Концепция СМ5(прототип)

[Концепция системы 1](#_Toc354565056)

[Глоссарий 3](#_Toc354565057)

[Система разграничения прав 5](#_Toc354565058)

[Настройка прав доступа 9](#_Toc354565059)

[Модификация настройки прав 16](#_Toc354565060)

[Недостатки новой системы прав доступа 17](#_Toc354565061)

[Бизнес-процессы в системе 18](#_Toc354565062)

[Activiti 19](#_Toc354565063)

[ToDo 23](#_Toc354565064)

[Версионность данных 24](#_Toc354565065)

[Наследование типов 26](#_Toc354565066)

[Серверные точки расширения 26](#_Toc354565067)

[Автоматическое заполнение атрибутов 27](#_Toc354565068)

[Требования к ядру 28](#_Toc354565069)

[Экспорт и импорт объектов 29](#_Toc354565070)

[Язык ссылок 30](#_Toc354565071)

# Концепция системы

**Общее описание системы**

Система автоматизации документооборота CM5(прототип) (далее «система») представляет собой автоматизированную многопользовательскую систему на СПО, сопровождающую процесс управления документооборотом организации с целью обеспечения выполнения этой организацией своих функций.

Система имеет модульную структуру, что позволяет собрать для конкретного заказчика необходимый функционал, как из кубиков, за непродолжительное время, а так же, предполагает возможность разработки дополнительных функциональных модулей для включения в систему без переработки (значительного изменения кода) уже существующих модулей. С одной стороны это позволяет максимально удовлетворить нужны заказчика, с другой - расширяется набор модулей для дальнейшего использования в системе. Разработка дополнительных функциональных модулей может вестись, как компанией разработчиком системы, так и партнерами. Для разработки дополнительных модулей партнерами должен быть разработан API.

Система реализуется как тонкий клиент и работает через браузер. Система работает в двух вариантах:

* по принципу организации отдельного рабочего стола (см. рис.1)
* упрощенная версия, без организации рабочего стола (см. рис.2)

*. . .  
Рисунок 1 . Внешний вид системы с организацией рабочего стола*

На рабочем столе можно настраивать фон и располагать ярлыки приложений, основным приложением является Document Box (оно открыто на рисунке).

*. . .  
Рисунок 2 .Внешний вид системы без организации рабочего стола*

Интегратор выбирает и поставляет заказчику один из вариантов в соответствии с его пожеланиями.

**Многозадачность**

Система поддерживает многозадачность – то есть одновременное выполнение нескольких задач для одного пользователя. Таким образом, пользователь может зайти в Личный кабинет(ЛК), поработать в нем, в случае необходимости перейти в поиск по атрибутам и запустить поисковый запрос, не дождавшись ответа системы, вернуться в личный кабинет и продолжить в нем работать (при возврате в ЛК пользователь попадает на страницу, с которой он ушел), при этом, когда он вернется в поисковое приложение, то увидит там результат выполнения своего поискового запроса.

**Набор базовых компонентов системы**

Если рассматривать систему в общем, то система состоит из двух частей:

* Пользовательская часть – в ней пользователи работают с документами – просматривают, редактируют, выполняют свои задания, осуществляют поиск документов и т.д.
* Административная часть - пользователи с соответствующими ролями осуществляют администрирование и настройку системы. Административная часть, в свою очередь подразделяется на две подсистемы:

1. Администрирование системы - создание/редактирование/удаление пользователей, настройка прав пользователей, делегирование прав, ведение справочников, журнал действий пользователей, статистика действий пользователей
2. Настройка работы системы – создание/редактирование шаблонов документов, создание/редактирование характеристик, настройка автоматизаций, создание/редактирование ЖЦ, настройка внешнего вида шаблонов документов.

**Базовые компоненты системы**

1. Document Box - основное рабочее приложение содержит:
   * Личный кабинет
   * Кабинет помощника
   * Документы - различные представления данных
   * Справочники
   * Поиск
2. Настройки приложений
3. Мессенджер
4. Администрирование системы, состоит из 2-х компонентов:
   * Администрирование системы
   * Настройка системы

**Пользовательская часть**

Пользовательская часть системы состоит из компонентов. Компоненты в свою очередь состоят из функциональных модулей. Каждый компонент может состоять из одного или нескольких модулей, при этом один модуль может влиять на работу нескольких компонентов.

Система включает в себя следующие функциональные модули:

* Работа с входящей, исходящей, внутренней корреспонденцией
* Работа с рассмотрением
* Работа с подписанием
* Работа с согласованием
* Работа с ознакомлением
* Работа с рассылкой
* Работа с поручениями
* Работа со справочниками
* Хранение/отображение истории карточек документов
* Взаимодействие с МЭДО
* Работа с ЭЦП
* Конструктор отчетов
* Печатные формы документов
* Поиск
* Версионность документов (вложений)
* Хранение и работа с архивной информацией
* Система уведомлений о поступивших документах - кабинеты пользователей
* Система уведомлений о поступивших документах по e-mail
* Оповещение о поступивших документах - всплывающие окошки как в outlook или skype
* Система обмена сообщениями (мессенджер)
* Настройка внешнего вида приложений и рабочего стола

**Административная часть**

Административная часть системы состоит из компонентов. Компоненты в свою очередь состоят из функциональных модулей.

**Администрирование системы:**

* Модуль работы с пользователями
* Модуль настройки прав пользователей,
* Модуль делегирования прав,
* Модуль работы со справочниками (учитывать версионность справочников)
* Журнал действий пользователей,
* Статистика действий пользователей

**Настройка работы системы:**

* Модуль работы с шаблонами документов,
* Модуль работы с характеристиками,
* Модуль работы с WorkFlow
* Модуль настройки автоматизаций,
* Модуль по настройке отображения документов

# Глоссарий

В этом разделе определяются термины разрабатываемой системы.

* **Карточка** (SysObject, предлагается переименовать в DataObject или найти иное хорошее название) - базовая единица хранения в системе. Представляет из себя набор значений атрибутов. Кроме того, имеет специальные характеристики: тип (фиксированный, задаётся при создании карточки), статус (меняется на протяжении существования карточки, скорее всего, в соответствии с её *жизненным циклом*), ссылка на родительскую карточку (необязательная, фиксированная, задаётся при создании карточки). Набор атрибутов, входящих в карточку, определяется её типом и, возможно, дополнительными *аспектами*, подключёнными к карточке.
* **Атрибут** - описание единичного элемента данных в системе. Не следует путать атрибут со значением атрибута. Из значений атрибутов состоят карточки, сам же атрибут определяет тип данных и их предназначение (семантику). Реализуемые типы данных: текст, число (целое; потребность в дробях пока не очевидна), дата, бинарные данные (файл), значение из справочника, ссылка на карточку. Последние два типа данных допускают хранение нескольких значений.

