точку расширения на сохранении и удалении доменных объектов. Проверив по конфигурации областей поиска, что сохраняемый объект подлежит индексации, компонент отправляет на сервер Solr запрос, содержащий все необходимые поля для добавления или обновления документа в индексе.

Предмет особого внимания этого компонента – доменные объекты вложений. Главной ценностью этих объектов является содержимое приложенного к ним файла, которое должно быть проиндексировано Solr. Компонент обеспечивает передачу в Solr ссылки на файл в случае их расположения на одном сервере и содержимого файла при использовании выделенного сервера Solr.

Индексация документов Solr осуществляется исключительно в асинхронном режиме.

3.6.3.3 Процесс переиндексации

Возможны ситуации, когда поисковые индексы Solr не соответствуют доменным объектам в БД:

- изменилась конфигурация областей поиска;
- изменился объект, поля которого используются в индексах связанных объектов;
- текущая индексация объекта не произошла по какой-либо причине;
- поисковый индекс потерян или испорчен из-за аппаратных сбоев.

В таких случаях требуется переиндексация всех или некоторых доменных объектов.

Эту задачу выполняет специальный компонент, осуществляющий выборку доменных объектов по заданным критериям, и передающий их для индексации в Solr. Компонент работает под управлением подсистемы запуска периодических задач.

3.7 Взаимодействие с Activiti

Процессный «движок» Activiti используется в системах на платформе AF5 для управления различными процессами, предполагающими взаимодействие между различными пользователями системы.

Основная идея использования бизнес-процессов Activiti заключается в том, чтобы прохождение документов в системе было максимально автоматизировано и требовало

минимальных усилий со стороны пользователей (отказ от ручных передач с этапа на этап). С помощью процессов можно настроить контроль сроков выполнения задач и рассылку уведомлений пользователям. Графический редактор бизнес-процессов представляет их в виде, удобном для понимания, а отчасти и для изменения аналитикам (не-программистам). Кроме этого, API Activiti позволяет легко построить административный интерфейс для контроля состояния бизнес-процессов в системе.

3.7.1 Технические аспекты

Каждый процесс привязан к карточке системы. Если процесс может менять статус карточки, то он называется управляющим процессом. К одной карточке в каждый момент времени может быть привязан только один управляющий процесс.

Идентификатор экземпляра управляющего процесса сохраняется в специальном поле карточки. По завершению процесса это поле очищается. Объект карточки делается доступным коду задач через переменные процесса.

Основные типы задач (активностей), из которых будут состоять процессы, это:

- UserTask – задачи, назначаемые пользователям (например, «Согласовать»). Activiti позволяет задавать в настройках задачи пользователей, которым будет назначена эта задача (в т.ч. через переменные процесса). Предполагается добавить собственные настройки, указывающие возможные варианты завершения пользователем задачи («Согласен», «Не согласен», «Согласен с замечаниями»). Назначенные пользователю задачи будут отображаться в специальной(ых) папке(ах), для каждой будут предложены кнопки в соответствии с вариантами завершения задачи. При выборе пользователем одного из этих действий код системы информирует Activiti о завершении задачи, и выполнение процесса продолжается.
- ServiceTask – задачи, выполняемые системой автоматически, некоторый Java-код (класс), указанный в настройках задачи. Предполагается разработать ряд стандартных «кубиков» - классов, выполняющих типовые действия, – чтобы составлять из них различные процессы. Такими действиями могут быть: изменение статуса карточки, вычисление полей, создание дочерней(их) карточки(ек), добавление карточки в папку и т.п. В некоторых случаях понадобятся также специальные классы, выполняющие некоторые нестандартные действия, например, регистрация документа, импорт документа из МЭДО и др.
- Sub-process call – вызовы других процессов. Планируется использовать для вызова стандартных процессов (как согласование, подписание) из макро-процессов, управляющих жизненным циклом документа в целом (например, внутреннего).

Возможно использование и других типов задач, предлагаемых Activiti:

- EmailTask – отсылка сообщений (например, уведомлений) по электронной почте.
- ScriptTask – возможно, простые задачи вроде изменения полей карточки документа будет проще описывать на скриптовом языке (JavaScript, Groovy), а не создавать собственный простой класс со сложными настройками.
- JavaReceiveTask – приостановка процесса до получения сигнала от Java-кода. В некоторых случаях может оказаться удобнее, чем посылка сообщений.

Создание процесса для карточки может производиться:

a) пользователем через вызов специального действия. Как и любое другое действие в системе, оно задаётся для определённого типа карточки и определённого статуса, а также является объектом назначения прав доступа;

b) системой в процессе обработки других действий пользователя или выполнения других процессов (включая периодические).

3.7.2 Процессы для обеспечения ЖЦ внутреннего документа

Таким образом, для реализации жизненного цикла внутреннего документа видится необходимым создание следующих процессов в Activiti:

1. Общий ЖЦ внутреннего документа: согласование (подпроцесс), подписание (подпроцесс), регистрация (ServiceTask), исполнение (включая рассмотрение – подпроцесс), передача в дело (Service или UserTask – по мере необходимости). Это управляющий процесс карточки внутреннего документа. Следует отметить, что в нём отсутствуют логические этапы подготовки документа и ознакомления. Первый из них слабо формализован и либо не ограничен по срокам и не требует контроля со стороны системы, либо такое ограничение наложено другой задачей (выданным поручением) и контроль сроков осуществляется в рамках той задачи. Что же касается ознакомления, то оно может проводиться в разные моменты ЖЦ документа (после регистрации) и инициируется пользователем, т.е. является просто самостоятельным неуправляющим процессом для карточки документа. Впрочем, если регламент работы с внутренними документами будет предусматривать автоматическую отправку документа на ознакомление при выполнении заданных условий, то, разумеется, будет целесообразным включить этот процесс в общий ЖЦ документа.
2. Согласование. Содержит схему процесса, позволяющую реализовать комбинированный (последовательный + параллельный) процесс согласования документа. При этом в документе для каждого согласующего создаётся специальная дочерняя карточка согласования, в которой он и будет указывать свою визу. Основная задача, входящая в этот процесс – UserTask для согласующего, для завершения которой ему необходимо заполнить форму визы. В зависимости от решения согласующих, процесс перед завершением установит статус документа в «Согласовано», либо в «Подготовка»/«Доработка».
3. Подписание. Процесс во многом похож на согласование, только основной UserTask предполагает выдачу формы «Подписание», также связанной с соответствующей дочерней карточкой документа, а статус документа, устанавливаемый при успешном завершении процесса – «Подписано».
4. Исполнение документа. Этот подпроцесс объединяет в себе процедуры наложения резолюций (рассмотрение документа), отправки их на исполнение и контроля исполнения. При этом для лица, рассматривающего документ (адресата или руководителя) представляется разумным создать карточку рассмотрения, по которой ему будет поставлена задача рассмотрения документа. Резолюции/поручения создаются в виде отдельных карточек в системе (привязанных к документу, конечно), и для каждой создаётся отдельный процесс исполнения поручения (см. ниже). Процесс исполнения документа ожидает завершения задачи рассмотрения и всех дочерних процессов, созданных в ходе выполнения этой задачи, устанавливает статус «Исполнен» или «Готов к списанию в дело» и завершается.
5. Исполнение поручения – управляющий процесс для карточки поручения/резолюции. Включает UserTasks для исполнителя поручения

(исполнить) и контролёра (утвердить отчёт об исполнении), ServiceTasks для изменений статуса карточки поручения, рассылки уведомлений. С помощью таймерных событий можно контролировать сроки исполнения поручения (например, для рассылки дополнительных уведомлений). Процесс исполнения поручения может также порождать собственные копии в качестве подпроцессов для реализации процедуры исполнения дочерних (связанных) поручений.

6. Ознакомление. Один из самых простых процессов, главная часть которого – UserTask для лиц, которые должны открыть документ и нажать кнопку «Ознакомлен». Вопрос о том, должен ли этот процесс включаться в качестве подпроцесса в общий ЖЦ документа, или остаться отдельным неуправляющим процессом для карточки внутреннего документа, подробно рассматривался выше.

3.7.3 О карточках и статусах

Один из важных моментов, который необходимо понимать в архитектуре AF5, это то, что понятие карточки (доменный объект платформы, ДОП) не совсем соответствует понятию РКК в предметной области. Карточками в системе представляются и РКК, и различные другие объекты, включая достаточно мелкие. Например, к карточке внутреннего документа привязана масса дочерних карточек, таких как карточки согласования (по одной на каждого согласующего и каждую итерацию согласования), подписания, ознакомления, поручений и т.п. Кроме того, по причине отсутствия полей со множественными значениями они также заменяются дочерними мини-карточками.

Каждая карточка имеет обязательное поле статуса, главное назначение которого – определение прав доступа к ней. Изменение поля статуса через операцию сохранения карточки невозможно, для этого в API предусмотрены отдельные операции. Главным инициатором выполнения этих операций должны быть задачи процессов Activiti. Для некоторых переходов можно также создать бизнес-действия, которые будут доступны пользователям в соответствии с правами доступа. Однако, важно не допускать неявного подключения сложного кода, реагирующего на такие «ручные» перехода, чтобы логика системы оставалась понятной и хорошо поддерживаемой.

Поскольку некоторые (но не все!) события смены статуса документа представляют особый интерес с точки зрения его истории, целесообразно предусмотреть возможность протоколирования таких переходов в виде отдельных дочерних карточек (истории).

3.7.4 Концепция использования Activiti в AF5

Целью интеграции Activiti в платформу AF5 является обеспечение возможности использования данного движка для решения задач маршрутизации и обработки доменных объектов в платформе AF5, удобство написания и понятность процессов в нотации BPMN2. Для этого движок Activiti внедрен в платформу AF5, написана обертка движка для удобства работы с ним, а также создана необходимая инфраструктура для удобного взаимодействия Activiti и остальных сервисов платформы, написаны тестовые процессы, охватывающие максимальное количество механизмов, которые потребуются при реализации реальных бизнес процессов в будущих системах на AF5.

Задачи, решаемые с помощью Activiti:

1. Маршрутизация и обработка доменных объектов, смена статуса, изменение атрибутов доменных объектов, создание и удаление доменных объектов.
2. Формирование задач в папке «Задачи» пользователей с возможностью сортировки и фильтрации.

3. Формирование элементов управления на карточках доменных объектов для выполнения действий доступных конкретному пользователю на данном этапе работы процесса.
4. Отправка уведомлений о поступление новой задачи по электронной почте.

Способ использования Activiti:

1. Движок Activity встраивается в ядро AF5
2. Данные Activiti хранятся в той же базе данных, что и данные AF5
3. Activiti использует информацию о пользователях и группах из AF5
4. Реализовываются spring бины для наиболее часто выполняемых операций, для упрощения процесса создания workflow
5. Доступные действия для каждого UserTask формируются с помощью заранее оговоренного имени переменной, в которой через запятую хранятся возможные действия. Имя переменной должно содержать имя активности, для исключения влияния друг на друга нескольких одновременно активных UserTask. Например NEGOTIATION_ACTIONS или EXECUTION_ACTIONS, где строка до нижнего подчеркивания имя активности, строка _ ACTIONS константа, по которой находится нужная переменная. Данная переменная устанавливается в процессе в автоматической активности, предшествующей ручной активности. В результате действия пользователя устанавливается в true переменная, имя которой так же заранее оговорено, И содержит имя активности, например NEGOTIATION_RESULT или EXECUTION_RESULT. Значение этой переменной анализируется в gateway расположенном за ручной активностью.
6. Идентификатор связанного с процессом документа хранится в специально именованной переменной, например – DOCUMENT_ID.
7. Для доступа из процесса к документу, привязанному к процессу, необходимо создать специальный спринг-бин, доступ к которому можно осуществлять из описания процесса с помощью выражений (expression). Например доступ к полю статус – «#{documentServise.getDocument(id).getAttribute(“status”)}», доступ к полю связанного документа - «#{documentServise.getDocument(DOCUMENT_ID).getAttribute(“author.full_name”)} ».
8. Для облегчения работы с документом в начале процесса необходимо определить переменную «DOCUMENT» и инициализировать ее с помощью следующего выражения - #{documentServise.getDocument(DOCUMENT_ID)}, далее в процессе доступ к документу можно осуществлять следующим выражением - DOCUMENT.getAttribute(“status”) или DOCUMENT.getAttribute(“author.full_name”) в случае связанных полей.
9. Модификации документа производить в скриптовых активностях с помощью выражений DOCUMENT.setAttribute(“description”, “any_string”); DOCUMENT.save();
10. К процессу могут быть привязаны не только документы, но и вспомогательные карточки, например карточка резолюции, карточка рассматривающего, карточка согласующего. Для процесса нет разницы какая карточка привязана, работа производится с доменным объектом, идентификатор которого указан в переменной DOCUMENT_ID. Из GUI, представления задачи поднимается карточка привязанного к процессу документа, и на карточке формируются доступные действия, полученные из переменной ACTION_NAME_ACTIONS (необходимо

продумать где хранить иконки и русские названия соответствующих элементов управления, я думаю что строка из ACTION_NAME_ACTIONS это имя соответствующих ресурсов в GUI).

3.7.5 Общеиспользуемые классы, реализующие общие операции в автоматических активностях

Перечень классов, и их описание.

Имя класса	Описание	Поля
InitProcess	класс, необходимый для инициализации общих переменных во всех процессах. Инициализирует переменную DOCUMENT	**variables** – Поле класса, инициализируемого списком переменных, устанавливаемых в процессе. Формат поля: var_name1=var_value1; var_name2=var_value2;…; var_nameN=var_valueN;
SetStatus	Класс, устанавливающий статус документа	**status** – Новый статус документа

3.7.6 Обоснование способа интеграции Activiti в AF5

3.7.6.1 Изучение движка Activiti

На данном этапе произведено глубокое изучение движка, интеграция его в платформу AF5, написание внутреннего API и тестовых процессов, функциональные и нагрузочные испытания, выявдены слабые стороны и пути их обхода или исправления.

При встраивании движка необходимо максимально облегчить работу с ним для его пользователей – разработчиков процессов. Для этого всю рутинную работу желательно вынести за пределы шаблона процесса, а так же при возможности обеспечить возможность взаимодействия с остальными сервисами платформы на понятном, лаконичном языке формул и выражений, работа с которыми не требует квалификации программиста.

Перечень наиболее предпочтительных функций окружения движка Activity в платформе:

1. При запуске процесса инициализировать переменную DOMAIN_OBJECT по идентификатору, который передается в функцию запуска процесса. Объект, на который ссылается переменная DOMAIN_OBJECT позволяет получать атрибуты доменного объекта и менять их.
2. В выражениях желательно использовать упрощенный синтаксис обращения к доменному объекту. Например, чтобы получить наименование вместо DOMAIN_OBJECT.getAttribute(“name”), желательно использовать выражение DOMAIN_OBJECT.name. Для реализации данного функционала изучить возможности интеграции в Activiti спринг библиотеки SpEL.

3.7.6.2 Вопросы по встраиванию и использованию Activiti

Есть ли необходимость создания доменного объекта задача?
Да, необходимость есть. Наличие такого объекта позволит стандартными способами вести историю, настраивать коллекции, назначать права, создавать карточки в интерфейсе.

Есть ли необходимость создания доменного объекта задача для каждого типа задачи?
Нет. Набор полей в карточке задачи фиксирован и фиксация в документации имен полей задачи движка и соответствующих им имен полей в доменном объекте позволит формировать практические любые интерфейсы в GUI.

Каким образом на интерфейсе карточки задачи или карточки документа отображается список доступных для данной задачи (задачи по данному документу) действий и каким образом в процесс передается результат этого действия?

- Вариант 1. В объекта задача движка фиксируется имя поля, в котором через запятую перечисляются возможные действия для данной задачи. В интерфейсе данное поле зачитывается и формируются элементы управления в виде кнопок или выпадающего списка с вариантами действий. При выборе одного из действий имя этого действия сохраняется в другое предопределенное поле задачи движка Activiti. Далее это поле анализируется в процессе.
- Вариант 2. На уровне конфигурации GUI к карточке задачи привязываются действия (Action). Состав этих действий зависит от значения определенных полей доменного объекта задача и его статуса. При выборе одного из действий имя этого действия сохраняется в предопределенное поле задачи движка Activiti. Далее это поле анализируется в процессе.
- Вариант 3. К процессу так или иначе привязан доменный объект (например объект рассмотрение). При формировании списка задач основным идентификатором является идентификатор доменного объекта. Действия привязываются на уровне конфигурации к данному доменному объекту. При выполнении тех или иных действий меняются поля данного доменного объекта, и затем эти поля анализируются в процессе. При этом в настройке действий их доступность определяется статусом доменного объекта.

Выбираем вариант 1 с небольшими дополнениями. Список действий это не произвольный набор строк, а перечисление через запятую имена действий в конфигурации Actions. Фиксируется 3 поля:
ATTACHMENT_ID – идентификатор прикрепленного к процессу основного оменного обьекта;
ACTIONS – список имен действий;
TASK_RESULT – выбранное пользователем действие.

Каким образом на карточке задачи формируется список присоединенных к процессу документов или как карточка документа определяет список задач по этому документу?

В объекте задача движка Activiti формируется поле с предопределенным названием ATTACHMENT_ID, содержащее идентификатор основного документа привязанного к процессу. Это может быть идентификатор входящего документа или идентификатор карточки рассмотрения.

3.7.7 Описание интеграции Activiti в ядро

Активити интегрировано в ядро как набор спринговых бинов. Необходимые таблицы в базе данных создаются автоматически при страте сервера приложений.

Для взаимодействия с activity создан EJB - ru.intertrust.cm.core.business.api.ProcessService. Ниже следует описание методов интерфейса:

Modifier and Type	Method and Description

void	**completeTask**(**Id** taskId, **List**<**ProcessVariable**> variables, **String** action) Завершить задачу
String	**deployProcess**(byte[] processDefinition, **String** processName) Установка шаблона процесса
List<**DeployedProcess**>	**getDeployedProcesses**() Получение информации об установленных процессах
List<**DomainObject**>	**getUserDomainObjectTasks**(**Id** attachedObjectId) Получение доступных задач для юзера и определенного доменного обьекта
List<**DomainObject**>	**getUserTasks**() Получение доступных задач для юзера
String	**startProcess**(**String** processName, **Id** attachedObjectId, **List**<**ProcessVariable**> variables) Запуск процесса
void	**terminateProcess**(**String** processId) Остановка процесса
void	**undeployProcess**(**String** processDefinitionId, boolean cascade) Удаление шаблона процесса(каскадно или нет).

Установка процесса производится с помощью вызова метода **deployProcess.** Методу передается массив байт описания процесса и его имя. Имя может быть произвольным, но должно заканчивается символами .bpm.

Старт процесса производится методом startProcess. Методу передается имя процесса, которое определено в шаблоне процесса (не путать с processName которое передается при инсталляции). Так же процессу передается идентификатор основного приаттаченного доменного объекта (опционально) и набор произвольных атрибутов (так же опционально). При передаче идентификатора основного доменного объекта в процессом создается переменная **MAIN_ATTACHMENT_ID**, и в нее записывается идентификатор процесса. Так же при передаче идентификатора основного приаттаченного объекта, в процессе создается переменная **MAIN_ATTACHMENT** инициализированная объектом типа ru.intertrust.cm.core.tools.DomainObjectAccessor. Эта переменная служит для более удобного доступа к атрибутам доменного объекта, возможности получить атрибуты, установить атрибуты а так же выполнить сохранение. Например, получить атрибут можно в активности типа скрипт следующей строкой MAIN_ATTACHMENT.get(“Name”). Описание всех методов класса ru.intertrust.cm.core.tools.DomainObjectAccessor будет приведено ниже. Так же в каждом экземпляре процесса создается переменная **SESSION** типа ru.intertrust.cm.core.tools.Session. Данный объект позволяет более удобно работать с сервисами из скриптовых активностей. Например для создания доменного объекта определенного типа необходимо вызвать метод SESSION.create(“type_name”).