***Примечание:*** *Б*о*льшая часть справочников, очевидно, будет реализована на карточках, т.к. требует богатого функционала: редактирование пользователями, хранение многочисленной и разнородной информации по одному элементу, иерархическая организация элементов и т.п. Для выбора значения из таких справочников будет использоваться тип данных "ссылка на карточку". Тем не менее, существует ряд мест, где требуется просто выбор из некоторого ограниченного и достаточно статического набора значений: да/нет, срочно/весьма срочно/немедленно и т.п. Использование справочных карточек для хранения простых строковых подстановок мне представляется неразумным. Кроме того, в списке отсутствует широко применявшийся в предыдущей версии системы тип "пользователь". Поскольку каждому пользователю системы соответствует специальная карточка, я считаю целесообразным в карточках хранить ссылки именно на эти карточки.*

* **Документ** - основная смысловая единица системы. Существует внутри системы в виде иерархии карточек. Операции, ориентированные на конечного пользователя системы, должны оперировать именно с документами, а не с отдельными карточками.
* **Жизненный цикл** - ориентированный граф, вершинами которого являются статусы карточки, а рёбрами - переходы между ними. Каждая карточка в обязательном порядке имеет собственный жизненный цикл, возможно, вырожденный - состоящий из единственного статуса и не имеющий переходов. Переходы карточки по этапам её жизненного цикла могут осуществляться как пользователем системы (через GUI), так и бизнес-процессами, работающими под управлением Activiti.

***Вариант:*** *Можно отказаться от ручного перевода карточки между статусами, но в таком случае должен существовать отдельный бизнес-процесс, управляющий карточкой на протяжении всего её жизненного цикла. Причём, на мой взгляд, такой бизнес-процесс должен быть должен быть достаточно гибким, чтобы уметь включать или исключать отдельные куски на основании, например, аспектов карточки или просто её содержимого - значений конкретных атрибутов.*

* **Процесс** (бизнес-процесс, workflow) - автоматизированная процедура обработки карточки или документа. Процесс всегда присоединяется к определённой карточке, но может создавать или управлять другими карточками, связанными с данной. Процесс задаётся в нотации BPMN 2.0 и содержит набор этапов (активностей, activities), которые могут быть автоматическими или ручными. Исполнение автоматических этапов производится внутри системы в асинхронном режиме с использованием движка Activiti. Карточки (документы), находящиеся на ручных этапах, помещаются в специальные папки системы и при отображении в GUI предлагают пользователю возможность произвести необходимые действия для завершения этапа.
* **Действие** - процедуры, выполнение которых может быть инициировано пользователем системы. К ним относятся:
  + создание, сохранение (изменение) и удаление карточки;
  + назначение процесса карточке (документу);
  + другие простые и составные действия.

Действия являются объектом назначения прав в системе. Они могут выполняться как из GUI системы, так и её внешними клиентами через специальный API. Список действий, доступных пользователю для данной карточки, может быть также получен через API системы.

* **Пользователь** - специальный объект в системе, хранящий основную информацию о пользователе системы - человеке, имеющим право выполнять вход в систему (login). Кроме того, для всех пользователей создаются *карточки*, хранящие дополнительную информацию об этом человеке, включая его контактные данные, привязку к организационно-штатной структуре, специфические настройки и т. п.

***Примечание:*** *Существуют некоторые сомнения относительно необходимости специального объекта пользователя. Единственная информация, которую содержит этот объект, и которая не дублируется в карточке пользователя, это его пароль и, возможно, логин в системе. Эта информация необходима для осуществления аутентификации пользователя через наши внутренние механизмы, но в случае подключения внешнего каталога пользователей она вовсе уходит из системы. С учётом этого есть смысл рассмотреть реализацию этого объекта исключительно в рамках подсистемы внутренней аутентификации пользователей, если она необходима.*

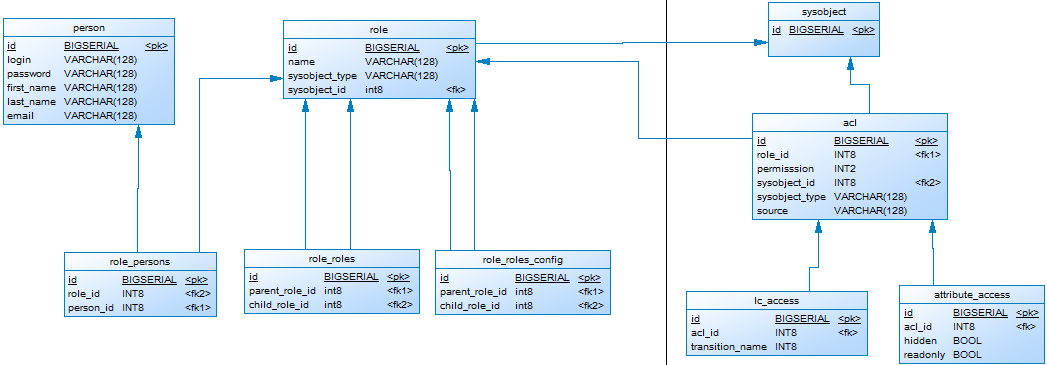
* **Аспект** - предполагаемая структура, состоящая из набора атрибутов и, возможно, включений в жизненный цикл карточки. Аспекты могут подключаться к карточке при её создании либо позже (реализацию второго варианта можно отложить на последующие этапы разработки или, по крайней мере, сильно ограничить).

# Система разграничения прав

Опыт эксплуатации предыдущих версий реализаций прав выявил, что это один из самых медленных элементов системы. Предлагаемый подход не зависит от структуры хранения данных и обеспечивает 100% потребностей заказчиков в настройке прав доступа.

**Основные подходы**

Самая ресурсоемкая задача при определения прав доступа - это получение списка документов, на которые есть права не ниже read у текущего пользователя. В результирующий массив может попасть большое число документов и нагрузка сильно возрастает при увеличении объема базы, поэтому при проектировании будем опираться на оптимизацию именно данной функции. Данную задачу должна выполнить СУБД, затратив минимум ресурсов. Для этого вводим таблицу, в которой будут храниться настроенные права на документ в удобном для запросов виде. Модель данных предоставлена на рисунке 1.

  
Рисунок 1.