При формировании задачи пользователю в Activiti автоматически создается доменный объект типа **Person_Task**. Дальнейшая работа с задачами производится с помощью существующих сервисов. Адресат задачи сохраняется в специальных доменных объектах типа **Assignee_Person** и **Assignee_Group** в зависимости пользователь или группа указаны в поле assignee задачи.

При формировании задачи в шаблоне процесса можно определить действия, которые в дальнейшем могут отобразиться. Действия определяются в поле ACTIONS формы, привязанной к задаче. При формировании доменного объекта (типа Person_Task) возможные действия копируются в поле Actions в формате name1=label1;… nameN=labelN.

При завершении задачи в метод completeTask надо передать идентификатор доменного объекта задача и имя действия из поля Actions. Далее идентификатор выполненного действия копируется в переменную, указанную в атрибуте variable поля формы ACTIONS. Пример описания задачи с действиями.

```
    <userTask id="usertask3" name="Проверка возможности маршрутизации с
помошью акшенов"
activiti:assignee="#{personManagement.getPersonId(&quot;admin&quot;).toString
Representation()}">
      <documentation>Получить список действий и выполнить одно из
них.</documentation>
      <extensionElements>
        <activiti:formProperty id="ACTIONS" name="Действие" type="enum"
variable="RESULT">
          <activiti:value id="YES" name="Да"></activiti:value>
          <activiti:value id="NO" name="Нет"></activiti:value>
        </activiti:formProperty>
      </extensionElements>
    </userTask>
```

В данном примере доступные действия YES и NO и после завершения задачи выполненное действие (значение параметра action метода completeTask) скопируется в переменную процесса RESULT, и в дальнейшем эта переменная может использоваться в написании логики процесса, например в условии потока. Например:

```
    <sequenceFlow id="flow-now" sourceRef="exclusivegateway1"
targetRef="scripttask2">
      <conditionExpression
xsi:type="tFormalExpression"><![CDATA[${RESULT.equals("NO")}]]></conditionExp
ression>
    </sequenceFlow>
```

Особенность текущей реализации activity такова, что не работает механизм автоматического создания переменных процесса, при их объявлении в javaScript. Для избежания данных ошибок необходимо при формировании ScriptTask устанавливать атрибут autoStoreVariables в false. Пример SctiptTask:

```
    <scriptTask id="scripttask2" name="Установка данных по действию НЕТ"
scriptFormat="javascript" activiti:autoStoreVariables="false">
      <script>var test =MAIN_ATTACHMENT.get("test_text");
test += " Результат маршрутизации НЕТ.";
MAIN_ATTACHMENT.set("test_text", test);
MAIN_ATTACHMENT.save();
</script>
    </scriptTask>
```

3.7.8 Описание интеграции Activiti-explorer в платформу

Для интеграции activiti-explorer необходимо:

- Скачать архив activiti http://www.activiti.org/download.html

- Из архива скопировать activiti-5.x\wars\activiti-explorer.war в jbossAsDir\standalone\deployments\ear.ear
- Изменить файл jbossAsDir \standalone\deployments\ear.ear\META-INF\application.xml. Добавить:

```
<web>
     <web-uri>activiti-explorer.war</web-uri>
     <context-root>/activiti-explorer</context-root>
   </web>
```

- В activiti-explorer.war изменить WEB-INF\ activiti-standalone-context.xml. Заменить строки:

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
  <property name="driverClassName" value="${jdbc.driver}" />
  <property name="url" value="${jdbc.url}" />
  <property name="username" value="${jdbc.username}" />
  <property name="password" value="${jdbc.password}" />
  <property name="defaultAutoCommit" value="false" />
</bean>
```

на:

```
<jee:jndi-lookup id="dataSource" jndi-name="java:jboss/datasources/CM5" />
```

- Перезапустить сервер
- Тестовый пользователь kermit\kermit

3.8 Уведомления

Главной чертой подсистемы уведомлений в системе CompanyMedia версий 4.x является её тесная взаимосвязь с бизнес-логикой документооборота, с механизмами контроля исполнительской дисциплины. В рамках разработки уведомлений для AF5 представляется разумным провести разделение функций между платформой и Сочи-сервером. Платформа должна взять на себя задачи, универсальные для большинства информационных систем различного назначения. К таким задачам можно отнести:

- формирование уведомлений по шаблонам при различных событиях в системе (в первую очередь, связанных с изменением данных);
- рассылку уведомлений по различным каналам, включая внешние (e-mail, SMS, мгновенные сообщения) и внутрисистемные (создание специальных доменных объектов, всплывающие окна в GUI);
- форматирование и локализацию сообщений;
- фильтрацию и, возможно, агрегирование сообщений в соответствии с системной политикой и настройками пользователя.

Последние три функции (рассылку, форматирование и фильтрацию) предполагается возложить на специальную службу уведомлений. Важно отметить, что служба уведомлений не является активным компонентом, т.е. инициатором посылки уведомлений. Как и любой другой сервис, она выполняет свою работу только в ответ на запросы от других частей системы или её клиентов.

Формирование уведомлений, т.е. определение того, кому, когда и какие уведомления должны посылаться, является функцией бизнес-логики системы. Платформа предоставляет несколько способов реализации бизнес-логики.

1. Конфигурация. Она позволяет использовать стандартные компоненты, входящие в платформу, и реализующие некоторые обобщённые функции, необходимые для информационных систем различного назначения. С точки зрения уведомлений, наиболее типичными (и необходимыми для системы электронного документооборота) являются посылка сообщений при некоторых изменениях документа лицам, связанным с этим документом определённым образом, а также напоминания о событиях намеченных на некоторую дату или время. Подсистема уведомлений в платформе AF5 должна включать компоненты, выполняющие эти задачи и настраиваемые через конфигурацию.
2. Бизнес-процессы (BPMN). Отправка уведомления – одна из наиболее распространённых задач (task), включаемых в бизнес-процесс. Платформа должна включать компонент, реализующий эту задачу (task) с использованием службы уведомлений.
3. Бизнес-модули (Java-код). Это наиболее гибкий, но и наиболее трудоёмкий вариант реализации бизнес-логики системы. Он используется в тех случаях, когда бизнес-логика не может быть реализована с использованием других методов. Бизнес-модули могут обращаться непосредственно к службе уведомлений, поэтому дополнительный компонент в платформе для этого сценария не требуется.

Таким образом, в платформу включаются только компоненты, реализующие механизмы генерации и рассылки уведомлений. Конфигурация событий и шаблонов уведомлений, а также бизнес-процессы и бизнес-код, содержащие обращения к службе уведомлений, помещаются в бизнес-модули, реализующие функционал конкретной информационной системы (например, СЭД CompanyMedia).

3.8.1 Важные аспекты уведомлений

3.8.1.1 Каналы доставки

Уведомления могут доставляться пользователям различными способами. Среди них могут быть как способы, требующие привлечения внешних ресурсов (e-mail, SMS, мгновенные сообщения), так и чисто внутрисистемные (создание специальных доменных объектов, всплывающие окна в GUI). Набор каналов может отличаться от системы к системе и даже от установки (installation) к установке. Некоторые каналы могут поставляться в виде отдельных модулей или разрабатываться специально для конкретного заказчика.

3.8.1.2 Управление доставкой

Выбор канала доставки уведомления может зависеть как от самого сообщения (его типа и важности), так и от настроек пользователя или системной политики. Одно и то же уведомление может доставляться разными способами в разных случаях (например, всплывающее окно, если пользователь вошёл в систему, и e-mail в ином случае), и может отправляться несколькими способами одновременно. Кроме того, зачастую полезно предоставить возможность пользователю с помощью персональных настроек выбирать каналы доставки уведомлений и необходимость их отправки в зависимости от типа. При этом нужна возможность ограничивать выбор пользователя некоторой системной политикой, задаваемой администратором.

3.8.1.3 События

Теоретически, уведомления могут отправляться в самые разные моменты времени. На практике же подавляющее большинство уведомлений формируются при изменении данных в системе. Поскольку все бизнес-данные платформа хранит в виде доменных объектов, наиболее важным для подсистемы уведомлений является событие изменения (сохранения) доменного объекта. При этом, как правило, уведомление конкретного типа формируется при соблюдении следующих условий:

1. изменилось значение одного или несколько конкретных полей объекта;
2. значения некоторых полей этого или связанных объектов удовлетворяют определённым условиям (равно, не равно, больше, меньше, пустое, не пустое и т.п.; аналогичные проверки могут быть на количество связанных объектов того или иного типа).

3.8.1.4 Содержание

Содержание уведомления зависит, в первую очередь, от произошедшего события, и определяется бизнес-логикой. Однако, в силу различия возможностей разных каналов доставки, оформление уведомления может быть своим для каждого канала. Например, сообщение e-mail должно содержать осмысленную тему (subject) и тело в формате HTML, содержащее, помимо прочего, ссылку, позволяющую открыть изменившийся документ в GUI системы для просмотра или редактирования, тогда как SMS-уведомление состоит из одной строки и должно быть достаточно кратким. Кроме того, в мультиязычных системах уведомления должны отправляться на языке, удобном для конкретного пользователя.

Чаще всего текст уведомления должен включать значения некоторых полей изменившегося (или каким-то иным образом связанного с произошедшим событием) доменного объекта. Этот текст формируется на основе шаблона, который может задаваться администратором.

Таким образом, отдельный шаблон должен быть задан для каждой возможной комбинации трёх элементов:

- тип уведомления;
- канал доставки;
- поддерживаемый язык системы.

Для упрощения администрирования желательно поддерживать вариант по умолчанию для каждого из этих элементов.

3.8.1.5 Внешние адресаты

Основной пункт назначения уведомлений – это пользователь системы. Тем не менее, иногда возникает необходимость формировать уведомления для лиц, не являющихся её пользователями. По сути, эта задача почти не отличается от посылки уведомления пользователю системы, только объектом назначения является не пользователь системы, а некий другой доменный объект. Служба уведомлений должна поддерживать получение таких объектов в качестве адресатов, а каналы – уметь определять адрес назначения по этому объекту. Разумеется, не каждый канал может (и обязан) поддерживать внешних получателей.

3.8.2 Компоненты подсистемы уведомлений

3.8.2.1 Служба уведомлений

Служба уведомлений является одним из публичных сервисов (EJB) системы. Она принимает запросы на уведомление пользователей о каких-либо событиях. Параметрами запроса являются:

- пользователь, группа пользователей или контекстная роль – получатели уведомления;
- информация о событии, включающая его тип, категорию, важность, идентификатор документа(ов) и др.

В зависимости от содержания запроса, настроек пользователя и системной политики служба выбирает один или несколько каналов, которым передаёт запрос на выполнение уведомления.

Служба уведомлений обрабатывает запросы в асинхронном режиме.

3.8.2.2 Каналы доставки

Канал доставки уведомлений – это программный код, выполняющий отправку сообщений некоторым способом. Несколько каналов доставки (изначально один – e-mail) входят в состав ядра системы, но модули могут добавлять собственные каналы.

Для внешних каналов доставки необходима дополнительная информация о пользователе – адрес (для e-mail; номер телефона для SMS и т.п.). Определение адреса по пользователю является одной из задач канала при отправке уведомления. Также канал должен позаботиться о добавлении в конфигурацию элементов, необходимых для хранения нужного ему адреса пользователей, или хотя бы проверить их наличие и корректность.

3.8.2.2.1 E-mail

Канал доставки уведомлений по электронной почте входит в ядро системы. Он формирует сообщение e-mail с использованием шаблона сообщения и передаёт его внешнему почтовому серверу с использованием технологии Java Mail (код может использовать надстройку Spring Mail для работы с этим API). Шаблон выбирается по типу сообщения, а также языковым настройкам пользователя.

Данные, используемые каналом:

- ДО Person – e-mail пользователя.
- Профиль пользователя (ДО Person_Profile?) – предпочитаемый язык (опционально).
- Конфигурация – шаблон, соответствующий типу сообщения.

3.8.2.2.2 Генератор задач CMJ

Такой канал может быть разработан в составе модуля сопряжения платформы AF5 с Сочи-сервером. Он должен формировать объекты задач и помещать их в персональные коллекции данных соответствующих пользователей. Правила создания этих объектов и используемые при этом данные будут определены архитекторами проекта «Сочи».

3.8.2.2.3 Агрегирующий e-mail

Альтернативный способ доставки уведомлений через электронную почту – не отдельными сообщениями, но накапливая данные за некоторый временной интервал. Этот канал доставки может входить в состав ядра системы или поставляться в виде отдельного дополнительного модуля.

Канал, принимая запросы на отправку уведомлений, не формирует сразу сообщения и не отсылает их, а сохраняет в специальных доменных объектах. Также в состав канала включается компонент, запускаемый через механизм периодических задач, и выполняющий формирование сообщений по накопленным запросам уведомлений, отсылку этих сообщений пользователям, а также удаление сохранённых запросов. Следует отметить, что этот канал доставки не может использовать те же шаблоны, что и канал «e-mail», т.к. в одном почтовом сообщении будет присутствовать информация не об одном, а о нескольких событиях, возможно, даже разнотипных.

3.8.2.3 Компонент раскрытия шаблонов

Внутренняя служба платформы, предоставляющая функцию формирования текста по шаблонам. Основными клиентами службы являются каналы доставки, которым необходимо сформировать «красивое» сообщение, содержащее различные данные о произошедшем событии.

Шаблон сообщения – это текст, оформленный по тем же правилам, которым должно удовлетворять сформированное сообщение (HTML, XML, CSV, простой текст), но содержащий в себе определённым образом оформленные ссылки на данные уведомления или поля доменных объектов, указанных в этих данных. Поддержка нетекстовых (бинарных) форматов сообщений пока не планируется.

Необходимо определить язык задания шаблонов. В качестве вариантов можно рассмотреть Apache Velocity и FreeMarker. Крайне желательна также поддержка DOEL-выражений.

3.8.2.3.1 Формирование ссылок

Важным элементом уведомлений, в особенности, получаемых пользователем на устройствах, с которых возможен доступ к системе, являются ссылки, позволяющие открыть нужный документ в системе. Компонент раскрытия шаблонов обязан уметь создавать такие ссылки.

Как правило, для создания ссылки нужен идентификатор документа и некоторая информация, зависящая от экземпляра системы (доменное имя сервера, контекст приложения). В некоторых случаях может потребоваться также какая-то специфичная для пользователя информация (в распределённой системе – имя его «домашнего» сервера, например).

Ещё один аспект, который должен быть учтён, – перспектива наличия нескольких разных клиентов у системы, для каждого из которых требуется отдельная ссылка. При этом желательно дать пользователю возможность выбирать клиенты, ссылки на которые он хочет получать.

3.8.2.4 Генератор уведомлений

Генератор уведомлений предоставляет возможность настройки формирования уведомлений через конфигурацию. Он состоит из 2 компонентов, похожих по назначению, но различающихся по способу использования.

Каждый из этих компонентов является клиентом службы уведомлений.

3.8.2.4.1 Уведомления по событиям

Компонент генерации уведомлений по событиям активируется через механизм точек расширения при происхождении различных внутрисистемных событий, в первую очередь, изменениях доменных объектов.

Компонент управляется тегами <notification> в конфигурации. Каждый такой тег содержит 2 группы настроек:

- Фильтр событий: тип доменного объекта; список полей, изменение которых требует отправки уведомления; ограничения на значения полей этого или связанных объектов. Возможно также подключение дополнительного кода для сложной фильтрации событий.
- Правила формирования запроса в службу уведомлений: получатель(и); канал(ы) отправки; тип, категория и другие параметры уведомления.

При отправке уведомлений по событиям следует учитывать транзакционный характер событий в системе. Отправка уведомлений должна производиться только в случае успешного завершения транзакции.

3.8.2.4.2 Напоминания

Компонент генерации уведомлений-напоминаний работает не в ответ на какое-то событие в системе, а периодически. Он осуществляет выборку доменных объектов, по которым должны быть сгенерированы уведомления, и производит рассылку уведомлений.

Информация, необходимая для генерации напоминания одного типа, помещается в тег <reminder>:

- запрос на выборку доменных объектов (коллекция);
- параметры, передаваемые в службу рассылки уведомлений – аналогично генератору уведомлений по событиям.

Существенным отличием напоминаний от уведомлений по событиям является работа одновременно не с одним исходным доменным объектом, а с их коллекцией. Коллекции можно организовать по-разному: можно выбирать все объекты по некоторому бизнес-признаку (например, поле «Исполнить до» содержат текущую дату) и рассылать уведомления по каждому объекту в отдельности (формируя актуальный для него список получателей), а можно, организовав цикл по пользователям и использовав их id как параметр фильтра коллекции, выбирать интересующие объекты на персональной основе. Во втором случае легко сделать объединение нескольких напоминаний в одно уведомление. (Впрочем, этой же цели можно достичь и в первом случае, использовав канал доставки «агрегирующий e-mail».)

С точки зрения взаимодействия со службой периодических заданий можно рассмотреть 2 варианта реализации генератора напоминаний:

- отдельная задача на каждый тип уведомлений;
- единственная задача – диспетчер.

В первом варианте при старте системы после загрузки конфигурации некий код (менеджер) обеспечивает проверку наличия или создание объектов задач для каждого тега <reminder> в конфигурации. Он транслирует параметры настройки времени выполнения задачи из конфигурации в поля этих объектов, а часть XML с настройками параметров

уведомлений сохраняет в качестве конфигурации (настройки) этой задачи. Код же самой задачи, запускаемый под управлением службы периодических заданий, выполняет работу по выборке объектов и генерации уведомления.

Второй вариант предполагает наличие единственной задачи-диспетчера, вызывающейся достаточно часто (например, каждые 10 минут). При каждом запуске эта задача перебирает все теги <reminder>, отбирает по ним объекты и формирует уведомления.

Преимуществом первого варианта является возможность гибкой настройки графика формирования уведомлений для каждого типа напоминаний и даже конкретного пользователя. Кроме того, реализация этого варианта представляется более простой. Однако, большое количество периодических задач создаёт трудности при их администрировании, а их одновременный запуск может приводить к падению производительности системы. Если же идти вторым путём, то для реализации задачи-диспетчера придётся создать механизмы, похожие на те, что имеются в подсистеме периодических задач: задание и обработка расписания и учёт обработанных документов (по которым отправлены уведомления). Создание единственной задачи при этом обеспечит сама подсистема периодических задач, а генерация напоминаний – задача, как правило, низкоприоритетная – будет выполняться одним фоновым потоком. Кроме того, в рамках одной задачи можно организовать объединение разных напоминаний для одного пользователя в одном сообщении.

3.8.2.5 BPMN task

BPMN (Activiti) содержит стандартную задачу, выполняющую отправку e-mail – наиболее распространённый способ уведомления пользователей. Однако, для подсистемы уведомлений платформы e-mail – лишь один из способов отправки уведомлений. Кроме того, при использовании этой задачи код процесса должен сам сформировать текст сообщения, определить адрес пользователя, проверить его настройки по получению уведомлений данного типа и т.п. Поэтому необходима функциональная замена данной задачи (оформленная, видимо, как service task), осуществляющая отсылку уведомлений с использованием службы уведомлений платформы. Параметры этой задачи будут совпадать с параметрами метода службы. Инфраструктура бизнес-процессов позволит вычислять значения этих параметров из полей процесса или связанного с ним документа.

3.8.3 Реализация подсистемы уведомлений

3.8.3.1 Общие положения

Подсистема уведомлений состоит из следующих компонент:

1. Сервис уведомлений.
2. Набор каналов доставки.
3. Сервис формирования текста сообщения.
4. Сервис формирования ссылки.
5. Подсистема формирования уведомлений по событиям
6. Периодическое задание, формирующее уведомления по расписанию.
7. Задача подсистемы workflow для формирования уведомлений.

Схема подсистемы уведомлений предоставлена на рисунке 1.

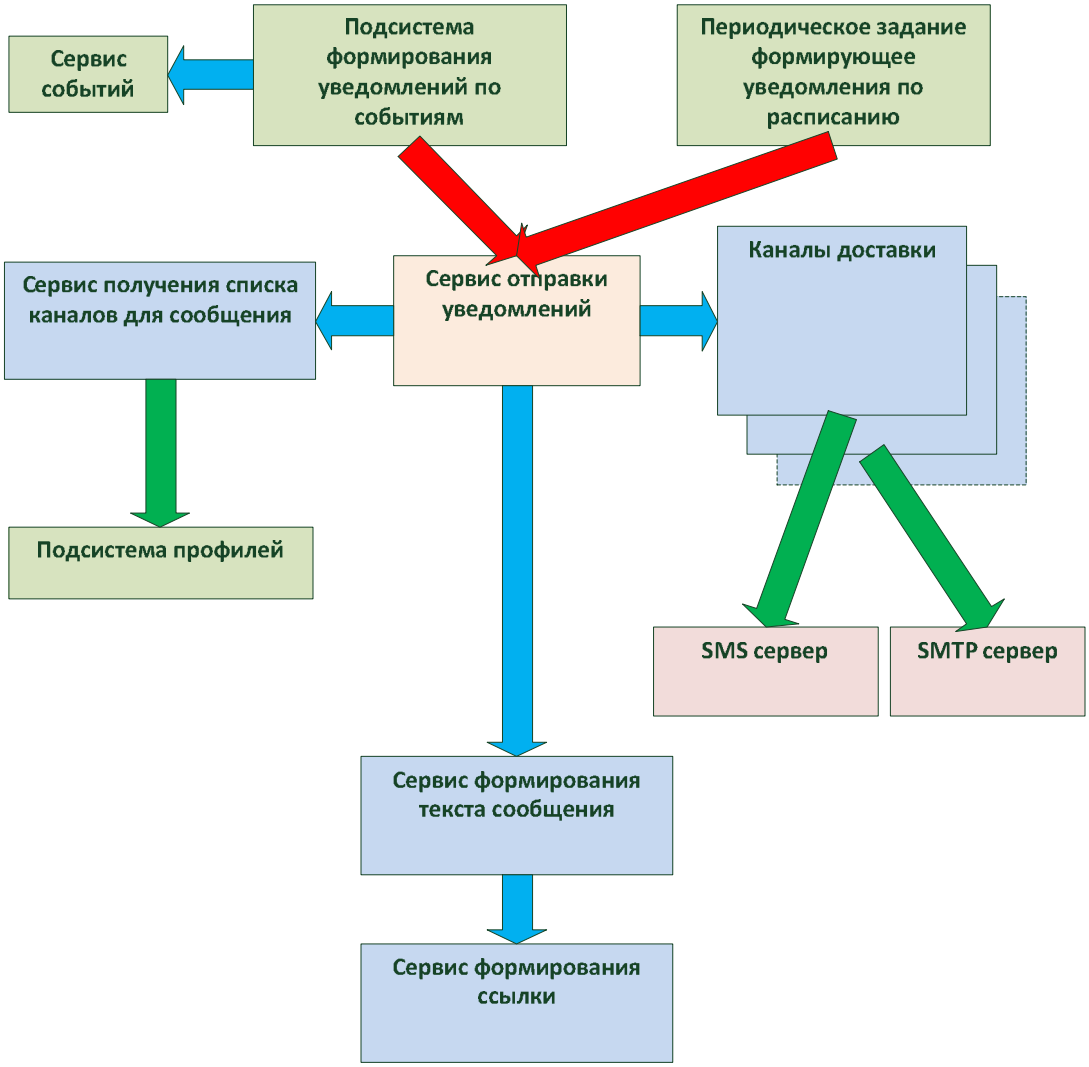


Рис. 1. Схема подсистемы отправки уведомлений. Красные стрелки обозначают использование подсистемы, Синие внутренние обращения, зеленые обращения системы к внешним сервисам.

3.8.3.2 Сервис уведомлений

Сервис уведомлений представляет собой EJB со следующим интерфейсом:

```
public interface NotificationService {

    /**
     * Отправка уведомления после успешного завершения транзакции. Метод
вызывается бизнес методами во время открытой транзакции. Реальная отправка
происходит после удачной завершенной транзакции
     * @param notificationType
     *            Тип уведомления
     * @param sender
     *            Идентификатор персоны отправителя, может быть nullБ в этом
случае подставится текущая персона
     * @param addresseeList
     *            список адресатов. Может быть персона, группа, динамическая
группа или контекстная роль
     * @param priority
     *            приоритет сообщения. Влияет на отображение данного
```

```
сообщения, но не влияет на очередность отправки
     * @param context
     *            Контекст сообщения. Содержит информацию о объекте системы,
относительно которой производится отправка
     *            уведомления
     */
    void sendOnTransactionSuccess(String notificationType, Id sender,
List<NotificationAddressee> addresseeList, NotificationPriority priority,
            NotificationContext context);

    /**
     * Метод асинхронной отправки уведомления. Используется из метода
sendOnTransactionSuccess.
     * Отправка производится независимо от результата транзакции.
     * @param notificationType
     *            тип сообщения
     * @param sender
     *            идентификатор персоны отправителя
     * @param addresseeList
     *            список адресатов
     * @param priority
     *            приоритет
     * @param context
     *            контекст сообщения
     * @return
     */
    Future<Boolean> sendNow(String notificationType, Id sender,
List<NotificationAddressee> addresseeList,
            NotificationPriority priority,
            NotificationContext context);
}
```

Особенность работы метода send такова, что он при выполнение формирует список сообщений в атрибутах транзакции, а реальное отправление осуществляется при успешном окончание транзакции при получение события beforeCompletion (Выбран метод beforeCompletion потому что есть подозрение что в методе afterCompletion ничего нельзя менять в базе, так как транзакция уже завершена, если это не подтвердится то необходимо использовать метод afterCompletion). Для этого необходимо применить сервис UserTransactionServiceImpl. Отправка в конце транзакции осуществляется в асинхронном режиме под правами системы путем вызова метода sendNow этого же EJB. Метод необходимо вызывать не напрямую из класса, а получив предварительно экземпляр EJB путем вызова метода SessionContext. getBusinessObject. Если при вызове метода send параметр sender равен null, то при вызове sendAsync передается параметр sender – идентификатор пользователя вызвавшего метод send. Данный идентификатор получается с помощью Сервиса CurrentUserAccessor и будет использоваться каналами для определения отправителя.

Адресатом сообщения могут быть следующие классы имплементирующие интерфейс NotificationAddressee: NotificationAddresseePerson, NotificationAddresseeGroup, NotificationAddresseeDynamicGroup, NotificationAddresseeContextRole. Метод sendNow раскрывает список адресатов в список персон и для каждой персоны получает список каналов, которые вычисляются исходя настройки каналов для соответствующей персоны и общих настроек системы. Список каналов для системы в целом и для отдельной персоны в частности хранятся в соответствующих профилях системы, которые будут описаны ниже. Далее сервис уведомлений вызывает метод send каждого полученного выше канала и передает в него параметры notificationType, personId, priority, context. Далее канал выполняет отправку сообщения исходя из собственной реализации.

Еще одной задачей сервиса уведомлений является сбор информации о всех каналах отправки уведомлений путем получения всех спринг бинов имплементирующих интерфейс NotificationChannel и создание реестра данных каналов по имени. Имя канала получается путем вызова метода NotificationChannel.getName.

При работе сервис определяет список каналов доставки конкретного сообщения с помощью спринг сервиса получения списка каналов для сообщения NotificationChannelSelector со следующим интерфейсом:

```
public interface NotificationChannelSelector {

    /**
     * Получение списка каналов для сообщения исходя из типа сообщения,
адресата и приоритета
     * @param notificationType
     *            тип сообщения
     * @param addressee
     *            идентификатор персоны адресата
     * @param priority
     *            приоритет сообщения
     * @return
     */
    List<String> getNotificationChannels(
            String notificationType,
            Id addressee,
            NotificationPriority priority);
}
```

Параметр notificationType – это строковой идентификатор сообщения. По этому идентификатору каналы получают у сервиса формирования текста сообщения непосредственно текст сообщения.

Параметр priority – приоритет, является экземпляром перечисления NotificationPriority и может принимать три значения HIGH, NORMAL, LOW. Код перечисления:

```
public enum NotificationPriority {
    HIGH,
    NORMAL,
    LOW
}
```

Приоритет используется каналами для установки визуального флага сообщения, а так же сервисом отправки уведомлений при получение списка каналов из профилей пользователя и системы.

Параметр context сообщения является экземпляром класса NotificationContext и содержит информацию о контекстных объектах для сообщения. Например, содержит информацию о доменном объекте документа, ссылку на который необходимо будет прикрепить к почтовому сообщению, или атрибуты которого необходимо будет отобразить в SMS сообщение. Контекст может содержать неограниченное количество идентификаторов доменных объектов и иных объектов, информация из которых может быть использована сервисом формирования текста сообщения. За содержание объектов в контексте и их имена отвечает код отправляющий сообщение. Уведомления одного типа всегда должны иметь одинаковое количество и одинаково названные объекты контекста. Код класса NotificationContext:

```
public class NotificationContext implements Dto{
```

```
    private Map<String, Dto> contextObjects = new Hashtable<String, Dto>();

    /**
     * Добавление контекстного объекта
     * @param name
     * @param object
     */
    public void addContextObject(String name, Dto object) {
        contextObjects.put(name, object);
    }

    /**
     * Получение контекстного объекта
     * @param name
     * @return
     */
    public Dto getContextObject(String name) {
        return contextObjects.get(name);
    }

    /**
     * Получение имен всех контекстных объектов
     * @return
     */
    public Set<String> getContextNames() {
        return contextObjects.keySet();
    }

}
```

3.8.3.2.1 Профили системы и пользователей

Подсистема профилей системы и пользователей не входит в состав подсистемы отправки уведомлений, но используется для получения списка каналов. Интерфейс подсистемы профиля:

```
public interface ProfileService {
    /**
     * Получение профиля системы. Профиль содержит данные профиля без учета
иерархии профилей. Предназначен для
     * редактирования системных профилей администраторами при его вызове
должен создаваться AdminAccessToken
     * @param name имя профиля
     * @return
     */
    Profile getProfile(String name);

    /**
     * Получения профиля персоны. Профиль содержит данные профиля без учета
иерархии профилей. Предназначен для
     * редактирования пользовательских профилей администраторами. При его
вызове должен создаваться AdminAccessToken
     * @param personId
     * @return
     */
    Profile getPersonProfile(Id personId);

    /**
     * Сохранения профиля системы. Профиль содержит данные профиля без учета
```

```
иерархии профилей. Предназначен для
     * редактирования системных профилей администраторами при его вызове
должен создаваться AdminAccessToken
     */
    void setProfile(Profile profile);

    /**
     * Получение пользовательского профиля. Профиль содержит данные профиля
пользователя с учетом иерархии профилей.
     * Предназначен для работы под провами простого пользователя
     * @return
     */
    PersonProfile getPersonProfile();

    /**
     * Сохранение пользовательского профиля. Профиль содержит данные профиля
пользователя с учетом иерархии профилей.
     * Предназначен для работы под провами простого пользователя.
     * При сохранения профиля меняются данные только профиля пользователя.
Данные системных профилей остаются не изменными.
     * @param profile
     */
    void setPersonProfile(PersonProfile profile);
}
```

Подсистема профилей предназначена для получения значения профиля по имени профиля для конкретного пользователя. Каждый пользователь имеет свой профиль со своими значениями. Например пользователь может выбрать язык, в котором ему необходимо присылать уведомления. Отличительной особенностью профиля в отличие от обычного доменного объекта являются:

- Неизвестное количество хранимых ключей. То есть каждая подсистема может хранить в профиле неограниченное значение профильных значений. Например подсистема GUI может хранить высоту и ширину окна и положения его на экране. Отсюда следует что в хранилище профилей не должно быть жестко зашито названия ключей профиля.
- Профили являются расширяемыми. То есть существует иерархия профилей. На самом нижнем уровне существует профиль пользователя. Профиль пользователя имеет родительский системный профиль, который в свою очередь так же имеет родительский. При получение пользовательского профиля сначала получается цепочка от самого нижнего до самого вышестоящего у которого атрибут родительский профиль не заполнен. Зачитываются атрибуты из профиля самого высокого уровня (корневого профиля) и полученные данные сохраняются в объекте PersonProfile. Далее тоже самое происходит по всей цепочки профилей с верху в низ, и на каждом этапе данные в PersonProfile дополняются данными из зачитываемого профиля, причем если атрибут профиля уже существует то более низкий профиль его перезаписывает. Так происходит до самого нижнего уровня, который является профилем пользователя.

Для хранения данных профиля следующие типы доменных объектов:

profile - базовый тип для профилей.

Имя	Тип	Описание
parent	Reference на тип profile	Идентификатор родительского профиля. В случае если атрибут равен null то запись является

		профилем родительского уровня.

system_profile – системный профиль. Является наследником типа PROFILE

Имя	Тип	Описание
Name	String	Имя профиля

person_profile –профиль пользователя. Является наследником типа PROFILE. Не имеет атрибутов. Выделен в отдельный тип для обеспечения ссылочной целостности и не возможности сослатся на тип system_profile или profile из типа person. На данный тип создается ссылка в типе person

Модификация типа **person**

Имя	Тип	Описание
profile	Reference на тип person_profile	Ссылка на профиль пользователя

3.8.3.3 profile_value - базовый тип для хранения одного значения профиля

Имя	Тип	Описание
profile	Reference на тип profile	Ссылка на профиль пользователя
key	String	Имя ключа профиля
readonly	Boolean	Флаг невозможности переопределить значение профиля конечным пользователем. Поля профиля помеченные данным флагом могу правится только администраторами. Пользователь не может переопределить в своем пользовательском профиле данный атрибут профиля, если в системном профиле атрибут помечен как readonly

profile_value_long – тип для хранения численных значений

Имя	Тип	Описание
profile	Reference на тип profile	Ссылка на профиль пользователя
key	String	Имя ключа профиля

value	Long	Значение профиля

profile_value_string – тип для хранения строковых значений

Имя	Тип	Описание
profile	Reference на тип profile	Ссылка на профиль пользователя
key	String	Имя ключа профиля
value	String	Значение профиля

profile_value_boolean – тип для хранения булевых значений

Имя	Тип	Описание
profile	Reference на тип profile	Ссылка на профиль пользователя
key	String	Имя ключа профиля
Value	Boolean	Значение профиля

profile_value_date – тип для хранения значений типа дата

Имя	Тип	Описание
profile	Reference на тип profile	Ссылка на профиль пользователя
key	String	Имя ключа профиля
value	Date	Значение профиля

profile_value_locale – тип для хранения значений типа ссылка на справочник локалей. Для хранения в профиле ссылочных полей, по мере необходимости, надо будет создавать подобные типы с разными типами на которые ссылаются поле value. Подсистема профилей должна создавтся с таким расчетом, что таких типов будет много и они должны будут зачитываться из конфигурации при старте системы. При сохранение в профиль значения типа ReferenceValue необходимо будет из конфигурации найти соответствующий сохраняемому типу доменный объект из конфигурации и создать его запись.

Имя	Тип	Описание
profile	Reference на тип profile	Ссылка на профиль пользователя
key	String	Имя ключа профиля
value	Reference на тип locale	Значение профиля

Работа с профилем осуществляется в двух режимах.

1. Режим администратора. В этом режиме используются методы getProfile, getPersonProfile(Id personId) и setProfile. При этом получаются и редактируются записи непосредственно полученного профиля без учета иерархии профилей.

Пользователь не входящий в группу администраторов при вызове данных методов должен получать ошибку.

2. Режим пользователя. Для этого предназначены методы getPersonProfile() и setPersonProfile(PersonProfile profile). При получение профиля методом getPersonProfile() зачитываются данные всех профилей в иерархии профилей находящихся выше по уровню относительно текущего пользовательского профиля. Данные всех профилей собираются в объект PersonProfile причем профили более низкого уровня перезатирают значения атрибутов профилей более высокого уровня, этим реализовывается возможность переопределения значений системных профилей. Если в профиле более высокого уровня установлен атрибут readonly то данное значение не может быть переопределено профилем пользователя. При сохранение пользовательского профиля сохраняются данные только в профиль пользователя. В случае если сохраняемое значение профиля пользователя отличается от значения в родительском профиле с учетом иерархии профилей то атрибут сохраняется в пользовательском профиле. Если атрибут не отличается от значения в родительском профиле то доменный объект хранящий данный атрибут удаляется из пользовательского профиля.

Для передачи данных о флаге readonly используются наследники класов StringValue, LongValue, DateValue, BooleanValue и ReferenceValue соответсвенно ProfileStringValue, ProfileLongValue, ProfileDateValue, ProfileBooleanValue и ProfileReferenceValue. Каждый из этих наследников должен имплементить интерфейс ProfileValue:

```
public interface ProfileValue {
    /**
     * Получение флага только для чтения
     * @return
     */
    boolean isReadOnly();

    /**
     * Установка флага только для чтения
     * @param readOnly
     */
    void setReadOnly(boolean readOnly);
}
```

3.8.3.4 Сервис получения каналов доставки использующий подсистему профилей

Ядро предоставляет одну реализацию сервиса NotificationChannelService, который использует подсистему профилей для получения списка каналов отправляемого сообщения.

Для работы данной реализации сервиса NotificationChannelService создаются 2 подтипа. Первый подтип используется для получения списка типа уведомлений которые не получает пользователь.

Имя	Тип	Описание
profileId	Reference	Идентификатор основного профиля
disableNotificationType	String	Тип сообщения которое не получает пользователь. Если

		нет ни одной записи пользователь получает все сообщения.

Второй подтип используется для настройки списка каналов в зависимости от приоритета уведомления.

Имя	Тип	Описание
profileId	Reference	Идентификатор основного профиля
notificationPriority	String	Приоритет сообщения, которое получает пользователь. Допустимые значения HIGH, NORMAL, LOW
channelName	String	Имя канала отправки уведомлений.

В качестве значения получается строка с значениями HIGH, NORMAL, LOW, DISABLE. Значения интерпретируются следующим образом: в случае значения профиля DISABLE канал не используется, в случае значения HIGH используется для уведомлений с приоритетом HIGH, в случае значения NORMAL канал используется для уведомлений с приоритетом HIGH и NORMAL, в случае значения LOW канал используется для всех уведомлений. В случае если для канала не указано ни одного значения в профиле то пользователь получает все уведомления по этому каналу (аналогично значению LOW).

Профиль так же используется сервисом формирования текста сообщения для получения языка текста сообщения для пользователя.

3.8.3.5 Набор каналов доставки

Канал доставки представляет из себя класс имплементирующий интерфейс NotificationChannelHandle и аннотированный с помощью анотации NotificationChannel.

```
public interface NotificationChannelHandle {

    /**
     * Отправка сообщения с помощью канала
     * @param notificationType
     *            тип сообщения
     * @param senderId
     *            идентификатор персоны отправителя. Может быть null в случае
если отправитель система
     * @param addresseeId
     *            идентификатор персоны адресата
     * @param priority
     *            приоритет
     * @param context
     *            контекст сообщения
     */
    void send(String notificationType, Id senderId, Id addresseeId,
NotificationPriority priority, NotificationContext context);
```

```
}
```

```
public @interface NotificationChannel {
    /**
     * имя канала
     * @return
     */
    String name();

    /**
     * Описание канала
     * @return
     */
    String description();
}
```

При старте сервера специальный сервис NotificationChannelLoader просматривает все классы аннотированные данной аннотацией и создает реестр каналов, ключом элемента реестра является строка возвращаемая методом name () аннотации. Интерфейс сервиса NotificationChannelLoader:

```
public interface NotificationChannelLoader {

    /**
     * Получение списка имен всех каналов отправки уведомлений
     * @return
     */
    List<String> getNotificationChannelNames();

    /**
     * Получение информации о канале доставки
     * @param channelName
     * @return
     */
    NotificationChannelInfo getNotificationChannelInfo(String channelName);

    /**
     * Получение обработчика канала отправки уведомления по его имени
     * @param channelName
     * @return
     */
    NotificationChannelHandle getNotificationChannel(String channelName);

}
```

Сервис отправки уведомлений вызывает метод send канала и передает информацию об уведомление каналу. Далее канал выполняет отправку уведомления по своему персональному алгоритму. Платформа в базовой поставке предоставляет два канала отправки уведомлений: канал папки «Входящие уведомления» и канал отправки по электронной почте. Все каналы могут пользоваться публичными сервисами платформы (EJB), внутренними сервисами платформы (DAO) в том числе сервисами формирования текста уведомления и сервисом формирования ссылок.

3.8.3.5.1 Канал отправки уведомления по электронной почте

Данный канал использует Spring Mail для отправки сообщений по электронной почте. Канал использует сервис формирования текста уведомлений для получения текста и

заголовка письма. Все настройки требуемые для обеспечения работы канала (smtp сервер, логин, пароль пользователя SMTP сервера, адрес отправителя по умолчанию) должны в server.properties. Канал разрабатывается таким образом, чтобы можно было его переиспользовать на другом сервере, передав задание на отправку уведомления с помощью сервиса распределенных задач, основанном на JMS.

3.8.3.5.2 Канал отправки уведомления в папку «Входящие уведомления»

Данный канал создает запись доменного объекта с типом notifications. GUI периодически опрашивает записи в таблице notifications и отображает их в интерфейсе. Тип содержит следующие атрибуты:

Имя	Тип	Описание
from	Reference	Идентификатор персоны отправителя сообщения
to	Reference	Идентификатор персоны адресата сообщения
subject	String	Заголовок сообщения
body	String	Текст сообщения
priority	String	Приоритет сообщения
new	Boolean	Флаг нового сообщения, после открытия данного сообщения GUI сбрасывает этот флаг в 0. Используется для выделения новых сообщений на интерфейсе и оптимизации запросов новых сообщений

Для формирования текста сообщения канал использует сервис генерации текста сообщения.

3.8.3.6 Сервис формирования текста сообщения

Сервис формирования текста сообщения представляет собой spring бин и предназначен для формирования текста сообщения по типу сообщения, идентификатору адресата и контексту. Шаблон сообщения хранится в специальном доменном объекте notification_text в виде скрипта на языке шаблонов Apache Velocity (язык шаблонов может быть изменен разработчиком подсистемы, при выборе необходимо опираться на максимальную простоту и понятность использования языка шаблонов). Интерфейс сервиса:

```
public interface NotificationTextFormer {

    /**
     * Метод формирует текст сообщения по типу сообщения и каналу
     * @param notificationType
     *            тип сообщения
     * @param notificationPart
     *            имя фрагмента сообщения. Используется если сообщение
состоит из нескольких частей, например почтовое
     *            сообщение состоит из заголовка и тела сообщения
     * @param addressee
     *            адресат сообщения
```

```
     * @param locale
     *            Идентификатор локали
     * @param channel
     *            имя канала
     * @param context
     *            контекст сообщения
     * @return
     */
    String format(String notificationType, String notificationPart, Id
addressee, Id locale, String channel,
            NotificationContext context);

    /**
     * Формирование всех частей текста уведомления по его типу
     * @param notificationType
     *            тип сообщения
     * @param addressee
     *            идентификатор персоны адресата
     * @param locale
     *            Идентификатор локали
     * @param channel
     *            имя канала
     * @param context
     *            контекст сообщения
     * @return
     */
    List<NotificationText> format(String notificationType, Id addressee, Id
locale, String channel,
            NotificationContext context);
}
```

Для хранения шаблонов сообщений сервис использует доменные объекты типа notification_text со следующими полями:

Имя	Тип	Описание
notification_type	String	Тип сообщения
notification_part	String	Имя фрагмента сообщения
locale	Reference	Идентификатор системного справочника языков
channel	String	Имя канала
notification_text	String	Текст сообщения на языке шаблонов сообщения

При формирование сообщения сервис получает запись notification_text по переданным параметрам. Далее создает контекст движка языка шаблонов и внедряет туда следующие объекты:

- Объект сессии ru.intertrust.cm.core.tools.Session под именем session.
- Объекты из параметра context, каждый объект под именем с которым этот объект добавляли в context. При этом все объекты типа DomainObject оборачиваются классом DomainObjectAccessor.
- Все спринговые бины платформы под именами с какими они зарегистрированы в Spring контексте платформы.

Далее выполняется скрипт на языке шаблонов и полученный результат возвращается. При формирование сообщения необходимо учитывать часовой пояс пользователя, которому отправляются сообщения при формирование дат со временем в тексте сообщения. Часовой пояс пользователя хранится в профиле пользователя.

Для работы сервиса используется системный справочник языков locale, который содержит следующие атрибуты:

Имя	Тип	Описание
name	String	Наименование языка

3.8.3.7 Сервис формирования ссылки

Сервис формирования ссылок представляет из себя spring бин со следующим интерфейсом:

```
public interface UrlFormer {

    /**
     * Формирует url к объекту системы. Формирование url производится в
контексте определенного клиента платформы.
     * @param clientName
     *            имя клиента
     * @param addressee
     *            идентификатор персоны адресата
     * @param objectId
     *            идентификатор доменного объекта
     * @return
     */
    URL getUrl(String clientName, Id addressee, Id objectId);
```

Сервис конфигурируется с помощью XML конфигурации следующего вида:

```
<url-config name="GWT-CLIENT">
    <![CDATA[Текст скрипта на языке разметки шаблонов]]>
</url-config>
```

При вызове метода getUrl ищется конфигурация name равным переданному параметру. У полученного доменного объект берется текст внутри тэга, которое передается движку языка шаблонов Apache Velocity (язык шаблонов может быть изменен разработчиком подсистемы, в первую очередь необходимо учитывать удобство написания конечных шаблонов). Так же в движок внедряются все спринговые бины системы под теми же именами как они зарегистрированы в spring контексте, и также, передается обертки DomainObjectAccessor над переданным параметром objectId и addressee под именами domainObject и addressee. Сформированная таким образом строка возвращается в качестве результата работы метода.

3.8.3.8 Подсистема формирования уведомлений по событиям

Платформа предоставляет три точку расширения для формирования уведомлений. Точка расширения реагирует на создание, смену статуса, изменение и удаление доменного объекта. Точка расширения имеет настройки, которые выносятся в конфигурацию системы на уровне ядра. Конфигурация точки расширения:

Тэг/атрибут	Тип	Описание

notification	Тэг	Тэг конфигурации верхнего уровня
name	Атрибут, String	Имя конфигурации
notification-type	Тэг, String	Тип уведомления
addressee	Тэг, FindObjectsConfig	Тег описывающий адресатов. Может содержать внутри теги получения адресатов с помощью запроса, с помощью контектной роли или контекстной группы, а также может быть указан класс, с помощью которого получается список адресатов.
find-person	Тэг, FindObjectsConfig	Тэг описывающий поиск персон с помощью запроса, вщуд или класса см. «Сервис поиска доменных объектов с помощью класса, запроса или DOEL выражения»
query	Тэг, String	Запрос получения адресатов. Принимает параметр текущего доменного объекта в параметре {0} Запрос возвращает список идентификаторов персон
class-name	Тэг, String	Имя класса определяющего список адресатов. Класс должен имплементировать интерфейс NotificationAddresseeReceiver
doel	Тэг, String	DOEL выражение определяющеее список адресатов. Класс должен имплементировать интерфейс NotificationAddresseeReceiver
context-role	Тэг, string	Имя контекстной роли относительно измененного доменного объекта
dynamic-group	Тэг, string	Имя динамической группы относительно измененного доменного объекта
trigger	Тэг	Описание события, по возникновению которого отправляются уведомления.

		Событие сожжет быть описано непосредственно здесь, или ссылается на именованное событие уровня конфигурации
name	Атрибут, String	Имя именованного события. Если атрибут заполнен то остальное содержание тэга игнорируется.
domain-object-type	Тэг, String	Тег содержит имя типа доменного объекта, при изменение которого возникает событие.
event	Тэг, String	Тег содержит тип события. Может быть CREATE, CHANGE, CHANGE_STATUS, DELETE
fields	Тэг	Тег описывающий имена полей, изменение которых приводит к возникновению события (проверяется для типа события CHANGE)
field	Тэг, String	Имя поля в списке полей, может быть множество полей. Если нет ни одного поля то при любом изменение доменного объекта возникает событие
statuses	Тэг	Тег описывающий имена статусов, при установки которых возникает событие (проверяется для типа события CHANGE-STATUS)
status	Тэг, String	Имя статуса в списке статусов
class-name	Тэг, String	Имя класса, отвечающего за проверку возникновения события. Класс должен имплементировать интерфейс Trigger
conditions-script	Тэг, String	JavaScript результат выполнения которого определяет возникновение события. В скрипт внедряется информация о доменном объекте и измененных полях

Для описания независимого события и возможности сослаться на него в конфигурации добавляется описание именованного события. Конфигурация именованного события:

Тэг/атрибут	Тип	Описание
named-trigger	Тэг	Тэг конфигурации верхнего уровня
name	Атрибут, String	Имя конфигурации
trigger	Тэг	Описание события, по возникновению которого отправляются уведомления. Событие сожжет быть описано непосредственно здесь, или ссылается на именованное событие уровня конфигурации
name	Атрибут, String	Имя именованного события. Если атрибут заполнен то остальное содержание тэга игнорируется.
type-name	Тэг, String	Тег содержит имя типа доменного объекта, при изменение которого возникает событие.
event	Тэг, String	Тег содержит тип события. Может быть CREATE, CHANGE, CHANGE_STATUS, DELETE
fields	Тэг	Тег описывающий имена полей, изменение которых приводит к возникновению события (проверяется для типа события CHANGE)
field	Тэг, String	Имя поля в списке полей, может быть множество полей. Если нет ни одного поля то при любом изменение доменного объекта возникает событие
statuses	Тэг	Тег описывающий имена статусов, при установки которых возникает событие (проверяется для типа события CHANGE-STATUS)
status	Тэг, String	Имя статуса в списке статусов

class-name	Тэг, String	Имя класса, отвечающего за проверку возникновения события. Класс должен имплементировать интерфейс TriggerService
script	Тэг, String	JavaScript результат выполнения которого определяет возникновение события. В скрипт внедряется информация о доменном объекте и измененных полях

Интерфейс EventTrigger

```
public interface EventTrigger {

    /**
     * Метод определяющий факт возникновения события
     * @param event
     *            тип события CREATE, CHANGE, CHANGE_STATUS, DELETE
     * @param domainObject
     *            длменный объект по которому произошло событие
     * @param changedFields
     *            измененные поля
     * @return Возвращается флаг сработал триггер или нет
     */
    boolean isTriggered(String eventType, DomainObject domainObject,
List<FieldModification> changedFields);

    /**
     * Получение списка имен триггеров сработавших на изменение доменного
объекта.
     * @param domainObject
     *            длменный объект по которому произошло событие
     * @param changedFields
     *            измененные поля
     * @return Возвращается список сработавших триггеров, если не сработал не
один триггер возвращается пустой список
     */
    List<String> getTriggeredEvents(DomainObject domainObject,
List<FieldModification> changedFields);
}
```

Интерфейс NotificationAddresseeReceiver

```
public interface NotificationAddresseeReceiver {

    /**
     * Метод возвращает список адресатов
     * @param domainObject доменный объект по которому произошло событие
     * @return
     */
    List<NotificationAddressee> getNotificationAddressee(DomainObject
domainObject);
}
```

XSD тип тэга trigger в теге named-trigger и notification является одним и тем же XSD типом. Для определения условия возникновения события необходимо создать спринг сервис (Сервис событий), принимающий на вход конфигурацию триггера и измененный доменный объект. Данный сервис будет использоваться как сервисом формирования уведомлений, так и иными сервисами, реагирующие на события. Интерфейс сервиса:

```
public interface TriggerService {

    /**
     * Метод проверки возникновения события
     * @param event
     *            тип события
     * @param domainObject
     *            доменный объект
     * @param changedFields
     *            измененные поля
     * @return
     */
    boolean isTriggered (String eventType, DomainObject domainObject,
List<FieldModification> changedFields);

}
```

Точки расширения при возникновение события проверяет конфигурацию уведомлений и при совпадение всех условий описанных в тэге trigger отправляет уведомление с типом notification-type пользователям описанным в тэге addressee. В контекст сообщения добавляется текущий измененный доменный объект. Отправляют уведомления полученным персонам с помощью сервиса NotificationService.

3.8.3.9 Периодическое задание, формирующее уведомления по расписанию

Для отправки уведомлений по расписанию в платформе реализовано периодическое задание для отправки уведомлений согласно настройке этого периодического задания. Данное задание является периодическим заданием платформы то есть имплементирует интерфейс ScheduleTaskHandle и аннотирован с помощью @ScheduleTask. Данное задание имеет тип SheduleType.Multipliable то есть можно создать множество заданий этого типа с разными параметрами. Параметр задания это класс имплементирующий интерфес ScheduleTaskParameters следующего вида:

```
public class NotificationTaskConfig implements ScheduleTaskParameters {

    private static final long serialVersionUID = 2618754657538579112L;

    /**
     * Описание способа получения доменных объектов
     */
    @Attribute
    private FindObjectsConfig findDomainObjects;

    /**
     * Описание способа получения персон
     */
    @Attribute
    private FindObjectsConfig findPersons;

    /**
     * Тип сообщения
```

```
     */
    @Attribute
    private String notificationType;

    /**
     * Приоритет сообщения
     */
    @Attribute
    private NotificationPriority notificationPriority;

    /**
     * Флаг типа формирования сообщений, относительно доменного объекта или
относительно персоны
     */
    @Attribute
    private NotificationTaskMode taskMode;

    public String getNotificationType() {
        return notificationType;
    }

    public void setNotificationType(String notificationType) {
        this.notificationType = notificationType;
    }

    public NotificationPriority getNotificationPriority() {
        return notificationPriority;
    }

    public void setNotificationPriority(NotificationPriority
notificationPriority) {
        this.notificationPriority = notificationPriority;
    }

    public FindObjectsConfig getFindDomainObjects() {
        return findDomainObjects;
    }

    public void setFindDomainObjects(FindObjectsConfig findDomainObjects) {
        this.findDomainObjects = findDomainObjects;
    }

    public FindObjectsConfig getFindPersons() {
        return findPersons;
    }

    public void setFindPersons(FindObjectsConfig findPersons) {
        this.findPersons = findPersons;
    }

    public NotificationTaskMode getTaskMode() {
        return taskMode;
    }

    public void setTaskMode(NotificationTaskMode taskMode) {
        this.taskMode = taskMode;
    }
}
```

Периодическое задание работает следующим образом. В случае если атрибут taskMode равен BY_DOMAIN_OBJECT - при запуске вызывается метод getFindDomainObjects и с помощью сервиса FindObjectsService получается список доменных объектов. Далее

вызывается метод getFindPersons и с помощью сервиса FindObjectsService получаются персоны для каждого полученного на первом шаге доменного объекта. Выполняется формрование уведомлений каждому пользователю с помощью метода NotificationService.sendOnTransactionSuccess. В метод передается тип сообщения полученное методом getNotificationType и приоритет сообщения, полученное методом getNotificationPriority. В контекст добавляется доменный объект под именем document. В случае если атрибут taskMode равен BY_PERSON то происходит сначала поиск пользователей с помощью конфигурации поиска описанного в атрибуте getFindPersons, а затем получается список доменных объектов для каждого пользователя с помощью метода getFindDomainObjects. Далее отправка происходит как и в первом варианте. В контекст добавляется доменный объект под именем document. Для облегчения создания подобных задач можно воспользоваться импортом из CSV файлов, параметры в этом случае передаются в виде XML.

3.8.3.10 Задача подсистемы workflow для формирования уведомлений

Для отправки уведомлений из процессов необходимо создать класс сервиса workflow, который выполняет отправку сообщений с помощью метода NotificationService.transactionSend. Класс должен настраиваться следующими полями:

- addressee – строка в виде xml, по которой формируется список адресатов. Для облегчения формирования данной строки необходимо создать метод session.getNotificationAddressee(…)
- context – строковой идентификатор основного доменного объекта процесса. Данные доменный объект будет передан в качестве контекста сообщения под именем document
- notificationType – строка, тип сообщения
- notificationPriority – строка, одно из значений HIGH, NORMAL, LOW.

При получение управления сервис берет из полей задачи необходимые для отправки данные, и формирует уведомления используя метод NotificationService.transactionSend.

Для облегчения формирования списка адресатов в классе сессии необходимо создать пять методов: четыре метода по созданию каждого типа адресата (createPersonAddressee, createGroupAddreseee, createDynamicGroupAddressee, createContextRoleAddressee), и один метод принимающий на вход массив адресатов, а на выходе xml строка, которая далее передается в поле сервиса. Сервис выполняет обратную задачу десериализации XML в массив адресатов, и далее отправка этим адресатам сообщений.

3.8.3.11 Доработка сервисов динамических групп и контекстных ролей

В рамках задач рассылки уведомлений требуется расширить функциональность сервиса формирования динамических групп DynamicGroupService и сервиса формирования контекстных ролей PermissionService. В данных сервисах требуется добавить метод получения состава персон, исходя из имени группы или роли и идентификатора контекстного доменного объекта. Сигнатура методов данных сервисов:

```
DynamicGroupService{
…
List<Id> getPersons(Id contextId, groupName);
}
```

```
PermissionService {
…
List<Id> getPersons(Id contextId, roleName);
}
```

В обоих методах возвращается список идентификаторов персон, соответствующих динамической группе или контекстной роли.

3.8.3.12 Сервис поиска доменных объектов с помощью DOEL, запроса или класса

При разработки подсистемы отправки уведомлений периодически возникает потребность получить список идентификаторов доменных объектов с помощью одного из способов: с помощью DOEL выражения, с помощью запроса, с помощью класса. Данная потребность возникает достаточно часто, и по этому необходимо создать сервис, выполняющий эти действия. Сервис представляет собой spring сервис со следующим интерфейсом:

```
public interface DomainObjectFinderService {

    /**
     * Метод получения списка идентификаторов доменных объектов по одному из
параметров: запроса, класса или DOEL
     * выражения
     * @param getObjectsConfig
     *            Параметр в котором содержится или имя класса или запрос или
doel выражение
     * @param contextDomainObjectId
     *            идентификатор доменного объекта, относительно которого
производятся вычисления. Идентификатори
     *            передается в запрос в параметре номер 1 ({1}), а в класс
как параметр метода получения списка
     *            объектов.
     * @return
     */
    List<Id> findObjects(FindObjectsConfig getObjectsConfig, Id
contextDomainObjectId);
}
```

В качестве параметра методу findObjects передается экземпляр класса FindObjectsConfig, а в нем есть поле типа интерфейс FindObjectsType, которое может быть FindObjectsDoelConfig, FindObjectsQueryConfig, FindObjectsClassConfig. Данные классы могут мапится на XML благодаря классу FindObjectsConfig, в котором с помощью аннатаций описаны теги XML на которые производится маппинг XML файла.