Права доступа хранятся в группе таблице acl. Таблица заполняются специальным сервисом, который будет описан отдельно. Особенность данного подхода - это то, что права раздаются не персонам, а контекстным ролям. Контекстные роли формируются отдельным сервисом и хранятся в базе данных. Данный сервис так же будет описан ниже. Контекстные роли предназначены для минимизации затрат на перерасчет прав, которые возникают при изменении таких сущностей, изменение которых может повлечь пересчет прав на большое количество документов. Например, добавление или удаление делегатов, смена руководителя подразделения или организации и т.д. При таком подходе не меняется ACL документа, а только состав контекстной роли. Роли могут включать другие роли, при этом само включение отображается в таблице role\_roles\_config, и при этом корректно заполняется специальным сервисом таблица role\_roles, с учетом наследования состава включаемой роли. Данный подход позволяет так же упростить запросы на получение данных, исключая необходимость выполнять рекурсивные запросы. При использовании данной модели данных запрос документов, на которые есть права у текущего пользователя, приводится к виду:

select s.\*

from sysobject s

inner join acl a on (s.acl\_id = a.id)

inner join role\_roles rr on(rr.parent\_role\_id = a.role\_id)

inner join role\_persons rp on(rp.role\_id = rr.child\_role\_id)

where rp.person\_id = <id текущего пользователя>

или

select s.\*

from sysobject s

where exists (

select 1 from acl a

inner join role\_roles rr on(rr.parent\_role\_id = a.role\_id)

inner join role\_persons rp on (rp.role\_id = rr.child\_role\_id)

where s.acl\_id = a.id

and rp.person\_id = <id текущего пользователя>)

Данный подход требует от системы содержания в актуальном состоянии записей в таблице acl. За этим следит сервис назначения прав доступа. Данный подход позволяет давать права как контекстным ролям, так и статическим ролям не усложняя базу данных и не добавляя условия в запрос. Сервис назначения прав. Далее, под назначение прав я буду понимать заполнение таблицы acl сервисом назначения прав далее PermissionService, по правилам, описанным в матрице прав доступа. Для этого вводим понятие матрица прав доступа. Матрица прав доступа - это таблица, в которой определяется алгоритм формирования записей в таблице acl в зависимости от статуса документа и динамической группы. Под динамической группой здесь понимается именованное правило получения списка контекстных ролей. Динамическая группа не хранится в базе данных, и используется только для более удобного описания правил заполнения таблицы acl. Матрица строится для каждого типа документа. Матрица представлена в виде xml файла и установлена как конфигурация системы. Пример матрицы для одного, абстрактного, типа документа приведен в таблице 1.

Таблица 1:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Статус\Роль** | **Автор** | **Согласующий** | **Подписывающий** | **Руководитель автора документа** |
| Проект | W | R |  |  |
| На согласовании | R | R | R |  |
| На доработке | W |  |  |  |
| На подписании | R | R | R | R |
| В архиве | R | R | R | R |

В матрице указывается, какие права есть на документ данного типа в определенном статусе и для определенной динамической группы. Отдельно о динамических группах. Как я уже написал, это именованное правило получения списка контекстных ролей, получаемая по определенному алгоритму. Таких алгоритмов может быть бесчисленное множество, но все эти алгоритмы покрываются способами, перечисленными в таблице 2.

Таблица 2

|  |  |  |
| --- | --- | --- |
| **№** | **Способ получения состава динамической группы** | **Информация, необходимая для получения состава динамической роли** |
| 1 | Получение контекстной роли текущего документа | Имя контекстной роли, идентификатор текущего документа. |
| 2 | Получение контекстной роли связанного документа | Имя атрибута типа ID, имя контекстной роли, идентификатор текущего документа |
| 3 | Получение контекстных ролей с помощью запроса в СУБД. Запрос строится относительно текущего документа. | Sql запрос, идентификатор текущего документа, запрос должен возвращать идентификаторы контекстных ролей (role.id) |
| 4 | Получение контекстных ролей с помощью специального java класса. Типовой случай применения - дать права всем вышестоящим руководителям вверх по иерархии подразделений. | Имя java класса, идентификатор текущего документа. |

Класс имплементит интерфейс SysobjectDynamicGroup приведенный ниже. Метод getContextRoles(long sysobjectId) принимает идентификатор текущего документа и возвращает список идентификаторов контекстных ролей.

package ru.intertrust.cm5.core;

import java.util.List;

import ru.intertrust.cm5.core.dto.ServerException;

public interface SysobjectDynamicGroup {

public List<Long> getContextRoles(long sysobjectId) throws ServerException;

}

После сохранения нового и при смене статуса существующего документа права пересчитываются полностью. Для этого:

* из настроек получается матрица прав для определенного типа документа;
* из матрицы выбирается строка, соответствующая статусу документа;
* формируется цикл по всем динамическим группам в пределах выбранного статуса;
* для каждой динамической группы по одному из алгоритмов описанных в таблице 2 определяется набор контекстных ролей;
* полученные идентификаторы контекстных ролей записываются в таблицу acl.

При сохранение существующего документа без смены статуса происходит перерасчет прав выборочно, а именно для тех динамических групп у которых алгоритм получения контекстных ролей соответствует пункту 2 таблицы 2 (контекстная роль связанного документа). К тому же этот пересчет выполняется только в случае изменения этих атрибутов.

**Сервис формирования контекстных ролей**

Данный сервис предназначен для создания, и поддержания в актуальном состоянии контекстных ролей. Как я уже раньше было написано контекстная роль – это именованная роль, привязанная к произвольному sysobject. Контекстная роль содержит ссылки на персоны и хранится в базе данных. Состав контекстной роли определяется настройкой, и хранится в системе в качестве XML конфигурации. Конфигурации определяет имя контекстной роли, к какому типу она относится и алгоритм ее получения. Так же в конфигурации определяется событие, когда необходимо пересчитывать данную контекстную роль. При получении состава контекстной роли можно воспользоваться возможностью включения роли в роль, соответственно при формировании состава контекстной роли можно получать как список карточек, и у данных карточек получать определенные роли, дили непосредственно получать список пользователей. Алгоритмы классифицируются по способу получения состава контекстной роли и определены в таблице 3.