```
public class FindObjectsConfig implements Dto{

    @ElementUnion({
        @Element(name="class-name", type=FindObjectsClassConfig.class),
        @Element(name="doel", type=FindObjectsDoelConfig.class),
        @Element(name="query", type=FindObjectsQueryConfig.class)
     })
    private FindObjectsType findObjectType;

    public FindObjectsType getFindObjectType() {
        return findObjectType;
    }

    public void setFindObjectType(FindObjectsType findObjectType) {
        this.findObjectType = findObjectType;
```

```
    }
}
```

При работе сервиса, если на вход подается FindObjectsConfig содержащий FindObjectsClassConfig то создается экземпляр класса, имя которого указано внутри FindObjectsClassConfig. Этот класс жолжен имплементить интерфейс `DomainObjectFinder`:

```
/**
 * Интерфейс получения списка идентификаторов доменных объектов относительно
контекстного доменного объекта
 * @author larin
 *
 */
public interface DomainObjectFinder {
    /**
     * Метод получения списка идентификатолв доменных объектов относительно
контекстного доменного объекта
     * @param contextDomainObjectId
     * @return
     */
    List<Id> findObjects(Id contextDomainObjectId);
}
```

В случае если передается FindObjectsQueryConfig то выполняется содержащийся в нем запрос а идентификатор доменного объекта передается в качестве параметра с номером 1.

В случае если передается FindObjectsDoelConfig то выполняется DOEL выражение, содержащееся внутри класса, относительно переданного контекстного доменного объекта.

3.9 Интеграция с JasperReports

3.9.1 Основные положения

Библиотека JasperReports интегрируется в ядро системы с целью выполнения следующих задач:

1. Формирование онлайн отчетов.
2. Формирование офлайн отчетов.
3. Формирование периодических отчетов.

3.9.2 Способ интеграции

Библиотеки JaserReports включают в состав итогового ear системы. Создается EJB ReportService для управления подсистемой отчетов. Отчеты могут формироваться в 3 режимах:

1. Онлайн – формирование отчета начинается непосредственно после вызова соответствующей функции EJB. Отчет возвращается как результат выполнения данной функции. Возвращаемое значение функции класс ReportResult, одним из полей которого сам отчет как массив байт.
2. Офлайн – формирование отчета асинхронно. Запуск формирования производится соответствующей функцией EJB. Для запуска используется технология Asynchronous EJB call. После окончания формирования отчета формируется уведомление в системе уведомлений. В процессе формирования отчета должна

быть возможность сформировать на вызывающей стороне progressBar о процессе формирования.

3. Периодические отчеты. В системе создается специальный тип доменных объектов – scheduled_task описывающий запуск задания по расписанию. Каждое из таких заданий имеет параметр тип задания, и соответствующее этому типу заданий набор параметров. Одним из таких заданий является задача формирования отчета. Как параметры к такому заданию является имя отчета и набор параметров отчета. Параметры могут быть как жестко заданные так и вычисляемые (пример вычисляемого параметра отчета – период за который формируется отчет, подразделение пользователя, сформировавшего задание на отчет) Периодические отчеты формируют сами пользователи, то есть в интерфейсе должен быть блок, позволяющий сформировать параметры периодических отчетов. Периодические отчеты формируются системой и отправляют уведомления о формирование по электронной почте. Если пользователь в данный момент подключен в системе на экране отображается уведомление о сформированном отчете. Описание подсистемы периодических заданий выходит за рамки данного документа.

Идентификацией отчета в системе является его имя. Оно уникально. Информация об отчете хранится в специальных доменных объектах report_template. В report_template хранится информация об имени отчета и привязаны вложения с шаблонами отчета и xml файлом описания отчета. Шаблонов может быть много (в случае с подотчетами, но только одно из вложений может быть основным шаблоном отчета). В xml файле описания отчета хранится метаинформация о параметрах отчета, позволяющая отобразить на интерфейсе форму ввода параметров отчета. В качестве формы ввода параметров отчета указывается форма из сконфигурированных форм основной конфигурации ядра, плюс имеется конфигурация для начальной инициализации полей форм. Так же в файле описания отчета хранится имя основного шаблона отчета, имя класса источника данных, доступные форматы, в которых формируется отчет, флаг времени хранения отчета, флаг возможности изменить время хранения отчета по умолчанию.

3.9.3 JDBC драйвер

Доступ к данным осуществляется с помощью JDBC драйвера, специально разработанного для платформы. Данный драйвер обращается к данным через ядро системы с учетом прав доступа. Драйвер имеет 2 режима работы с помощью локального интерфейса EJB и с помощью удаленного интерфейса EJB. Выбор способа работы драйвера определяется разработчиком использующего данный драйвер и указывается в строке подключения драйвера. В случае с работой через Remote интерфейсы так же в строке указывается имя сервера, логин и пароль. Драйвер разрабатывается таким образом, чтобы через него можно было бы работать с редактором шаблонов отчета iReport. Так же драйвер может использоваться для работы внешних систем построения отчетов, например CrystalReport Server или JasperReport Server. Драйвер должен работать с сервисом коллекций с учетом страничной подкачки данных, для исключения ошибки типа OutOfMemoryException.

3.9.4 Описание сущностей

Тип доменного объекта report_template – Шаблон отчета.

Поле	Описание
name	Строка. Имя отчета. Уникально в пределах системы. Используется для однозначной

	идентификации шаблона отчета в системе.
description	Строка. Описание отчета. Используется интерфейсом для формирования элементов управления, запускающих формирование отчета

К экземпляру данного типа прикрепляются вложения: шаблоны отчета, шаблоны подотчетов, xml файл с метаинформацией, класс скриптлета (не обязательно), класс источника данных (не обязательно).

Тип доменного объекта report_result – Сформированный отчет

Поле	Описание
name	Имя отчета. Формируется исходя из имени шаблона отчета и времени формирования отчета
template_id	Reference на report_template. Идентификатор шаблона отчета.
owner	Reference на Person. Идентификатор персоны сформировавшей отчет или сформировавший задание на отчет.
keep_to	Дата окончания хранения отчета. После этой даты отчет может быть удален специальным санитарным джобом.

К экземпляру данного объекта прикрепляется сформированный отчет и файл xml с параметрами отчета.

Интерфейс EJB ReportService.

Метод	Параметр	Описание
generate	Имя шаблона отчета, формат отчета, параметры отчета	Производится онлайн формирование отчета. Результатом работы функции является экземпляр класса ReportResult.
generateAsync	Имя шаблона отчета, формат отчета, параметры отчета, права на сформированный отчет.	Офлайн (асинхронное) формирование отчета. Функция возвращает управление немедленно. Затем в асинхронном режиме формирует отчет и отправляет результат в подсистему уведомлений (подсистема уведомлений выходит за рамки данного документа. В простейшем случае подсистема уведомлений отправляет email с отчетом пользователю запустившем формирование задания) Асинхронный вызов

		должен поддерживать механизм отслеживания статуса формирования отчета, для формирования progressBar на клиентских местах.

Интерфейс ReportServiceAdmin

Метод	Параметр	Описание
deploy	Передается структура описывающая шаблон отчета. В ту структуру входят имя отчета, все шаблоны отчетов в формате jrxml и xml файл с метоинформацией отчета, файл класса скриптлета, файл класса источника данных.	При вызове метода происходит компиляция шаблонов и классов скриптлета и источника данных, и формирование или обновление доменного объекта report_template. В случае неудачи компиляции происходит формирование Exception.
undeploy	Имя отчета	Удаляется информация о шаблоне отчета. Удаляется соответствующий report_template и прикрепленные к нему вложения.

XML файл метаинформации.

XML элемент	**Тип**	**Описание**
Template	тэг	Описание шаблона отчета
name	атрибут	Имя шаблона
description	атрибут	Описание отчета
mainTemplate	атрибут	Имя файла основного шаблона отчета без расширения
dataSourceClass	Атрибут	Имя класса источника данных (не обязательный)
form	атрибут	Имя конфигурации формы
Parametrs	Тэг	Паракметры отчета
Parameter	Тэг	В данном документе указан условно. Может быть StringParameter, NumberParameter, DateParameter, IntervalParameter, ListParameter и т.д. Содержит одинаковые для всех параметров атрибуты так и специфичные для параметра определенного типа (например источник записей для списка)
name	Атрибут	Имя параметра. Совпадает с именем параметра в шаблоне отчета именем

		источника данных для виджета на форме ввода параметров отчета.
settings	Тэг	Настройки конкретного элемента, здесь указан условно. Данный тэг зависит от виджета на форме ввода параметров отчета. Должен формировать правила заполнения виджета по умолчанию.

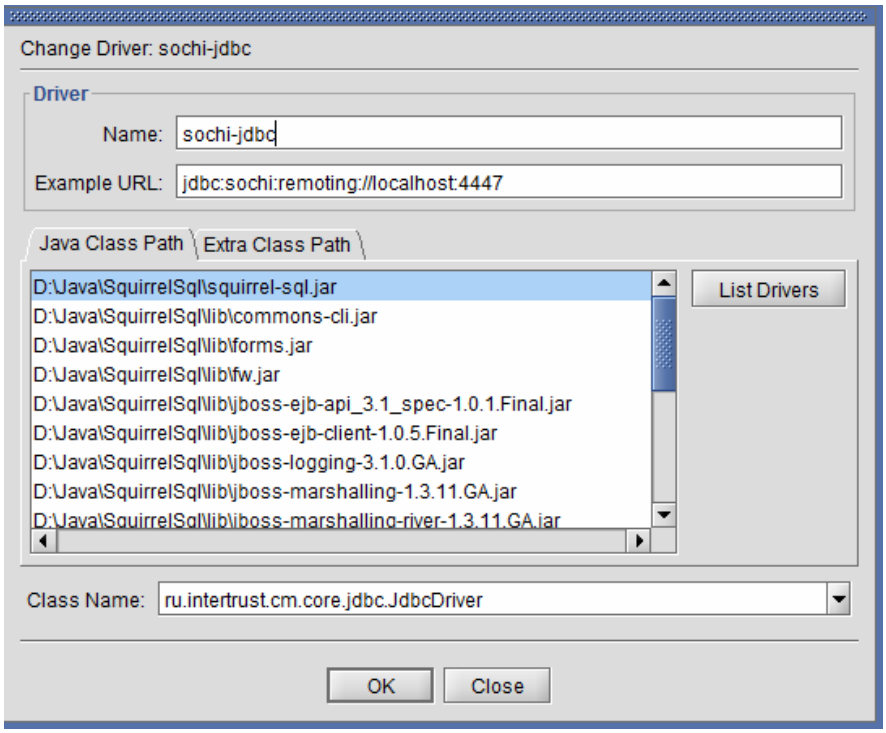
Формат файла параметров отчета

XML элемент	Тип	Описание
ReportParam	тэг	Тэг верхнего уровня
Params	тэг	Описание параметров отчета
Param	Тэг	Описание параметра отчета
name	Атрибут	Имя параметра
value	Атрибут	Значение параметра

3.9.5 Настройка подключения программ используя JDBC драйвер

3.9.5.1 SQuirreL SQL Client

1. Скопировать следующие библиотеки в директорию SquirrelSql\lib:
    a. service-api-1.0-SNAPSHOT.jar
    b. model-1.0-SNAPSHOT.jar
    c. xnio-nio-3.0.7.GA.jar
    d. jboss-remoting-3.2.17.GA.jar
    e. jboss-remote-naming-1.0.2.Final.jar
    f. xnio-api-3.0.7.GA.jar
    g. jboss-ejb-client-1.0.5.Final.jar
    h. jboss-ejb-api_3.1_spec-1.0.1.Final.jar
    i. jboss-transaction-api_1.1_spec-1.0.0.Final.jar
    j. jboss-logging-3.1.0.GA.jar
    k. jboss-marshalling-1.3.11.GA.jar
    l. jboss-marshalling-river-1.3.11.GA.jar
    m. log4j-1.2.16.jar
    n. jdbc-1.0-SNAPSHOT.jar
2. Удалить имеющийся в директории SquirrelSql\lib файл log4j.jar
3. Запустить приложение
4. В окне доступных драйверов Drivers создать драйвер sochi-jdbc как показано ка картинке

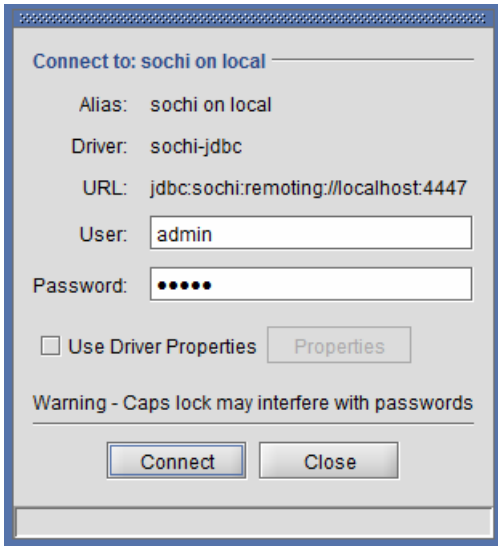


В поле Name ввести произвольное имя

В поле Example URL ввести jdbc:sochi:remoting://localhost:4447. Вместо localhost должно быть корректное имя сервера, вместо 4447 нужно указать корректный порт для remoting соединений.

В поле Class Name ввести имя класса драйвера ru.intertrust.cm.core.jdbc.JdbcDriver

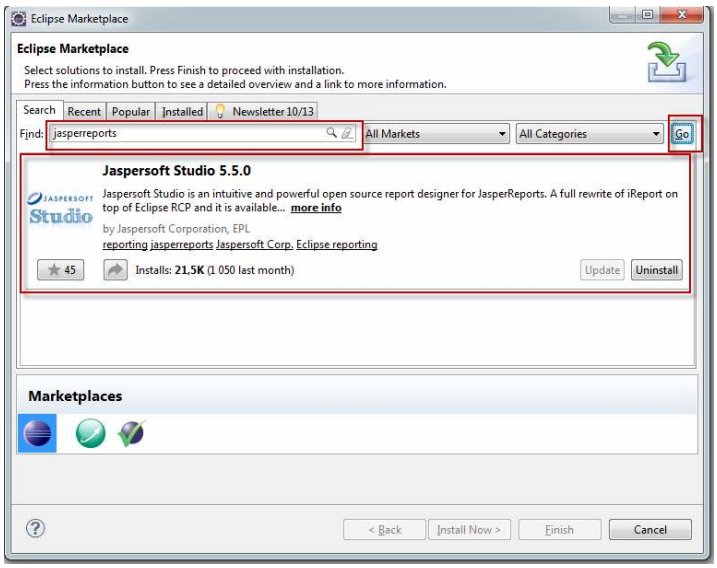
5. В окне подключений Aliases необходимо создать новое подключение как показано на рисунке



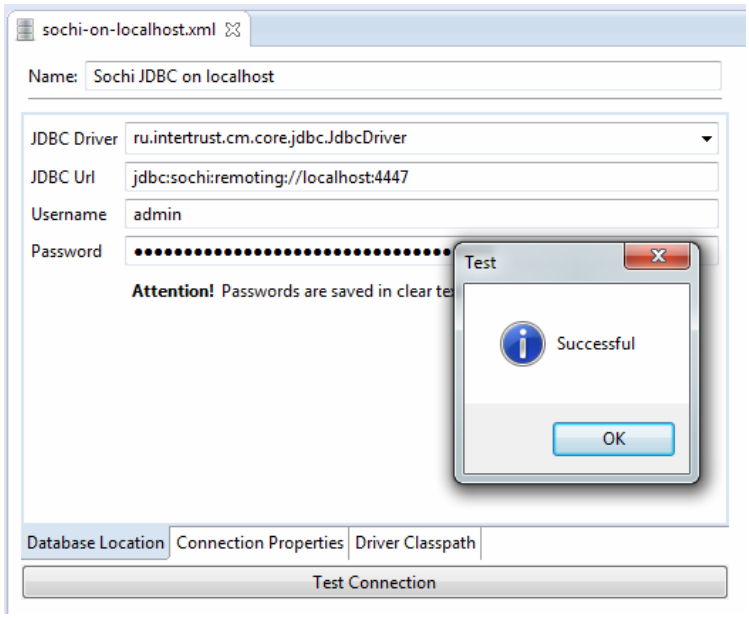
6. Нажать кнопку Connect, произойдет подключение и отобразится поле для ввода запроса. В данном поле можно писать запрос на внутреннем DSQL языке и получать результат.

3.9.5.2 JasperReport Studio

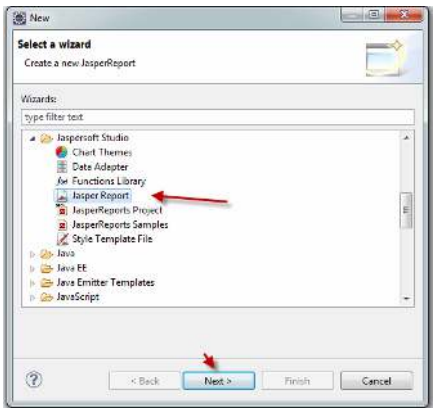
1. Установить плагин JasperReport Studio используя Eclipse Marketplace



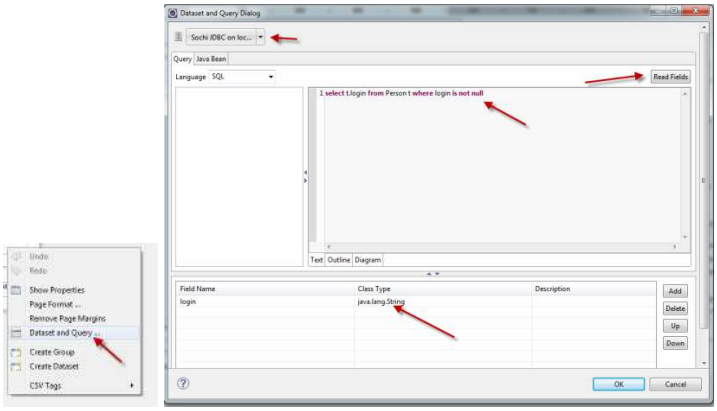
2. Создать workspace директорию отдельно от директорий проекта. Важно wokspace должен быть отдельным, и не являтся workspace проекта sochi.
3. Импортировать maven проект CM5\Client\reports
4. Отредактировать файл sochi-on-localhost.xml, указать корректные логин пароль и имя сервера.
5. Проверить соединение нажав кнопку test на форме проверки соединения. Тест должен пройти.



6. Создать новый шаблон отчета. Для каждого шаблона отчета нужна своя отдельная директория.



7. В шаблоне отчета из контекстного меню на свободном месте отчета вызываем конфигуратор источника данных. В нем указываем соединение и пишем запрос. Далее нажимаем кнопку Read Fields. Должны сформироваться поля.



8. Далее форматируем отчет и наполняем его содержимым.

3.10 Подсистема импорта данных из CSV-файла

Для импорта данных реализована подсистема загрузки данных из CSV файла.

Подсистема работает в 2-х режимах:

1. Загрузка данных с помощью remote вызова метода сервиса ImportDataService.
2. Автоматическая загрузка данных при старте сервера.

3.10.1 Загрузка данных с помощью remote вызова метода сервиса ImportDataService.

Для загрузки данных необходимо вызвать метод importData сервиса ImportDataService и передать методу как параметр массив байт зачитанного CSV файла. Интерфейс сервиса загрузки данных:

```
package ru.intertrust.cm.core.business.api;

/**
 * Сервис загрузки данных
 * @author larin
 *
 */
public interface ImportDataService {
    public static final String TYPE_NAME = "TYPE_NAME";
    public static final String KEYS = "KEYS";
```

```
    /**
     * Удаленный интерфейс
     * @author larin
     *
     */
    public interface Remote extends ImportDataService{
    }
    /**
     * Метод загрузки данных из файла.
     * @param loadFileAsByteArray зачитанный массив данных из файла
     */
    void importData(byte[] importFileAsByteArray);
}
```

3.10.2 Автоматическая загрузка данных при старте сервера

При старте сервера производится поиск всех файлов ru/intertrust/cm/core/importdata/import.xml. Файлы могут быть расположены в разных архивах. В файлах import.xml содержатся имена файлов csv для загрузки и их зависимость от других файлов csv для обеспечения корректной очередности загрузки файлов. Зависимость учитывается иерархически. Так же производится проверка на зацикленность зависимостей, в этом случае формируется исключение. Файлы csv должны так же лежать в директории ru/intertrust/cm/core/importdata. Формат файла import.xml:

```
<?xml version="1.1" encoding="UTF-8" standalone="yes"?>
<ImportData>
    <File name="import-organization.csv">
        <Depend>status.csv</Depend>
    </File>
    <File name="import-department.csv">
        <Depend>status.csv</Depend>
        <Depend>import-organization.csv</Depend>
    </File>
    <File name="import-employee.csv">
        <Depend>status.csv</Depend>
        <Depend>import-department.csv</Depend>
    </File>
    <File name="set-organization-boss.csv">
        <Depend>status.csv</Depend>
        <Depend>import-employee.csv</Depend>
    </File>
    <File name="set-department-boss.csv">
        <Depend>status.csv</Depend>
        <Depend>import-employee.csv</Depend>
    </File>
</ImportData>
```

3.10.3 Формат файла импорта данных

По умолчанию файл CSV должен быть сохранен в кодировке ANSI-1251. Сделано это для совместимости с редактором Excel. В случае если кодировка файла отлична от ANSI-1251 кодировка должна быть передана с помощью параметра encoding метода importData.

Две первые строки файла отведены под метаинформацию. В первой строке задается имя типа для импорта, ключевые поля через запятую и символ, который будет интерпретироватся как символ строки с нулевой длиной в виде:

```
TYPE_NAME=Department;KEYS=Name,Number;EMPTY_STRING_SYMBOL=$
```

Значение EMPTY_STRING_SYMBOL по умолчанию равен символу подчеркивания “_”.

Во второй строке хранятся имена полей для импорта:

```
Name;Organization;ParentDepartment
```

Начиная с третьей строки идут непосредственно данные для импорта

```
Подразделение 1;"name=""Организация 1""";
Подразделение 2;"name=""Организация 1""";
Подразделение 3;"name=""Организация 1""";
```

В полях типа Reference необходимо прописывать выражение, для получение значения ссылки на другой доменный объект. Выражения могут быть трех видов:

1. field_name=”field_value”. Данное выражение означает, что надо получит конфигурацию поля типа Reference из конфигурации доменного объекта. Далее у поля типа Reference получается тип доменного объекта на который ссылается данное поле, изо всех доменных объектов полученного типа делается запрос с условием field_name=”field_value” и из полученного множества берется первый результат.
2. type_name.field_name=”field_value”. Данное выражение означает, что изо всех доменных объектов типа type_name делается запрос с условием field_name=”field_value” и из полученного множества берется первый результат. Данная конструкция используется когда поле Reference может ссылаться на несколько типов доменных объектов.
3. Запрос на внутреннем sql языке. Запрос должен возвращать поле типа Reference. Например: select t.id from department t where t.name='Подразделение 2'.

3.11 Подсистема периодических заданий

3.11.1 Основные положения

Подсистема периодических заданий должна отвечать следующим требованиям:

1. Администратор должен иметь возможность менять расписание заданий, включать и выключать их, менять конфигурацию заданий. Конфигурация заданий это класс, способный сериализоваться в xml для последующего хранения экземпляра этого класса в доменном объекте.
2. Код периодического задания это спринг бин аннотированный специальной аннотацией и имплиментящий определенный интерфейс. Ядро системы при старте найдет все такие бины и зарегистрирует их в подсистеме периодических заданий
3. Периодические задания должны выполняться с учетом приоритета. То есть если одновременно должны запуститься несколько заданий с разным приоритетом, то сначала запускаются задания с более высоким приоритетом, потом с более низким.

4. Периодические задания должны использовать пул процессов, чтобы была возможность выполнять несколько заданий одновременно.
5. Периодические задания бывают двух видов.
    a. Первый вид - это периодическое задание способного существовать в системе в единственном экземпляре. Экземпляры периодических заданий этого вида всегда присутствуют в списке заданий, их нельзя удалить, можно сделать неактивными, можно поменять конфигурацию
    b. Второй вид – это периодические задания способные существовать в системе в нескольких экземплярах. Такие задания после установки в систему не имеют экземпляров, экземпляры таких заданий может создать администратор, установить периодичность, активность, конфигурацию, а так же удалить из системы.

3.11.2 Реализация

3.11.2.1 Инициализация

При старте системы сервисом ScheduleService производится сканирование классов на предмет наличия класса аннотированного анотацией ScheduleTask. Составляется список всех таких классов. В случае если задача относится к классу «Только один экзампляр», то в хранилище создается доменный объект schedule для данного класса и заполняются поля значениями по умолчанию, которые берутся из аннотации ScheduleTask.

Администратор, с помощью вызова методов сервиса ScheduleService. createScheduleTask может создать экземпляры задач класса «Много экземпляров». При создание задач такого класса значения полей доменного объекта schedule заполняются по умолчанию данными, полученными у аннотации ScheduleTask.

3.11.2.2 Работа

Разработан Statless EJB - SchedulerBean, с методом аннотированным @Schedule и настроенным на выполнение один раз в минуту. Данный EJB является инициатором выполнения всех периодических заданий. Разработан Statless EJB – ScheduleProcessor с асинхронным методом startAsync. Данный EJB асинхронно выполняет задания в пуле процессов. Создан тип доменного объекта schedule который хранит расписание, активность, параметры, приоритет и статус периодического задания. Раз в минуту вызывается методы SchedulerBean в котором производятся следующие действия:

1. Просматривается список запущенных задач и в случае выполнения их больше положенного времени задача прерывается и соответствующий статус сохраняется в доменном объекте задачи. Для прерывания используется объект класса Future, возвращаемый методом асинхронного запуска задач. Данные действия выполняются в методе checkTimeoutBackgroundProcessing и сдвинуты относительного старта задачь на 10 секунд.
2. Анализируются записи объектов schedule и производится вычисление тех задач, которые надо запустить в данный момент времени. Задачи сортируются по приоритету.
3. Из тех списков задач, которые надо запустить в текущий момент времени удаляются те задачи, которые еще выполняются.
4. У оставшихся задач устанавливается статус READY.

5. Зачитываются задачи в статусе READY (на текущий момент все, в будущем возможно надо будет зачитывать не все, а только первые N с наивысшим приоритетом, для того чтобы не забивать пул потоков), им устанавливается статус WAIT и отдаются на выполнение путем вызова асинхронного метода ScheduleProcessor.
6. ScheduleProcessor выполняет задачу и по окончании устанавливает статус SLEEP

Все задачи выполняются в одном пуле потоков. Размер пула потоков задается в конфигурации сервера приложений.

Задачи выполняются от имени системы.

3.11.2.3 Описание интерфейсов и доменных объектов

Интерфейс администрирования сервиса периодических заданий

```
package ru.intertrust.cm.core.business.api;

import java.util.List;

import ru.intertrust.cm.core.business.api.dto.DomainObject;
import ru.intertrust.cm.core.business.api.dto.Id;
import ru.intertrust.cm.core.business.api.schedule.Schedule;
import ru.intertrust.cm.core.business.api.schedule.ScheduleTaskParameters;

/**
 * Сервис периодических заданий
 * @author larin
 *
 */
public interface ScheduleService {

    public static final String SCHEDULE_NAME = "name";
    public static final String SCHEDULE_TASK_CLASS = "task_class";
    public static final String SCHEDULE_TASK_TYPE = "task_type";
    public static final String SCHEDULE_YEAR = "year";
    public static final String SCHEDULE_MONTH = "month";
    public static final String SCHEDULE_DAY_OF_MONTH = "day_of_month";
    public static final String SCHEDULE_DAY_OF_WEEK = "day_of_week";
    public static final String SCHEDULE_HOUR = "hour";
    public static final String SCHEDULE_MINUTE = "minute";
    public static final String SCHEDULE_TIMEOUT = "timeout";
    public static final String SCHEDULE_PRIORITY = "priority";
    public static final String SCHEDULE_PARAMETERS = "parameters";
    public static final String SCHEDULE_LAST_RESULT = "last_result";
    public static final String SCHEDULE_LAST_RESULT_DESCRIPTION =
"last_result_description";
    public static final String SCHEDULE_ACTIVE = "active";
    public static final String SCHEDULE_LAST_REDY = "last_redy";
    public static final String SCHEDULE_LAST_WAIT = "last_wait";
    public static final String SCHEDULE_LAST_RUN = "last_run";
    public static final String SCHEDULE_LAST_END = "last_end";

    public static final String SCHEDULE_STATUS_SLEEP = "Sleep";
    public static final String SCHEDULE_STATUS_READY = "Ready";
    public static final String SCHEDULE_STATUS_WAIT = "Wait";
    public static final String SCHEDULE_STATUS_RUN = "Run";

    /**
     * Удаленный интерфейс
     * @author larin
     *
```

```
     */
    public interface Remote extends ScheduleService{
    }

    /**
     * Получение всех задач, которые обрабатываются сервисом периодических
заданий.
     * @return
     */
    List<DomainObject> getTaskList();

    /**
     * Получение классов задач которые могут существовать во множественном
числе экземпляров. Используется GII дла
     * отрисовки диалога создания периодического задания
     * @return
     */
    List<String> getTaskClasses();

    /**
     * Получение расписания задачи
     * @return
     */
    Schedule getTaskSchedule(Id taskId);

    /**
     * Установка расписания задачи
     * @param taskId
     * @param schedule
     */
    void setTaskSchedule(Id taskId, Schedule schedule);

    /**
     * Получение параметров задачи
     * @return
     */
    ScheduleTaskParameters getTaskParams(Id taskId);

    /**
     * Установка параметров задачи
     * @param taskId
     * @param schedule
     */
    void setTaskParams(Id taskId, ScheduleTaskParameters parameters);

    /**
     * Активировать задание
     * @param taskId
     */
    void enableTask(Id taskId);

    /**
     * Деактивировать задание
     * @param taskId
     */
    void disableTask(Id taskId);

    /**
     * Запустить задание
     * @param taskId
     */
    void run(Id taskId);
```

```
    /**
     * Установка приоритета. Значения могут быть от 0 до 4
     * @param priority
     */
    void setPriority(Id taskId, int priority);

    /**
     * Установка таймаута в минутах
     * @param timeout
     */
    void setTimeout(Id taskId, int timeout);

    /**
     * Создание периодического задания класса multible
     * @param string
     * @return
     */
    DomainObject createScheduleTask(String className, String name);

}
```

Интерфейс ScheduleTaskHandle, данный интерфейс должен имплиментить класс выполняющий задачу

```
package ru.intertrust.cm.core.business.api.schedule;

/**
 * Интерфейс, который должны имплиментить все классы периодических заданий
 * @author larin
 *
 */
public interface ScheduleTaskHandle {
    /**
     * Запуск выполнения периодического задания
     * @param parameters
     * @return возвращает результат работы периодического задания в виде
строки. Строка будет хранится в доменном обете задания
     */
    String execute(ScheduleTaskParameters parameters);
}
```

Интерфейс ScheduleTaskDefaultParameters, данный интерфейс должен иплиментить класс описывающий настройку задачи по умолчанию

```
package ru.intertrust.cm.core.business.api.schedule;

/**
 * Интерфейс конфигурации периодического задания по умолчанию.
 * Наследники используются в анотации ScheduleTask в качестве параметра
 * @author larin
 *
 */
public interface ScheduleTaskDefaultParameters {
    /**
     * Получение параметров по умолчанию
     * @return
     */
    ScheduleTaskParameters getDefaultParameters();
}
```

Доменный объект schedule

Поле	Тип	Коментарии
name	String	Имя задания
task_class	String	Имя java класса задачи. По данному имени будет производится поиск в контексте спринг бинов для поиска нужного экземпляра.
task_type	Long	Тип задачи. 0 – может существовать в единствееном экземпляре. 1 – может существовать во множестве экземпляров.
day_of_week	String	День недели старта. * - всегда, число – когда равен числу, */число когда число кратно числу в дроби (как в crond)
day_of_month	String	День месяца
hour	String	Час старта
minute	String	Минута старта
month	String	Месяц старта
year	String	Год
timeout	Long	Допустимое время работы (в минутах)
Priority	Long	Приоритет задачи. Может принимать значения от 0 до 4. Наивысший приоритет – 0, наинизший приоритет – 4.
parameters	String	Сеарелизованные в строку параметры задачи
Status	Reference	Системный атрибут статуса. Может иметь значения: SLEEP – не запущен; READY-готов к выполнению; WAIT- ожидает выполнения в пуле потоков; RUN-В работе.
last_result	long	0- Никогда не запускался 1- отработал без ошибки,

		2- прерван в результате ошибки внутри задачи, 3- прерван в связи с превышением допустимого времени работы
last_result_description	String	Строковое описание последнего результата работы. Здесь сохраняется результат метода execute интерфейса ScheduleTaskHandle или ошибка в случае завершения работы с ошибкой
active	Long	0- задача активна и запустится по расписанию 1- задача не активна
last_redy	Date	Время крайней установки флага готов к выполнению
last_wait	Date	Время крайнего добавления в очередь на выполнение
last_run	Date	Время крайнего запуска на исполнение
last_end	Date	Время окончания исполнения при крайнем запуске

Пример периодического задания.

TestSingleSchedule – Single задание выполняющееся по умолчанию один раз в минуту

```
package ru.intertrust.cm.test.schedule;

import ru.intertrust.cm.core.business.api.schedule.ScheduleTask;
import ru.intertrust.cm.core.business.api.schedule.ScheduleTaskHandle;
import ru.intertrust.cm.core.business.api.schedule.ScheduleTaskParameters;
import ru.intertrust.cm.core.model.ScheduleException;

@ScheduleTask(name = "TestSingleSchedule", minute = "*/1")
public class TestSingleSchedule implements ScheduleTaskHandle {

    @Override
    public String execute(ScheduleTaskParameters parameters) {
        try {
            System.out.println("Run TestSingleSchedule");
            Thread.currentThread().sleep(10000);
            return "COMPLETE";
        } catch (Exception ex) {
            throw new ScheduleException("Error exec TestSingleSchedule", ex);
        }
    }
```

```
}
```

Множественное задание выполняющееся один раз в минуту

package ru.intertrust.cm.test.schedule;

```
import ru.intertrust.cm.core.business.api.schedule.ScheduleTask;
import ru.intertrust.cm.core.business.api.schedule.ScheduleTaskHandle;
import ru.intertrust.cm.core.business.api.schedule.ScheduleTaskParameters;
import ru.intertrust.cm.core.business.api.schedule.SheduleType;
import ru.intertrust.cm.core.model.ScheduleException;

@ScheduleTask(name = "TestScheduleMultiple", minute = "*/1", configClass =
TestSheduleDefaultParameter.class,
        type = SheduleType.Multipliable)
public class TestScheduleMultiple implements ScheduleTaskHandle {

    @Override
    public String execute(ScheduleTaskParameters parameters) {
        try {
            TestScheduleParameters testScheduleParameters =
(TestScheduleParameters) parameters;

            //Тестируем обработчик ошибки в поле Result может быть как строка
так и число. Если строка то упадем с ошибкой, что должны увидеть в
результатах выполнения задачи
            long value = Long.parseLong(testScheduleParameters.getResult());

            System.out.println("Run TestScheduleMultiple");
            Thread.currentThread().sleep(value);
            return testScheduleParameters.getResult();
        } catch (Exception ex) {
            throw new ScheduleException("Error exec TestScheduleMultiple",
ex);
        }
    }

}
```

3.12 Подсистема AuditLog

Подсистема AuditLog предназначена для хранения всех изменений доменных объектов с целью определения пользователя, который внес изменения в доменный объект, состав и дату этих изменений, а так же возможности восстановления доменного объекта на определенный момент времени.

3.12.1 Способ реализации

Для хранения всех версий доменных объектов в базе данных создается структура хранения, аналогичная структуре для хранения самих доменных объектов с добавлением дополнительных полей.

На каждый тип доменных объектов создается набор таблиц с суффиксом «_log» для базового типа и для всех дочерних типов. Состав атрибутов аналогичен основным таблицам, за исключением:

1. Добавляется поля для хранения идентификатора доменного объекта domain_object_id и domain_object_type.
2. Не строится внешний ключ на основные таблицы хранения доменных объектов. Это необходимо для того, чтобы была возможность сохранить аудит после удаления основного доменного объекта.
3. Добавляются поля для хранения дополнительной информации о версии:
    a. component – идентификатор компоненты из accessTocken
    b. ip_address – IP адрес удаленной машины выполнившей изменения
    c. info – дополнительная произвольная информация (зарезервировано)
    d. operation – выполненное действие (1-создание, 2-изменение, 3-удаление)

Поле id таблицы хранения версий содержит уникальный идентификатор версии, который используется для однозначной идентификации версии в системе.

В конфигурацию добавляется раздел верхнего уровня <global-settings>, тэг может быть в xml только один раз. Внутри добавляется тэг <audit-log default-enable=”true|false”/> конфигурирующий поведение аудит лога по умолчанию.

Для детальной конфигурации AuditLog для конкретного типа в конфигурации доменных объектов добавляется атрибут audit-log в теге domain-object-type. Атрибут не обязательный. Если атрибут не указан то берется значение из глобальной конфигурации из тэга audit-log атрибута enable, если в глобальной конфигурации отсутствует тэг audit-log то значение принимается false.

Добавление записей в AuditLog производится внутри domainObjectDao бине и не доступно извне.

Для работы с аудитом создаются спринг бин AuditLogServiceDao и EJB AuditLogService:

```
package ru.intertrust.cm.core.dao.api;

import java.util.List;

import ru.intertrust.cm.core.business.api.dto.DomainObjectVersion;
import ru.intertrust.cm.core.business.api.dto.Id;

/**
 * Интерфейс сервиса работы с Audit логом
 * @author larin
 *
 */
public interface AuditLogServiceDao {

    /**
     * Получение всех версий доменного объекта
     * @param domainObjectId
     * @return
     */
    List<DomainObjectVersion> findAllVersions(Id domainObjectId);

    /**
     * Получение конкретной версии по известному идентификатору
     * @param versionId
     * @return
     */
    DomainObjectVersion findVersion(Id versionId);

    /**
```

```
     * Очистка аудита доменного объекта.
     * @param domainObjectId
     */
    void clean(Id domainObjectId);

}
```

```
package ru.intertrust.cm.core.business.impl;
```

```
import java.util.List;

import ru.intertrust.cm.core.business.api.dto.DomainObjectVersion;
import ru.intertrust.cm.core.business.api.dto.Id;
import ru.intertrust.cm.core.business.api.dto.VersionComparisonResult;

/**
 * Интерфейс сервиса работы с Audit логом
 * @author larin
 *
 */
public interface AuditService {

    /**
     * Получение всех версий доменного объекта
     * @param domainObjectId
     * @return
     */
    List<DomainObjectVersion> findAllVersions(Id domainObjectId);

    /**
     * Получение конкретной версии по известному идентификатору
     * @param versionId
     * @return
     */
    DomainObjectVersion findVersion(Id versionId);

    /**
     * Очистка аудита доменного объекта.
     * @param domainObjectId
     */
    void clean(Id domainObjectId);

    /**
     * Получение информации об изменениях между текущей версией доменного
объекта и версией с переданным идентификатором
     * @param baseVersionId
     * @return
     */
    VersionComparisonResult compare(Id baseVersionId);

    /**
     * Получение информации об изменениях в двух разных версиях доменного
объекта по известным идентификаторам версий
     * @param baseVersionId
     * @param comparedVersionId
     * @return
     */
    VersionComparisonResult compare(Id baseVersionId, Id comparedVersionId);
```

```
}
```

3.12.2 Описание моделей, используемых при работе с AuditLog.

Класс, описывающий версию - DomainObjectVersion.