Таблица 3

|  |  |  |
| --- | --- | --- |
| **№** | **Способ получения состава контекстной роли** | **Информация, необходимая для получения состава контекстной группы** |
| 1 | Пользователь, хранящийся в атрибуте sysobject | Имя атрибута, идентификатор sysobject |
| 2 | Пользователь, хранящийся в связанном sysobject | Имя атрибута текущего документа, имя атрибута связанного документа, идентификатор sysobject |
| 3 | SQL запрос, с помощью которого получаются все пользователи, входящие в данную контекстную роль | Sql запрос, идентификатор sysobject |
| 4 | Java класс, получающий список пользователей | Имя java класса имплементящего интерфейс CustomUserListFinder, идентификатор sysobject |

При использовании алгоритма 4 необходимо создать класс, и имплиментить интерфейс ContextRole.

public interface CustomUserListFinder {

List<Long> find(SysObjectKey relSysobjectKey) throws ServerException;

}

Метод find(SysObjectKey relSysobjectKey) должен возвращать список идентификаторов пользователей.   
События на пересчет контекстной роли необходимы во-первых - для поддержании состава роли в актуальном состоянии при изменении состояния sysobject которые влияют на состав роли. Во вторых для минимизации затрат на вычисления контекстных ролей путем выборочного пересчета ролей для текущего sysobject. По сути конфигурация события определяет алгоритм получения текущего sysobject по sysobject который влияет на текущий. Например у sysobject с типом “Организация” есть контекстная роль “Сотрудники организации”. При добавлении, изменении и удалении сотрудника необходимо пересчитать контекстную роль для организации. При этом в конфигурации контекстной роли должен присутствовать тип на изменение которого надо реагировать сервису контекстных ролей «Сотрудники организации» и запрос как по текущему сотруднику получить организацию, в которой надо пересчитать контекстную роль. Алгоритмы получения sysobject на который надо пересчитать права приведены в таблице 4

Таблица 4

|  |  |  |
| --- | --- | --- |
| **№** | **Описание** | **Информация, необходимая для идентификации события на пересчет роли** |
| 1 | Объект влияет сам на себя | Имя атрибута, при изменении которого необходимо пересчитать роль |
| 2 | Связанный объект влияет на контекстную роль | Имя типа и имя атрибута, ссылающегося непосредственно на sysobject и имя атрибута в связанном sysobject при изменении которого на текущий sysobject пересчитываются права |
| 3 | На контекстную роль влияет sysobject который можно получить запросом | Имя типа и sql запрос с помощью которого получается sysobject, на который надо пересчитать контекстную роль. |
| 4 | На контекстную Роль влияет sysobject, который не возможно получить запросом, в этом случае пишется java класс. | Имя типа и имя java класса, имплементящего интерфейс CustomSysObjectListFinderс помощью которого получается sysobject, на который надо пересчитать контекстную роль. |

При использовании алгоритма 4 необходимо использовать интерфейс описанный ниже:

public interface CustomSysObjectListFinder {

List<SysObjectKey> findSysObjectList(SysObjectKey sysObjectKey) throws ServerException;

}

Принцип работы сервиса формирования контекстных ролей:

1. При создании:
   1. При создании экземпляра sysobject получается конфигурация всех контекстных ролей для этого sysobject. После сохранения запускается метод по актуализации контекстной роли для каждой роли из конфигурации иными словами создаются или изменяются контекстные группы, привязанная к данному sysobject т.е заполняется таблица role и role\_persons. В метод передается созданный sysobject после сохранения.
   2. В конфигурации контекстных ролей в описаниях событий ищутся вхождения данного типа в описании событий на пересчет контекстных ролей для других типов. В случае если данные вхождения найдены с помощью одного из алгоритмов описанных выше находится объект который требует пересчета контекстной роли и вызывается пересчет контекстной роли для найденного sysobject. В метод получения sysobject на который надо пересчитать контекстные группы передается созданный текущий sysobject после сохранения. Далее выполняется пересчет контекстной роли для найденного sysobject.
2. При изменении:
   1. В конфигурации контекстных ролей ищутся вхождения данного типа в описания событий на пересчет контекстных ролей. Причем в методы, реализующие алгоритмы поиска sysobject ка который надо пересчитать права передается текущий изменяемый sysobject сначала до изменения, результат запоминается, а потом после изменения. Если результат разный то пересчет выполняется для sysobject с обоими полученными идентификаторами если одинаковый то только один раз. Пересчет выполняется после сохранения изменений.
3. При удалении:
   1. В конфигурации контекстных ролей ищутся вхождения данного типа в описания событий на пересчет контекстных ролей. Причем в методы, реализующие алгоритмы поиска sysobject передается текущий sysobject до удаления. Далее вызывается метод пересчета состава контекстной группы уже после удаления текущего sysobject.

**Сравнение контекстной роли и динамической группы**

Из статьи видно что правообразующими сущностями в системе являются контекстные роли и динамические группы, причем любую задачу можно решить как с помощью контекстных ролей так и динамических групп. Разделение произведено для оптимизации этого процесса. Приведем сравнительную таблицу этих сущностей:

|  |  |
| --- | --- |
| **Контекстная роль** | **Динамическая группа** |
| Хранится в базе данных | Не хранится в базе данных |
| В состав входят пользователи | В состав входят контекстные роли |

Контекстные роли - это первичная сущность, так как входит в динамическую группу и она хранится в базе, соответственно должна быть уже сформирована на этапе вычисления динамических групп. Контекстные роли необходимо создавать на справочники, такие ка организационная структура, Служебные группы и т.д. Эти операции достаточно редкие, поэтому сюда надо вынести трудно получаемые операции, такие как вычисление цепочек руководства вверх или вниз по иерархии, получение регистраторов, делопроизводителей, архивариусов подразделений или организаций и т.п. Контекстные роли нельзя создавать на документы, поручения, отчеты и другие сущности живущие с документами, так как при этом будет раздуваться база данных и эффективность пропадет.

Динамические группы – это вторичная сущность, группы содержат ссылки на контекстные роли. Группы необходимо создавать для документов, поручений и т.д. Алгоритм получения состава динамических групп надо делать как можно проще, чтобы назначение как можно быстрее выполнялось.

# Настройка прав доступа

**Основы настройки конфигурации прав доступа**

Особенность конфигурирования прав доступа в системе заключается в том, что права всегда выдаются не пользователю а роли. Роли бывают статическими и контекстными. Контекстные роли - это такие роли, когда состав их разный, в зависимости от контекста данной роли. Например контекстная роль "руководитель организации", в зависимости от организации состав данной роли будет разный. Соответственно в настройке прав можно выделить два больших раздела: настройка формированяи контекстных ролей; настройка прав для данных контекстных ролей для определенного типа карточек (Динамические группы).

**Контекстные роли**

Рекомендуется формировать контектные роли для редко изменяющихся объектов, и в то же время когда данные объекты могут влиять на права большого числа карточек. Во первых это все что касается организационно-штатной структуры, так же это касается объектов, влияющих на состав рабочих и функциональных групп. Конфигурация контекстных групп расположена внутри конфигурации типа карточки. Пример:

<config name="test\_employee">

<core:sysObjectType>

<core:attributes>

<core:attribute name="person\_id" type="id" required="true" relationType="person" />

</core:attributes>

<core:contextRoles>

<core:contextRole name="Я и мои делегаты" description="Я и мои делегаты">

<core:event type="test\_employee">

<core:attributes>

<core:attribute name="person\_id" />

</core:attributes>

<core:ownerGetter>

<core:link>.</core:link>

</core:ownerGetter>

</core:event>

<core:entries>

<core:users>

<core:link>person\_id</core:link>

</core:users>

</core:entries>

</core:contextRole>

</core:contextRoles>

</core:sysObjectType>

</config>

Конфигурация расположена внутри тега "contextRoles". В одном типе может быть описано множество контекстных ролей, все они будут пересчитаны и сохранены в хранилище данных. В атрибуте name тэга contextRole указывается имя контекстной роли. Это уникальный идентификатор. При настройке прав при ссылки на контекстную роль указывается именно это значение.

**События пересчета контекстной роли**

В тэге event указываются события, которые делают состав роли невалидным и приводят к пересчету ее состава. В теге указывается имя типа (@type) и имена атрибутов (attributes/attribute@attribute) при изменении которых роль становится невалидной. Событий может быть много. В каждом событие есть настройка того, как найти объект, которому принадлежит изменяемая роль, зная какой объект изменился. Данная настройка расположена в тэге ownerGetter. Например состав роли с именем "Руководители сотрудника" могут зависеть от поля "Руководитель" в карточке "Организация". В данном теге будет описано правило как найти сотрудников, у которых необходимо пересчитать роль "Руководители сотрудника". Поддерживаются несколько способов поиска карточек, у которых следует пересчитать роль:

* С помощью ссылки. В этом случае внутри тэга ownerGetter будет тэг link. Текст внутри тэга будет означать ссылку на искомые карточки на [Язык ссылок](#_Язык_ссылок). Пример

<core:ownerGetter>

<core:link>organization\_id</core:link>

</core:ownerGetter>

* С помощью запроса. В этом случае внутри тэга ownerGetter будет тэг query. Текст внутри тэга будет интерпретироватся как запрос на внутреннемм SQL языке. Запрос должен вернуть таблицу, содержащую два поля. Первое поле считается идентификатором найденной карточки, второрое - типом. Внутри запроса можно использовать специальные символы, обозначающие информацию из базовой карточки: (:currentId) - идентификатор текущей карточки;(:currentType) - тип текущей карточки. Пример:

<core:ownerGetter>

<core:query>select id, type from test\_employee e where department\_id=(:currentId)</core:query>

</core:ownerGetter>

* С помощью класса. Класс должен имплементить интерфейс ru.intertrust.cm5.core.CustomSysObjectListFinder. Пример:

<core:ownerGetter>

<core:className>ru.intertrust.cm5.core.test.contextrole.ManagementChain</core:className>

</core:ownerGetter>

**Состав контекстной роли**

В теге entries указывается состав роли. Роль может состоять из пользователей или контекстных ролей других объектов. В случае когда необходимо указать в качестве состава пользователей, внутри тэга entries распологается тэг users, в котором описываются правила получения спсика пользователей, относительно текущего объекта, на который назначаются права. Пользователи могут быть получены тремя разными способами:

* С помощью ссылки. В этом случае внутри тэга users будет тэг link. Текст внутри тэга будет означать ссылку на искомые карточки персон на [Язык ссылок](file:///D:\CM5\br4jv2\@title=_25d0_25af_25d0_25b7_25d1_258b_25d0_25ba__25d1_2581_25d1_2581_25d1_258b_25d0_25bb_25d0_25be_25d0_25ba.htm). Пример

<core:users>

<core:link>employee\_id</core:link>

</core:users>

* С помощью запроса. В этом случае внутри тэга users будет тэг query. Текст внутри тэга будет интерпретироватся как запрос на внутреннемм SQL языке. Запрос должен вернуть таблицу, содержащую два поля. Первое поле считается идентификатором найденной карточки персоны, второрое - типом. Внутри запроса можно использовать специальные символы, обозначающие информацию из базовой карточки: (:currentId) - идентификатор текущей карточки;(:currentType) - тип текущей карточки. Пример:

<core:users>

<core:query>select id, type from test\_employee e where department\_id=(:currentId)</core:query>

</core:users>

* С помощью класса. Класс должен имплементить интерфейс ru.intertrust.cm5.core.CustomUserListFinder. Пример:

<core:users>

<core:className>ru.intertrust.cm5.core.test.contextrole.ManagementChain</core:className>

</core:users>

Второй способ получения состава контекстной роли - это включение в роль контекстных ролей других карточек. В этом случае в тэге entries распологается тэг contextRoles. У тэга contextRoles необходимо заполнить атрибут roleName, который показывает какую роль необходимо включить в вычисляемую роль. Внутри тэга contextRoles размещена конфигурация, показывающая как найти карточки, у которых возьмутся роли для включения в текущую, вычисляемую роль. Карточки можно получить тремя разными способами:

* С помощью ссылки. В этом случае внутри тэга contextRoles будет тэг link. Текст внутри тэга будет означать ссылку на искомые карточки на [Язык ссылок](file:///D:\CM5\br4jv2\@title=_25d0_25af_25d0_25b7_25d1_258b_25d0_25ba__25d1_2581_25d1_2581_25d1_258b_25d0_25bb_25d0_25be_25d0_25ba.htm). Пример:

<core:contextRoles roleName="DELEGATES">

<core:link>employee\_id</core:link>

</core:contextRoles>

* С помощью запроса. В этом случае внутри тэга users будет тэг query. Текст внутри тэга будет интерпретироватся как запрос на внутреннемм SQL языке. Запрос должен вернуть таблицу, содержащую два поля. Первое поле считается идентификатором найденной карточки, второрое - типом. Внутри запроса можно использовать специальные символы, обозначающие информацию из базовой карточки: (:currentId) - идентификатор текущей карточки;(:currentType) - тип текущей карточки. Пример:

<core:contextRoles roleName="DELEGATES">

<core:query>select id, type from test\_employee e where department\_id=(:currentId)</core:query>

</core:contextRoles>

* С помощью класса. Класс должен имплементить интерфейс ru.intertrust.cm5.core.CustomUserListFinder. Пример:

<core:contextRoles roleName="DELEGATES">

<core:className>ru.intertrust.cm5.core.test.contextrole.ManagementChain</core:className>

</core:contextRoles>

В теге entries может быть несколько тэгов contextRoles и users.