```
package ru.intertrust.cm.core.business.api.dto;

import java.util.Date;


/**
 * Интерфейс версии доменного объекта
 * @author larin
 *
 */
public interface DomainObjectVersion extends IdentifiableObject {

    /**
     * Идентификатор доменного объекта
     * @return
     */
    Id getDomainObjectId();

    /**
     * Получение дополнительной информации о версии (зарезервировано)
     * @return
     */
    String getVersionInfo();

    /**
     * Получение информации о компоненте, производившей изменения. Информация
     * берется из systemAccessToken
     * @return
     */
    String getComponent();

    /**
     * Получение IP адреса хоста, с которого выполнялась работа при
выполнении
     * изменений
     * @return
     */
    String getIpAddress();

    /**
     * Идентификатор персоны (тип Person) выполнившей изменение
     * @return
     */
    Id getModifier();

    /**
     * Возвращает дату модификации данного доменного объекта
     *
     * @return дату модификации данного доменного объекта
     */
    Date getModifiedDate();
```

```
}
```

Класс описывающий разницу между двумя версиями VersionComparisonResult

```
package ru.intertrust.cm.core.business.api.dto;

import java.util.Date;
import java.util.List;

/**
 * Иньерфейс описывающий разницу в версиях доменных объектов
 * @author larin
 *
 */
public interface VersionComparisonResult {
    /**
     * Возвращает идентификатор версии, с которой производится сравнение
     * @return
     */
    Id getBaseVersionId();

    /**
     * Возвращает идентификатор версии, которую сравниваю с базовой
     * @return
     */
    Id getComparedVersionId();

    /**
     * Возвращает идентификатор доменного объекта
     * @return
     */
    Id getDomainObjectId();

    /**
     * Получение идентификатора персоны, выполнившей изменения. в случае если
     * изменения производились от имени системы то null
     * @return
     */
    Id getModifier();

    /**
     * Дата сохранения изменений
     * @return
     */
    Date getModifiedDate();

    /**
     * Получение дополнительной информации о версии (зарезервировано)
     * @return
     */
    String getVersionInfo();

    /**
     * Получение информации о компоненте, производившей изменения. Информация
     * берется из systemAccessToken
     * @return
     */
    String getComponent();

    /**
     * Получение IP адреса хоста, с которого выполнялась работа при
```

```
выполнении
     * изменений
     * @return
     */
    String geyIpAddress();

    /**
     * Информация о изменившихся атрибутах
     * @return
     */
    List<FieldModification> getModifiedFields();

}
```

Класс описывающий изменение одного атрибута ModifiedAttribute

```
package ru.intertrust.cm.core.business.api.dto;

/**
 * Информация о изменившемся поле доменного объекта
 * @author larin
 *
 */
public interface FieldModification {
    /**
     * Получение имени атрибута
     * @return
     */
    String getName();

    /**
     * Получение значение атрибута базовой версии
     * @return
     */
    <T extends Value> T getBaseValue();

    /**
     * Получение значения атрибута сравниваемой версии
     * @return
     */
    <T extends Value> T getComparedValue();
}
```

3.12.3 Необходимые доработки в ядре и других сервисах

1. Необходимо создать метод getModifier() в доменном объекте и соответствующее поле в базе данных. Поле возвращает идентификатор персоны, выполнившей изменение. В случае если изменение было выполнено системным процессом (джобом или workflow) в поле сохраняется null.
2. Не хватает информации о текущем пользователе внутри spring бинов или иных классов, где нет доступа к SessionContext. Необходимо продумать механизм создания RequestContext или ThreadContext с этой информацией.
3. Не хватает информации о адресе машины, инициирующей запрос к серверу внутри spring бинов или иных классов. Необходимо продумать механизм создания RequestContext или ThreadContext с этой информацией.

3.13 Настройка файлов конфигурации

Этот раздел описывает структуру файлов конфигурации, их расположение в проекте, инструкции по настройке расположения файлов конфигурации.

3.13.1 Файлы конфигурации и их схемы

В системе используются два вида файлов конфигурации: файлы конфигурации ядра и файлы конфигурации модулей расширения.

Настройка расположения файлов конфигурации ядра и их схемы, а также файла конфигурации файлов конфигурации модулей расширения и его схемы осуществляется в спринг-бине *configurationSerializer*, ответственном за сериализацию/десериализацию файлов конфигурации. Данный спринг-бин описывается в *beans.xml* модуля *configuration*. Ниже приведен пример такой конфигурации:

```
<bean id="configurationSerializer" depends-
on="topLevelConfigurationCacheInitializer"
          class="ru.intertrust.cm.core.config.ConfigurationSerializer"
          p:coreConfigurationSchemaFilePath="config/configuration.xsd"
          p:modulesConfigurationSchemaPath="config/modules-configuration.xsd"
          p:modulesConfigurationFolder="modules-configuration"
          p:modulesConfigurationPath="/modules-configuration.xml">
        <property name="coreConfigurationFilePaths">
            <set>
                <value>config/system-domain-objects.xml</value>
                <value>config/domain-objects.xml</value>
                <value>config/collections.xml</value>
                <value>config/access.xml</value>
            </set>
        </property>
 </bean>
```

3.13.2 Файлы конфигурации ядра

Файлы конфигурации ядра находятся в ресурсах модуля *configuration* в папке *config*:

cm-sochi\Core\configuration\src\main\resources\config

Расположение каждого файла конфигурации ядра указывается в свойстве *coreConfigurationFilePaths* спринг-бина *configurationSerializer* в виде абсоютного пути к ресурсу относительно модуля *configuration*.

Т.к. все такие файлы находятся в папке *config,* настройки путей к файлам конфигурации ядра выглядит следующим образом:

```
<property name="coreConfigurationFilePaths">
    <set>
        <value>config/system-domain-objects.xml</value>
        <value>config/domain-objects.xml</value>
        <value>config/collections.xml</value>
        <value>config/access.xml</value>
    </set>
</property>
```

При добавлении нового файла конфигурации ядра его также следует создать в папке *config* и добавить соответствующее значение в множество значений свойства *coreConfigurationFilePaths*.

3.13.2.1 Схема файлов конфигурации ядра

Все файлы конфигурации проверяются на соответветсвие схеме файлов конфигурации. Путь к схеме файлов конфигурации указывается в свойстве *coreConfigurationSchemaFilePath* спринг-бина *configurationSerializer* в виде абсоютного пути к ресурсу относительно модуля *configuration*.

В данный момент используется файл схемы *configuration.xsd* расположенный в подпапке *config*. Соответственно конфигруция пути к схеме файлов конфигурации ядра выглядиит следующим образом:

```
p:coreConfigurationSchemaFilePath="config/configuration.xsd"
```

3.13.3 Файлы конфигурации модулей расширения

Файлы конфигурации модулей расширения расположены в ресурсах модуля *modules-configuration* в папке *config*:

```
cm-sochi\modules-onfiguration\src\main\resources\config\
```

Для каждого модуля расширения рекомендуется создавать отдельную одноименную папку в папке *config* и размещать в ней файлы конфигурации данного модуля. Например, для модуля расширения *extension-module* это будет папка:

cm-sochi\modules-onfiguration\src\main\resources\config\extension-module

3.13.3.1 Схемы файлов конфигурации модулей расширения

Файлы конфигурации модулей расширения так же как и файлы конфигурации ядра проверяются на соответствие схеме. По умолчанию используется схема файлов конфигурации ядра (указывается в свойстве *coreConfigurationSchemaFilePath* спринг-бина *configurationSerializer*), но может использоваться и собственная схема (это потребуется, если в кофигурации модуля расширения описаны собственные сущности, не описанные в конфигурации ядра). В этом случае схема файлов конфигурации модуля расширения должна импортировать схему файлов конфигурации ядра:

```
<xs:import namespace="https://cm5.intertrust.ru/config"/>
```

и переопределять элемент *configuration*, расширяя его новыми типами элементов, например:

```
<xs:element name="configuration" type="custom:configurationType"/>

<xs:complexType name="configurationType">
    <xs:complexContent>
        <xs:extension base="core:configurationType">
             <xs:choice minOccurs="0" maxOccurs="unbounded">
                 <xs:element name="test-type" type="custom:test-typeType"
                                 minOccurs="0" maxOccurs="unbounded"/>
             </xs:choice>
        </xs:extension>
```

```
    </xs:complexContent>
</xs:complexType>
```

Полный пример схемы файлов конфигурации модуля расширения, расширяющей схему файлов конфигурации ядра, и соответствующего файла конфигурации можно найти в папке *test-module*:

cm-sochi\modules-configuration\src\main\resources\config\test-module\custom-configuration.xsd

cm-sochi\modules-configuration\src\main\resources\config\test-module\custom-config.xml

3.13.3.2 Регистрация файлов конфигурации модулей расширения для загрузки ядром

Для того, чтобы файлы конфигурации модуля расширения были загружены ядром их необходимо зарегистривать в файле *modules-configuration.xml*:

cm-sochi\modules-configuration\src\main\resources\config\modules-configuration.xml

Данный путь (и имя файла) также является настраиваемым и указывается в свойстве *modulesConfigurationPath* спринг-бина *configurationSerializer*:

```
p:modulesConfigurationPath="/modules-configuration.xml"
```

Путь указывается относительно папки cm-sochi\modules-configuration\src\main\resources\config.

Ниже приведен пример регистрации файлов конфигурации модуля расширения:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<modules-configuration xmlns="https://cm5.intertrust.ru/modules-
configuration">

    <module-configuration>
        <path>/test-module/custom-config.xml</path>
        <schema-path>/test-module/custom-configuration.xsd</schema-path>
    </module-configuration>

    <module-configuration>
        <path>/test-module/domain-objects.xml</path>
    </module-configuration>

</modules-configuration>
```

В элементе *path* указывается путь к файлу конфигурации (относительно папки *cm-sochi\modules-configuration\src\main\resources\config*). В элементе schema-path указывается (опционально) схема для валидации этого файла конфигурации. Если путь к схеме не указан, используется схема файлов конфигурации ядра.

3.13.3.3 Некоторые детали реализации загрузки ядром файлов конфигурации модулей расширения

Для того, чтобы файлы конфигурации модулей расширения были доступны для загрузки загрузчиком классов ядра, они копируются во время сборки модуля *web-app* из модуля *modules-configuration* в папку */WEB-INF/classes/modules-configuration*. Для этого в *pom.xml* модуля *web-app* настроен соответствующий плагин. Далее для спринг-бина *configurationSerializer* в свойстве *modulesConfigurationFolder* указывается абсолютный

путь (относительно модуля *web-app*) к папке, где лежат файлы конфигурации модулей расширения, т. е. к папке *WEB-INF/classes/modules-configuration*:

```
p:modulesConfigurationFolder="modules-configuration"
```

Для валидации файла регистрации файлов конфигурации модулей расширения *modules-configuration.xml* используется схема *modules-configuration.xsd*. Путь к данной схеме также является настраиваемым и указывается как абсолютный путь (в модуле *configuration*) в свойстве *modulesConfigurationSchemaPath* спринг-бина *configurationSerializer*:

```
p:modulesConfigurationSchemaPath="config/modules-configuration.xsd"
```

3.14 Точки расширения

Точки расширения предназначены для обеспечение модернизации логики работы ядра системы и встраивания дополнительного функционала, который должен отрабатывать в те или иные моменты работы системы.

3.14.1 Создание точки расширения

Для создания точки расширения необходимо создать интерфейс точки расширения, расширяющий интерфейс ru.intertrust.cm.core.dao.api.extension.ExtensionPointHandler. В данном интерфейсе объявить метод точки расширения. Например, интерфейс точки расширения, вызывающейся после сохранения доменного объекта выглядит следующим образом:

```
package ru.intertrust.cm.core.dao.api.extension;
import ru.intertrust.cm.core.business.api.dto.DomainObject;

public interface AfterSaveExtensionHandler extends ExtensionPointHandler {
      void onAfterSave(DomainObject domainObject);
}
```

Далее, в том месте где необходимо вызвать все обработчики данной точки расширения необходимо вставить код получающий обработчики точки расширения с помощью spring сервиса ru.intertrust.cm.core.dao.api.ExtensionService и вызвать их исполнение. Например, код вызывающий обработчик точки расширения AfterSaveExtensionHandler выглядит следующим образом:

```
@Autowired
    private ExtensionService extensionService;
…
// Вызов точки расширения после сохранения
        AfterSaveExtensionHandler afterSaveExtension =
(AfterSaveExtensionHandler) extensionService
                .getExtentionPoint(AfterSaveExtensionHandler.class,
                        domainObject.getTypeName());
        afterSaveExtension.onAfterSave(domainObject);
```

В качестве параметров метода возвращающего обработчики точек расширения передается интерфейс точки расширения и фильтр. Фильтр предназначен для оптимизации данного механизма по скорости. Оптимизация заключается в том что вызываться будут не все точки расширения, которые имплементят интерфейс, переданный в первом параметре, а только те у которых совпадает значение фильтра с тем, который указан при объявлении обработчика точки расширения.

3.14.2 Создание обработчика точки расширения

Для создания обработчика точки расширения необходимо создать класс, имплементяший интерфейс нужной точки расширения и аннотировать его аннотацией ru.intertrust.cm.core.dao.api.extension. ExtensionPoint. В аннотации можно указать фильтр, когда вызывать данный обработчик. Если фильтр не указан данный обработчик будет вызываться всегда, когда вызываются обработчики данного типа. Например, реализация обработчика точки расширения, вызывающейся после сохранения доменного объекта может выглядеть следующим образом:

```
package ru.intertrust.cm.core.dao.impl.personmanager;

import ru.intertrust.cm.core.business.api.dto.DomainObject;

import ru.intertrust.cm.core.dao.api.extension.AfterSaveExtensionHandler;
import ru.intertrust.cm.core.dao.api.extension.ExtensionPoint;

@ExtensionPoint(filter="Person")
public class OnSavePersonExtensionPointHandler implements
AfterSaveExtensionHandler{

      @Override
      public void onAfterSave(DomainObject domainObject) {
            System.out.println("On after save for person " +
domainObject.getTypeName() + ":" + domainObject.getId());
      }
}
```

Обработчиков точек расширения может быть неограниченное множество, все они вызовутся. Порядок вызова точек расширения не декларируются. Точки расширения должны лежать в пакетах, являющимися дочерними к тому пакету, который указан в конфигурации spring бина ru.intertrust.cm.core.dao.impl.ExtensionServiceImpl в атрибуте basePackage. Вот так выглядит конфигурация на текущий момент в файле bean.xml проекта dao-impl.

```
    <bean id="extensionService"
class="ru.intertrust.cm.core.dao.impl.ExtensionServiceImpl">
        <property name="basePackage" value="ru/intertrust/cm" />
    </bean>
```

3.15 Validation API

3.15.1 Назначение

Validation API предназначено для проверки корректности данных вводимых пользователем. Проверка данных осуществляется в 2 этапа:

–на клиентской стороне — проверка на соответствие доменной модели (напр. «not empty», «max/min width», «max/min value» etc. Правила проверки данных на уровне клиента описываются на уровне доменной модели и являются едиными для всех визуальных представлений атрибутов доменной модели.

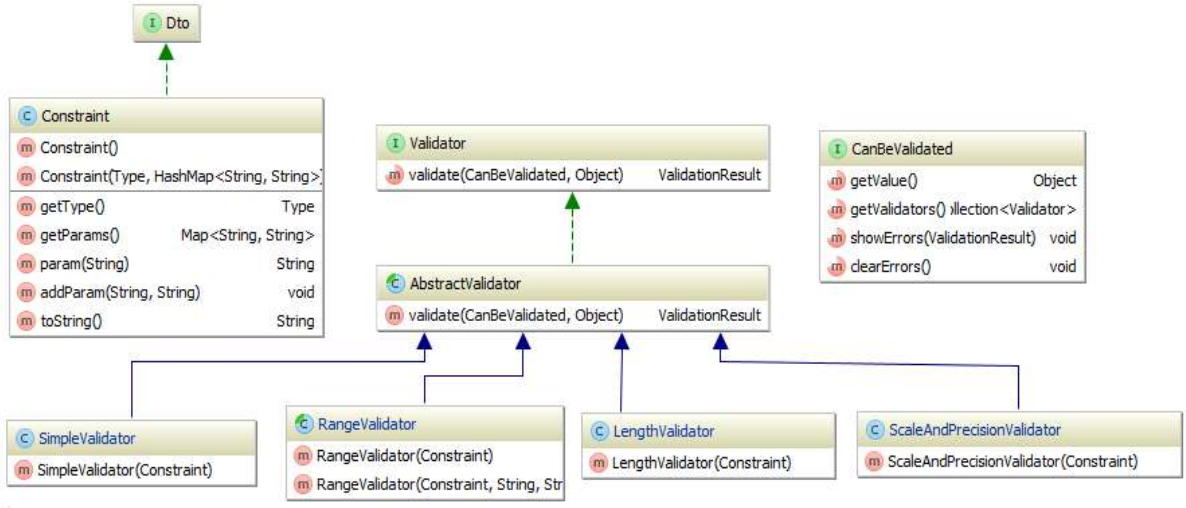
– на серверной стороне — проверка на соответствие бизнес процессам и бизнес логики (напр. «данные доменной модели не противоречат друг другу» etc). Включает в себя простую и сложную (кастомную) валидацию.

3.15.2 Client validation api

3.15.2.1 Базовые компоненты

Основные интерфейсы и классы клиентской части приведены на рис.1 и 2.

Рис. 1: Основные интерфейсы и класы client validation api



Dto – интерфейс, определяющий тип данных передаваемых на сервер (уже существующий).

Constraint – описывает ограничения (правила валидации), налагаемые на поле доменного объекта. Состоит из типа валидации (simple, length, range etc) и параметров(например, начало и конец диапазона).

CanBeValidated – интерфейс объекта, подлежащего валидации. Все виджеты реализуют этот интерфейс. Содержит методы: getValue() - возвращает значения для валидации (например, строка введенная пользователем в TextBox): getValidators() - возвращает список валидаторов, применимых к данному объекту; showErrors(ValidationResult) и clearErrors() - используются для визуального отображения результатов валидации.

Validator – определяет тип валидаторов данных клиента. Определяется в виде компонента (@Component), что позволяет создавать экземпляр прямо на клиенте. Содержит метод validate, на вход которого подаются экземпляр класса, реализующий CanBeValidated (в частности, widget) и дополнительные данные, которые могут понадобиться для проверки. Валидатор позволяет изменить внешний вид widget, вызвав метод showErrors или clearErrors в зависимости от результата проверки.

SimpleValidator – простейшая реализация интерфейса Validator, позволяющая проводить валидацию данных по заранее предопределенным словам ("not-empty", "integer", "positive-int" etc) или по регулярному выражению.

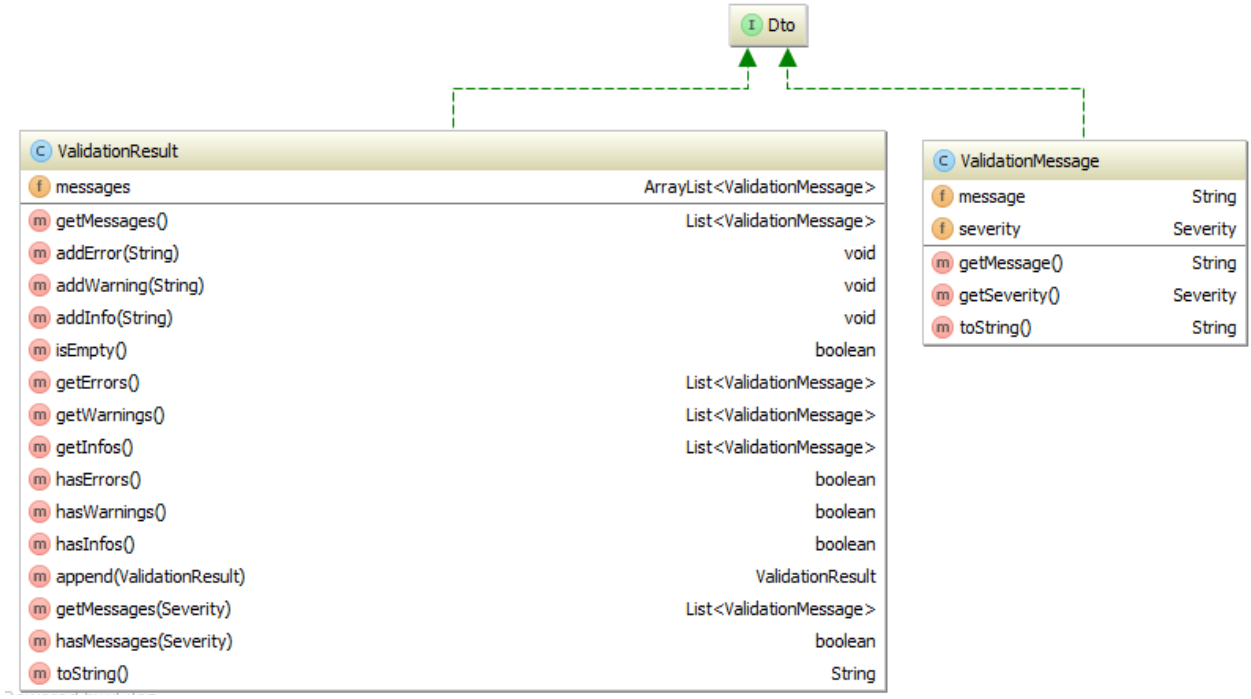
RangeValidator – валидатор диапазона значений.

LengthValidator – валидатор длины. Длина может проверяться на точное совпадение с заданным значением, либо на соответствие минимальному и/или максимальному значению.

ScakeAndPresionValidator - валидатор масштаба и точности для числа с плавающей запятой.

Всем реализациям валидатора в конструктор передается объект Constraint, из которого они извлекают необходимые параметры проверки.

Иногда возникает неоходимость осуществлять валидацию посредством асинхронного вызова сервера (например, даты в часовых поясах Олсона). В таких случаях класс валидатора также реализует общий интерфейс Validator. В методе validate() необходимо вызвать нужную серверную команду и определить AsyncCallback. Необходимо



реализовать validationRequest (Dto) и Handler, специфический для данной валидации.

Рис. 2: Основные интерфейсы и класы client validation api (продолжение)

ValidationResult – содержит информацию о результатах проверки (список объектов ValidationMessage), и имеет ряд вспомогательных методов (addError, addWarning, hasError, isEmpty etc).

ValidationMessage – сообщение результата проверки. Содержит непосредственно текст сообщения и уровень ошибки (SEVERITY, WARNING, INFO).

Severity – внутренний enum, определяет уровень сообщения: ошибка, предупреждение etc.

3.15.2.2 Требования к другим компонентам приложения

Бо**о**льшую часть необходимых проверок можно определить из конфигурации полей доменного объекта. В эту группу входят: проверка на обязательное заполнения поля, проверка максимальной длины для текстового поля, проверка числового формата для LongField и DecimalField, проверка масштаба и точности для DecimalField.

Для более сложных проверок при описании доменной модели, для ее полей, необходимо добавить тэг constraints, в котором перечислить все дополнительные правила проверки для атрибута, например:

```
<domain-object-type name="country" initial-status="Active">
   <fields>
       <string name="name" length="128">
```

```
                <constraints>
                    <simple-constraint value="not-empty"/>
                    <simple-constraint value="^[^0-9]*$"/>
                </constraints>
            </string>
            <date-time name="independence_day">
                <constraints>
                    <date-range start="01.01.1990" end="31.12.2014"/>
                </constraints>
            </date-time>
            <long name="population">
                <constraints>
                    <int-range start="1000"/>
                </constraints>
            </long>
            <string name="description" length="1024">
                <constraints>
                    <length min-value="10"/>
                </constraints>
            </string>
            <!-- остальные поля -->
        </fields>
            ...
    </domain-object-type>
```

Для поддержки новых тэгов испоользуются классы ConstraintsConfig, ConstraintConfig (абстрактный) и классы для конфигов разных типов валидации, которые наследуются от ConstraintConfig. Каждый такой конфиг определяет список constraints. В класс FieldConfig и некоторые его подклассы также добвлен метод getConstraints(), который возвращает объединенный список ограничений неявно заданных типом поля и ограничений заданных пользователем в аттрибутах поля и в тэгах `<constraints>` Это позволяет получить список необходимых валидаторов на этапе инициализации виджетов .

Каждый widget для поддержки валидации данных на стороне клиента должен реализовать интерфейс CanBeValidated и содержать метод getValidators, который возвращает коллекцию валидаторов, применимых к данному виджету в соответствии с заданными ограничениями (constraints).

В виджеты также добавлены методы showErrors (выводит сообщение и/или каким-либо способом изменяет вид виджета сигнализируя об ошибке) и clearErrors (возвращает виджет к обычному виду).

3.15.2.3 Последовательность выполнения проверки

На уровне виджетов

В обработчике определенного события (например, потеря фокуса) выполняется проверка всех constraint, и если обнаружатся ошибки, изменяется вид виджета. Если введенное значение корректно, виджет отображается в нормальном виде.

На уровне Действий

По умолчанию считается, что все Action в приложении подлежат немедленному исполнению, без валидации данных. Флажок-признак необходимости валидации задается в конфигурации action с помшощью атрибута validate (по умолчанию false), либо устанавливается в ActionConfigBuilder-е. Если для Action необходима валидация данных, он вызывает утилитный метод вспомогательного класса, на вход которого передается

плагин, которому принадлежит данный Action. Метод перебирает все widget, которые содержит плагин, с учетом вложенности

Для каждого виджета вызывается метод getValidators(). Для каждого возвращенного валидатора вызывается мето validate(), который осуществляет проверку данных и, при необходимости, изменяет визуальное представления widget и формирует сообщения, дружественные пользователю.

3.15.3 Server validation api

3.15.3.1 Базовые компоненты

Основные интерфейсы и классы server validation api приведены на рис.3

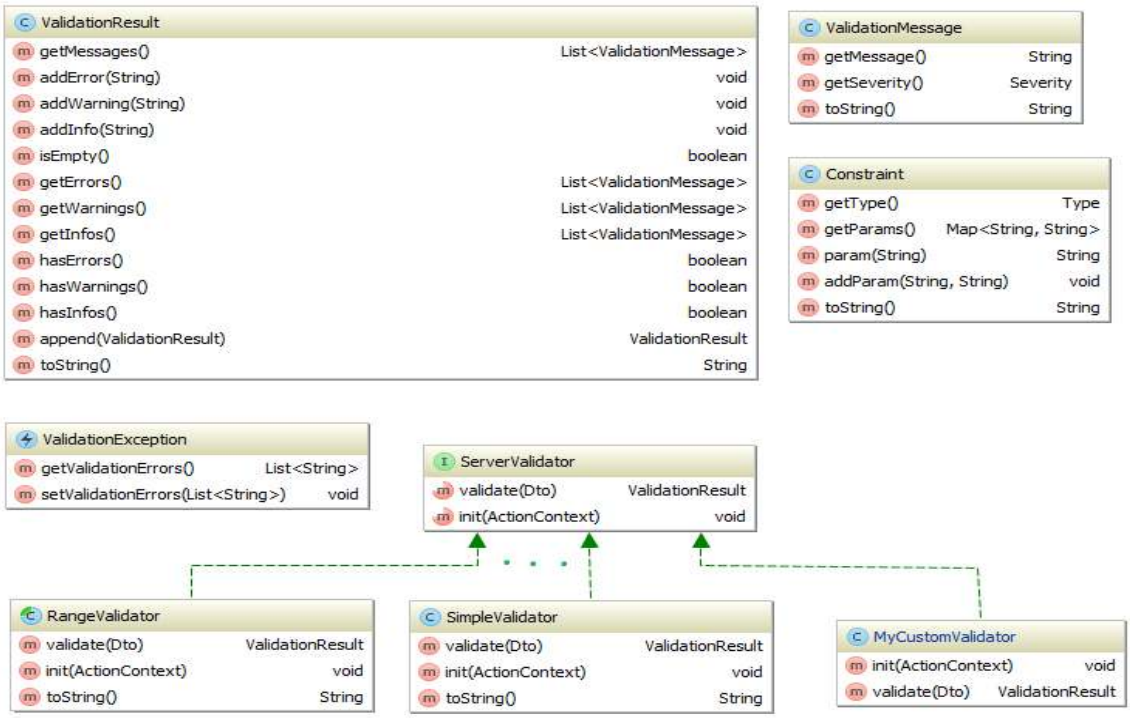


Рис. 3: Основные интерфейсы и классы server validation api

Здесь описываем только новые сущности

ValidationException – exception, который выбрасывается при обнаружении ошибок валидации.

ServerValidator – интерфейс определяющий тип валидатора данных на соответствие бизнес процессам и бизнес логике. Включает метод init(ActionContext context) и validate(Dto dto).

SimpleValidator, RangeValidator, LengthValidator, ScaleAndPrecisionValidator – простые реализации серверного валидатора.

Кастомные валидаторы, предоставляемые пользователем для сложной серверной валидации, должны реализовать интерфейс ServerValidator так же, как и обычные валидаторы,. Информация о классах кастомных валидаторов предоставляется в конфигурационном файле.

```
<validators>
```

```
<validator class="fully-qualified class name" widget-id="widget-id" />
</validators>
```

Тэги валидаторов объявляются внутри тэгов <action>.

Интерфейс ServerValidator применим к любым Dto, но на начальном этапе будем валидировать только WidgetState.

3.15.3.2 Требование к другим компонентам приложения

ActionConfig — содержит перечень конфигураций валидаторов, которые должны быть вызваны при обработке данных на сервере.

WidgetState — содержит список constraints.

3.15.3.3 Последовательность выполнения проверки

При обработке Action на сервере последовательно вызываются все зарегистрированные валидаторы (как обычные, так и кастомные), и результаты проверки накапливаются в ValidationResult.

Получение простых валидаторов: итерируем по всем объектам WidgetState, получаем список constraints и создаем на их базе валидаторы. Получение сложных валидаторов: берем список ValidatorConfig из ActionConfig и создаем экземпляры валидаторов по указанным именам классов.

После окончания всех проверок, в зависимости от состояния ValidationResult продолжается выполнение Action на сервере, или формируются дружественные для пользователя сообщения и отправляются на клиент, где они выводятся в отдельном модальном окне ошибок.

Изменение внешнего вида индивидуальных виджетов в принципе возможно, но на данном этапе слишком трудоемко и не целесообразно до полноценной реализации Action API.

3.15.4 Определение зависимости валидаторов от бизнес логики

В зависимости от бизнес состояния документа, часть валидаторов, как на серверной, так и на клиентской части, могут быть пропущены. Для этого при декларировании валидаторов предлагается добавить внутренний тег use, в котором перечислить условия использования валидатора. Например:

```
<constraints>
      <constraint component-name="number.range" minValue=0 maxVal=1000>
            <use field="status" in="DRAFT, APPROVED"/>
      </constraint>
      <constraint component-name="simple.constraint" reg-exp="[/d]+"/>
</constraints>
```

В первом варианте эта функциональность не предусматривается, но, возможно, понадобится в будущем, т.к. валидаторы (особенно кастомные) могут быть как угодно сложными.

3.15.5 Локализация

Примечание: *в рамках данных требований локализация реализована лишь в объеме необходимом для самой валидации. Весь механизм локализации подлежит дальнейшему обсуждению и уточнению.*

Для получения строковых ресурсов (в частности, сообщений об ошибках) используется класс MessageResourceProvider со статическими методами

```
            public static  Map<String, String> getMessages(locale);
            public static  Map<String, String> getMessages();  // для локали
по умолчанию
```

```
            public static String getMessage(String messageKey);
```

Сообщения будут подгружаться с property-файла для указанной локали и локали по умолчанию. Значения по умолчанию также можно захардкодить.

Возможный механизм использования строковых ресурсов на стороне клиента:

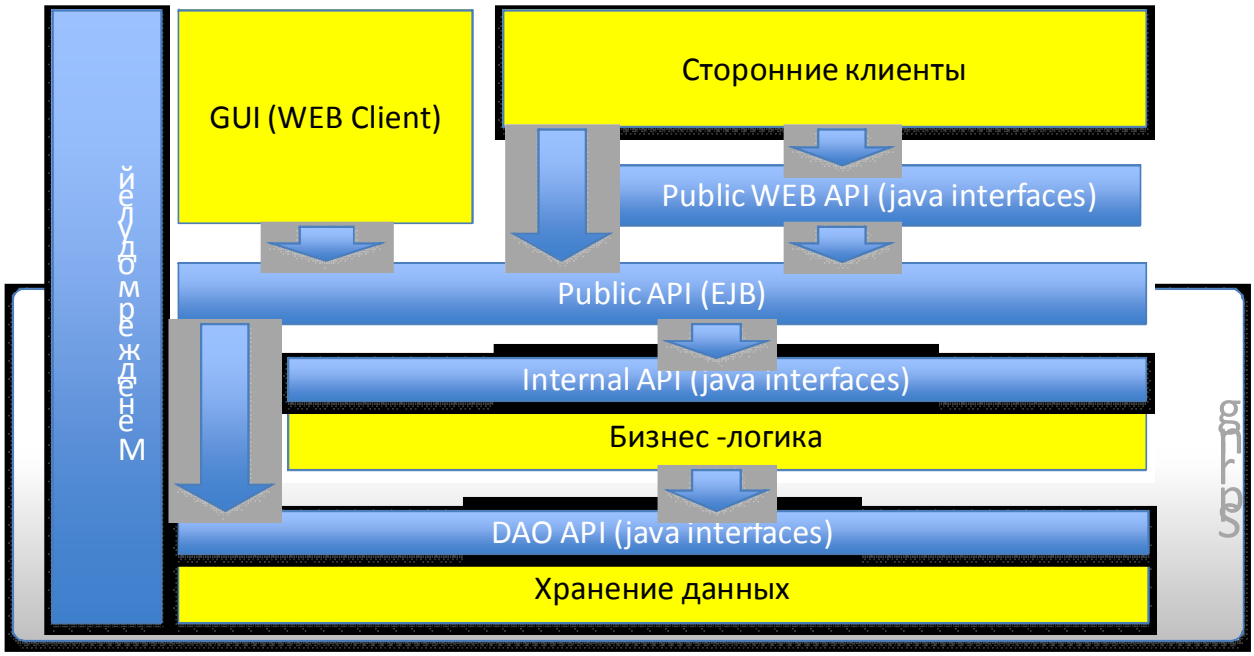
вызвать MessageResourceProvider.getMessages() в методе buildDomainObjectForm класса FormRetriever и передать результат в конструктор FormState. Там он будет доступен для FormPanel, который в свою очередь передаст его в виджеты.

Сообщения могут содержать поля для подстановки. Для их заполнения конкретными значениями будет использоваться утильный класс PlaceholderResolver.

4 Модель реализации

4.1 Обзор

Высокоуровневая диаграмма декомпозиции платформы AF5 на слои



Слои (логические) следующие:

- Конфигурация (возможно использование Apache Commons Configuration)
- Бизнес-логика (EJB + Spring). EJB - декларативная транзакционность, Spring - расширяемость бизнес-логики. Бизнес-логика интегрирована с Activiti, отвечающим за Workflow
- Подменяемый слой DAO - в том числе отвечающий за автоматическое создание мета-информации - таблиц, ключей, индексов
- Слой DAO интегрируется с подсистемой фильтрации по правам. Возможно, фильтарция тоже будет вынесена в отдельный слой (компонентом, модулем), но задача не первоочередная
- Слой GUI, реализующий конфигурируемый интерфейс пользователя разработан с использованием Google Web Toolkit (GWT) и описан в отдельном документе.

4.2 Слои

В таблице перечислены компоненты системы с указанием слоёв и подсистем, к которым они относятся.

Компонент	Слой	Важн.	Описание (с примечаниями)	Подсистема	Используется
AttachmentContentDao	DAO	осн.	Хранение вложений (2 реализации - для файловой системы и для IBM FileNet)	Вложения	AttachmentService, ImportDataService

AuditLogServiceDao	DAO	доп.	Доступ к истории изменений ДО	Журналирование	AuditService
AuthenticationDao	DAO	доп.	Управление объектами аутентификации	Аутентификация	AuthenticationService
CollectionsDao	DAO	осн.	Получение коллекций ДО	Коллекции	CollectionsService
ConfigurationDao	DAO	осн.	Управление конфигурацией	Конфигурация	ConfigurationControlService
DataStructureDao	DAO	служ.	Управление структурами данных в хранилище (БД)	Хранилище	DomainObjectDao
DoelEvaluator	DAO	служ.	Вычисление DOEL-выражений (DOEL-выражения используются в конфигурации, шаблонах уведомлений…)	Вычисления над ДО	многие компоненты
DomainObjectCacheService	DAO	всп.	Транзакционный кэш ДО	Хранилище	DomainObjectDao
DomainObjectDao	DAO	осн.	Основные операции с ДО	Хранилище	CrudService и другие
DomainObjectFinderService	DAO	служ.	Поиск ДО различными способами	Вычисления над ДО	NotificationService…
DomainObjectTypeIdCache	DAO	всп.	Кэш типов ДО	Хранилище	многие компоненты
DomainObjectTypeIdDao	DAO	всп.	Сопоставление числовых идентификаторов типов ДО именам	Хранилище	многие компоненты
ExtensionService	DAO	служ.	Сервис точек расширения	Расширения	DomainObjectDao, AttachmentContentDao, потенциально - другие
IdGenerator	DAO	всп.	Создание уникальных идентификаторов	Хранилище	компоненты DAO
PersonManagementServiceDao	DAO	осн.	Управление пользователями и группами	Управление пользователями	PersonManagementService
PersonServiceDao	DAO	доп.	Кэш пользователей	Управление пользователями	PersonService
StatusDao	DAO	всп.	Сопоставление числовых идентификаторов статусов именам		
UserTransactionService	DAO	служ.	Выполнение действий при завершении транзакции	Ядро	DynamicGroupService и др. компоненты подсистемы прав; AttachmentContentDao и др.
AccessControlService	DAO	служ.	Служба контроля доступа	Права доступа	DomainObjectDao, CollectionsDao, а также все компоненты, взаимодействующие с ними
ContextRoleCollector	DAO	служ.	Вычисление контекстных ролей	Права доступа	
DynamicGroupCollector	DAO	служ.	Вычисление динамических групп	Права доступа	
DynamicGroupService	DAO	служ.	Пересчёт динамических групп	Права доступа	

PermissionServiceDao	DAO	служ.	Сервис обновления списков доступа	Права доступа	
UserGroupGlobalCache	DAO	всп.	Кэш групп пользователей	Права доступа	
ConfigurationExplorer	Conf	осн.	Чтение конфигурации	Конфигурация	многие компоненты
ModuleService	Conf	служ.	Реестр модулей системы	Расширения	ConfigurationExplorer, ExtensionService, …
AttachmentService	API	осн.	Сервис работы с вложениями	Вложения	клиенты
AuditService	API	доп.	Сервис журнала изменений ДО (позволяет только получать исторические данные, не управлять ими)	Журналирование	клиенты
AuthenticationService	API	доп.	Сервис управления данными аутентификации	Аутентификация	
CollectionsService	API	осн.	Сервис получения коллекций ДО	Коллекции	клиенты, …
ConfigurationControlService	API	осн.	Сервис загрузки (изменения) конфигурации	Конфигурация	
ConfigurationService	API	осн.	Сервис получения конфигурации	Конфигурация	клиенты, …
CrudService	API	осн.	Сервис работы с ДО	Хранилище	клиенты
IdService	API	всп.	Сервис формирования идентификаторов (необходим для сокрытия реализации Id)		клиенты
ImportDataService	API	доп.	Сервис пакетной загрузки данных	Импорт данных	клиенты
NotificationService	API	доп.	Сервис отправки уведомлений	Уведомления	клиенты, …
NotificationTextFormer	API	служ.	Сервис формирования текста сообщений	Уведомления	NotificationService, …
PermissionService	API	доп.	Сервис определения прав доступа (позволяет превентивно определить возможность выполнения операции)	Права доступа	клиенты
PersonManagementService	API	осн.	Сервис управления пользователями и их принадлежностью к группам	Управление пользователями	клиенты
PersonService	API	служ.	Сервис получения данных о пользователях	Управление пользователями	клиенты
ProcessService	API	осн.	Сервис управления процессами (Activiti)	Процессы	клиенты
ProfileService	API	доп.	Сервис профилей пользователей	Профили пользователей	NotificationTextFormer, …
ReportService	API	доп.	Сервис формирования отчётов	Отчёты	клиенты
ReportServiceAdmin	API	доп.	Сервис управления отчётами	Отчёты	клиенты
ScheduleService	API	доп.	Сервис управления периодическими задачами	Периодические задачи	клиенты

ScriptService	API	служ.	Сервис выполнения скриптов	Вычисления над ДО	NotificationTextFormer, потенциально - другие клиенты
SearchService	API	осн.	Сервис поиска	Поиск	
TriggerService	API	служ.	Сервис проверки условий	Расширения	
UrlFormer	API	служ.	Сервис формирования ссылок	Уведомления	NotificationTextFormer

*) Важность

осн. Основной функционал системы
доп. Дополнительный функционал, доступный клиентам
служ. Важный служебный компонент
всп. Вспомогательный служебный компонент

На следующем рисунке представлена диаграмма слоёв, подсистем и основных зависимостей между ними в платформе AF5