**Настройка правил заполнения ACL**

**Костяк кофигурации**

<config name="acl\_name">

<core:aclConfig>

<core:dynamicGroups>

...

</core:dynamicGroups>

<core:rules>

...

</core:rules>

</core:aclConfig>

</config>

В конфигурации есть два основных блока:

1. <core:dynamicGroups> - описание состава динамических групп и событий, во время которых должна обновляться динамическая группа
2. <core:rules> - матрица прав доступа

Для привзяки конфигурации к конкретному типу, в конфигурации типов появился атрибут acl.

**Динамические группы**

Динамическая группа - это логическое объединение нескольких контекстных групп, которым даются идентичные права.

**Описание состава динамических групп и событий, во время которых должна обновляться динамическая группа**

<core:dynamicGroups>

<core:dynamicGroup name="Исполнители резолюций">

<core:entries>

...

</core:entries>

<core:events>

...

</core:events>

</core:dynamicGroup name="Исполнители других резолюций">

<core:dynamicGroup name="Исполнители других резолюций">

...

</core:dynamicGroup name="Исполнители других резолюций">

</core:dynamicGroups>

Описание групп представляет собой список групп, каждая из которых делится также на два блока:

1. <core:entries> - состав динамической группы
2. <core:events> - события, во время которых должна обновляться динамическая группа

**Состав динамической группы**

<core:entries>

<core:contextRolesReader>

<core:roles>

<core:role>Я и мои делегаты</core:role>

</core:roles>

<core:sysObjectListReader>

<core:link>author</core:link>

</core:sysObjectListReader>

</core:contextRolesReader>

...

</core:entries>

Тег <core:entries> состоит из тегов <core:contextRolesReader>, каждый из которых представляет собой, описание получения контекстных ролей

<core:contextRolesReader> включает в себя два тега:

1. <core:roles> - список имен контекстных ролей
2. <core:sysObjectListReader> - список объектов, у которых будут искаться роли указанные в теге <core:roles>

**Способы получения списка объектов**

Есть три способа:

1. Запрос
2. Указать имя класса реализующего интерфейс CustomSysObjectListFinder
3. Указать ссылку - ссылка облегченный синтаксис для указания получения списка по какому-либо атрибуту

В системе имеется класс реализующий интерфейс CustomSysObjectListFinder, для получения списка объектов в котором будет только объект для которого сработало событие - ru.intertrust.cm5.core.workflow.CurrentSysObject(пример использования: Тип "Входящие", группа "Автор документа", событие срабатывает для типа "Входящие", при изменение атрибута "автор", тогда в качестве способа получения списка объектов можно указать данный класс).

**События, во время которых должна обновляться динамическая группа**

Событие представляет собой изменение какого-либо атрибута в объекте(изменение включает в себя удаление, создание и изменение атрибута)

<core:events>

<core:event sysObjectType="any\_type">

<core:attributes>

<core:attribute>attr1</core:attribute>

<core:attribute>attr2</core:attribute>

</core:attributes>

<core:sysObjectListReader>

....

</core:sysObjectListReader>

</core:event>

<core:event>

...

</core:event>

</core:events>

Данный блок представяет собой список событий. Описание событие состоит из двух блоков, а также включает атрибут sysObjectType - тип объекта, для которого срабатывает данное:

1. <core:attributes> - список атрибутов(из типа который указан в атрибуте sysObjectType), при изменении которых совершится данное событие.
2. <core:sysObjectListReader> - способ получения списка объектов, у которых должна обновится данная динамическая группа(например: событие у группы "Автор документа основания" которая есть у типа "Резолюция" срабатывает для типа "Входящие", при изменении атрибута "автор", то в этом блоке мы должны указать как из данного входящего документа получить список резолюций.

Cпособы получения списка объектов были описаны ранее в описании состава динамической группы

**Матрица прав доступа**

<core:rules>

<core:rule groupName="Автор резолюции" status="1" value="D" />

<core:rule groupName="Автор резолюции" status="2" value="R" />

...

</core:rules>

Данный блок представляет собой список правил. Каждое правило представляет собой:

1. Имя группы
2. Статус
3. Уровень доступа(R - Read, W - Write, D - Delete)

**Пример готовой конфигурации**

Полную кофигурацию можно посмотреть в Core-test testPermission.xml, и проверить её работоспособность с помощью запуска теста TestPermission

<config name="test\_indoc\_acl">

<core:aclConfig>

<core:dynamicGroups>

<core:dynamicGroup name="Автор документа">

<core:entries>

<core:contextRolesReader>

<core:roles>

<core:role>Я и мои делегаты</core:role>

</core:roles>

<core:sysObjectListReader>

<core:link>author</core:link>

</core:sysObjectListReader>

</core:contextRolesReader>

</core:entries>

<core:events>

<core:event sysObjectType="test\_indoc">

<core:attributes>

<core:attribute>author</core:attribute>

</core:attributes>

<core:sysObjectListReader>

<core:className>ru.intertrust.cm5.core.workflow.CurrentSysObject</core:className>

</core:sysObjectListReader>

</core:event>

</core:events>

</core:dynamicGroup>

<core:dynamicGroup name="Исполнители резолюций">

<core:entries>

<core:contextRolesReader>

<core:roles>

<core:role>Я и мои делегаты</core:role>

</core:roles>

<core:sysObjectListReader>

<core:query>SELECT emp.id, emp.type

FROM test\_indoc ind

JOIN test\_resolution res ON (ind.id = res.document\_id)

JOIN test\_employee emp ON (res.executor = emp.id) WHERE ind.id=(:currentId)

</core:query>

</core:sysObjectListReader>

</core:contextRolesReader>

</core:entries>

<core:events>

<core:event sysObjectType="test\_resolution">

<core:attributes>

<core:attribute>executor</core:attribute>

<core:attribute>document\_id</core:attribute>

</core:attributes>

<core:sysObjectListReader>

<core:query>SELECT ind.id, ind.type

FROM test\_indoc ind

JOIN test\_resolution res ON (ind.id = res.document\_id)

WHERE res.id=(:currentId)

</core:query>

</core:sysObjectListReader>

</core:event>

</core:events>

</core:dynamicGroup>

</core:dynamicGroups>

<core:rules>

<core:rule groupName="Автор документа" status="1" value="D" />

<core:rule groupName="Автор документа" status="2" value="W" />

<core:rule groupName="Автор документа" status="3" value="R" />

<core:rule groupName="Исполнители резолюций" status="2" value="R" />

<core:rule groupName="Исполнители резолюций" status="3" value="R" />

</core:rules>

</core:aclConfig>

</config>

**Права на создание карточек**

По умолчанию карточки разрешено создавать только пользователям имеющим роль ADMIN. Если необходимо разрешить другим ролям создавать карточки того или иного типа то необходимо в конфигурацию типа добавить тэг cratePermissions, внутри которого в дочернем тэге role добавить те статичные роли, которым надо дать права на создание карточек данного типа. Пример:

<core:cratePermissions>

<core:role>TEST\_ROLE</core:role>

</core:cratePermissions>

**Права на связи**

По умолчанию, на карточку любого типа, любой пользователь может создать ссылку в атрибуте типа ID из другой карточки, при условии что в атрибуте ID в конфигурации в relationType прописан корректный тип. Но можно разрешить ссылатся не всем, а только конкретным контекстным ролям той карточки, на которую создается ссылка. Для этого в настройках типа необходимо указать атрибут referenceByAll и прописать значение ему false.

<config name="indoc">

<core:sysObjectType acl="test\_indoc\_acl" referenceByAll="false">

<core:attributes>

...

</core:attributes>

</core:sysObjectType>

</config>

При этом на карточки данного типа возможно будет создать связь только тем пользователям, которым разрешено это делать в настройках acl. Настройка производится в конфигурации acl в теге rule. Настройка производится отдельно на создание связи и на разрыв связи. Пример:

<core:rule groupName="AUTHOR" status="DRAFT" value="D">

<core:relationPermissions>

<core:relationPermission type="resolution" attribute="document\_id" canCreate="true" canDelete="true" />

</core:relationPermissions>

</core:rule>

Назначение атрибутов relationPermission:

* type - тип карточки, который ссылается на текущую;
* attribute - имя атрибута в ссылающейся карточку, в который записываеься идентификатор текущей карточки;
* canCreate - разрешает или запрещает создавать связь;
* canDelete - разрешает или запрещает разрывать связь.

Пользователи имеющие роль ADMIN могут устанавливать и разрывать любые связи, не зависимо от данной настройки.

**Права на атрибуты**

По умолчанию все атрибуты карточек видны всем пользователям, которые имеют право видеть данный документ, и если пользователь имеет право править документ то атрибуты доступны на редактирование. Возможно запредить править некоторые атрибуты или скрыть их для определенных динамических групп. Для этого внутри тэга rule настройки acl необходимо добавить тэг attributePermissions. Пример

<core:rule groupName="AUTHOR" status="DRAFT" value="D">

<core:attributePermissions>

<core:attributePermission name="name" hidden="false" readOnly="true"/>

</core:attributePermissions>

</core:rule>

Назначение атрибутов attributePermission:

* name - имя атрибута;
* hidden - атрибут скрытый, по умолчанию false;
* readOnly - атрибут доступен только на чтение, по умолчанию fals.;

Если пользователь входит в несколько динамических групп, то действует максимальное разрешающее права, т.е. если одна гроуппа не видит атрибут а другая видит, то пользователь будет видеть данный атрибут.

**Права на бизнес действия**

По умолчани права на бизнес действия разрешены пользователям имеющим роль ADMIN. Для того чтобы предоставить права на бизнес действия другим пользователям необходимо в тэг rule добавить тэг actionPermissions. Пример:

<core:rule groupName="AUTHOR" status="DRAFT" value="D">

<core:attributePermissions>

<core:actionPermission action="test\_action" />

</core:attributePermissions>

</core:rule>

Назначение атрибутов actionPermission:

* action - имя действия, на которое предоставляется право.

**Права на действия по процессам**

По умолчани права на действия по процессам разрешены пользователям имеющим роль ADMIN. Для того чтобы предоставить права на действия по процессам другим пользователям необходимо в тэг rule добавить тэг flowPermissions. Пример:

<core:rule groupName="AUTHOR" status="DRAFT" value="D">

<core:flowPermissions>

<core:flowPermission process="testPermission" flow="flowCancal" />

<core:flowPermission process="testPermission" flow="flowNegotiate" />

</core:flowPermissions>

</core:rule>

Назначение атрибутов flowPermissions:

* process - имя процесса, на переходы которого предоставляется право;
* flow - имя перехода процесса, на которорый предоставляется право.

**Права на установку статусов**

По умолчани права на установку статусов разрешены пользователям имеющим роль ADMIN. Для того чтобы предоставить права на установку статусов другим пользователям необходимо в тэг rule добавить тэг flowPermissions. Пример:

<core:rule groupName="AUTHOR" status="DRAFT" value="D">

<core:statusPermissions>

<core:statusPermission status="InArchive" />

</core:statusPermissions>

</core:rule>

Назначение атрибутов statusPermissions:

* status - имя статуса, в который выполняется переход.

# Модификация настройки прав

В процессе настройки прав вылезла проблема, о которой как то раньше не подумали. Напомню как описываются права сейчас. Создается матрица в строках которой указаны динамические группы, в столбцах – статусы карточки.

Пример:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Роль\Статус** | **Подготовка** | **Согласование** | **Подписание** | **Регистрация** | **Зарегистрирован** | **В дело** |
| Регистратор |  |  |  | W | R | R |
| Автор | D | R | R | R | R | R |
| Ответственный за ДОУ |  |  | R | R | R | R |
| Подписывающие |  |  | R | R | R | R |
| Ознакамливающиеся |  |  | R | R | R | R |

Проблема в том, что нет возможности описать права, если они зависят от статусов связанного объекта, а статус самого объекта при этом не меняется. Это нужно , например при назначение прав на вложение. Например такая задача: права W должен иметь автор документа основания, когда документ в статусе черновик или на доработке. Текущий механизм не покрывает такие требования. Обсудив проблему нарисовалось три варианта решения данной проблемы:

1. При смене статуса документа основания отрабатывают точки расширения, которые переводят вложения в те же статусы что и родительского документа. В матрице настраивается все так же как и раньше, так как у нас статусы меняются синхронно. Достоинства: не меняется механизм расчета прав. Недостатки: для таких случаев необходимо программирование; Логика назначения прав расплывается по конфигурации и коду; Возможно появятся лишние пересчеты, когда они в принципе ненужны. Наименее привлекательный метод, так как требует программирования.
2. Изменяется смысл заголовка столбцов в матрице прав. Это не статус объекта на который назначаются права, а результат работы некоторой функции. Например функция будет возвращать статус связанного объекта, или значение атрибута текущего объекта или связанного. Далее при расчете прав вызывается данная функция и из матрицы берется столбец, который соответствует результату работы данной функции. Достоинства: конфигурация прав сосредоточена в одном месте; добавляется гибкость настройки прав, которая поможет нам при реализации «извращенных» хотелок заказчика. Недостатки: меняется механизм расчета прав, он усложняется. Наиболее интересный и в то же время трудоемкий метод. Сложность заключается в определение момента, когда меняется объект, который влияет на вычисляемый статус.
3. Мы с помощью текущего механизма формируем динамические группы, которые меняют свой состав при изменение статуса связанного(ых) объекта(ов). Например создаем группы «сотрудники рассматривающие документ в текущий момент», автор документа в составе черновик. Достоинства: не меняется текущий механизм. Недостатки: на мой взгляд искажается смысл «группы» раньше это было множество людей, описываемое определенными правилами. А сейчас это множество меняется со временем, и в какой то смысле появляется третье измерение в нашей матрице. При сложных правах количество таких групп будет сильно увеличиватся. Пока останавливаемся на данном варианте, так ка он позволяет достич результата наименьшими трудозатратами.

# Недостатки новой системы прав доступа

Недостатки:

1. Длительный пересчет прав для одного документа от 5 сек.
2. Ограничение настроек. Можно создать правило получения персоны только по двум связям
3. Не используются контекстные группы, которыми можно было бы облегчить пересчет прав в случае изменения оргштатки
4. Избыточность информации. При добавление новых прав, дублируется информация в таблице описания правила
5. Механизм не обеспечивает все потребности в правах, например все сотрудники организации или подразделения
6. Права пересчитываются сразу, даже если этого и не нужно.

Предложения:

1. Вызов пересчета прав выполнять не сразу, а при необходимости. Решение принимает разработчик.
2. Ввести понятие динамической роли и контекстной группы. Привязывать ее к объектам системы, которые могут влиять на права. Например все элементы оргштатки
3. Отказатся от использования profile\_access\_rule
4. Отказатся от access\_attr\_rule.
5. Расширить возможности механизма прав, отказавшись от ограничения 3 карточек. Самым уневерсальным способом является класс, который будет формировать список персон, или контекстных групп, относящихся к той или иной роли. Данный класс будет указан в настройках правила. Класс должен будет расширять какой либо стандартный интерфейс.

Описание предлагаемого механизма контекстных групп: Сейчас в случае изменения отгштатной структуры, или появление или удаление делегатов, изменению подлежат все ACL документов, на которые должны появится права согласно новому штатному рассписанию или новому делегату. В случае с пересчетом прав в течение 5 сек, и наличия 100000 документов на которые должны пересчитатся права то это 500000 сек или 5,7 дней. Предлагаю использовать механизм контекстных групп примерно в следующем виде:

1. Выделяются объекты системы, которой могут быть выданы те или иные права, например персона, подразделение, организация, функциональная группа сотрудников и т.д.
2. Для каждого такого объекта создается контекстная группа system\_role, и с помощью специальной таблицы привязывается к данному объекту. В данную группу попадают персоны, логически связанные с этой группой. Например контекстная группа "Пользователи организации" содержит всех пользователей данной организации. Группа "Руководители подразделения" содержит персоны руководителя(лей) подразделения.
3. Создается процессоры, или иные программные модули, которые поодерживают состав контекстной группы в актуальном виде.
4. Когда система пересчитывает права на документ в его ACL попадает не персоны, а те или иные контекстные группы.

Теперь в случае изменение оргштатки, например смена руководителя подразделения, достаточно изменить контекстную группу "Руководитель подразделения", а ACL при этом остается неизменным, так как во всех ACL использовалась именно группа "Руководитель подразделения" а не конкретная персона. Аналогично, в случае появления делегата, или помошника, требуется дать права этому пользователю на все ранее созданные документы. Если мы создадим контекстную группу для каждого пользователя, и добавим в нее самого пользователя и его делегата, а в ACL пропишем эту контекстную группу то данная операция сведется к изменению контекстной группы, и не потребует изменения ACL документов.

# Бизнес-процессы в системе

**Требования к подсистеме бизнес-процессов**

Для работы системы документооборота необходим движок Workflow или BPM. Есть свободно распространяемы, бесплатные движки Workflow. Для того чтобы понять можно ли использовать данные движки или придется написать свой определимся с требованиями к данному движку:

* Шаблон должен состоять из активностей и переходов между ними, с возможностью выполнения условных переходов, и переходов на уже пройденные ранее активности (цикл);
* Наличие понятия шаблона процесса и экземпляра процесса;
* Возможность создания нескольких версий шаблона процесса при одновременной работе разных версий процесса;
* Описание шаблона процесса с помощью xml файла или иного текстового формата;
* Возможность производить мониторинг процессов с помощью готового интерфейса, или наличие API для этого мониторинга (запуск, приостановка, возобновление, уничтожение экземпляров процессов);
* Наличие ручных и автоматических активностей;
* Интеграция в пространство пользователей системы;
* Визуальный графический дизайнер процессов;

Сразу хочу высказать свое мнение по поводу использование для данных целей движка, реализующего язык BPEL. Данный язык, наверно, один из самых распространенных спецификаций для реализации бизнес процессов, и он удовлетворяет практически всем требованиям, перечисленным выше, но опыт работы с ним выявил следующие его недостатки:

* Для того чтобы процесс мог что то делать в системе документооборота, данная система должна иметь web-service API;
* Пространство пользователей системы и движка достаточно сложно объединить;
* Сложность написания шаблонов, даже имея визуальный дизайнер. Разработчик в совершенстве должен знать спецификации SOAP, WSDL.
* В шаблоне не используется понятия переходов, то есть находясь в определенной активности нельзя получить все возможные переходы в следующие активности. Модель больше похожа на блок схему, что, при наличие большого числа ветвлений, делает ее нечитаемой. Приведу пример работающего шаблона, реализующего процесс согласования:   
  . . .

По итогам работы с процессами на BPEL сделан следующий вывод. Данный язык имеет очень широкие возможности но ориентирован прежде всего на взаимодействие различных систем в масштабе организации. Для использование его в системе документооборота язык не будет использоваться в полном объеме, а его универсализация приводит к сложности написания шаблонов. К тому же в системе документооборота 95% действий, совершаемых из активностей процесса изменяют документы в той же самой системе документооборота, что сводит на нет достоинства использования web сервисов. Напротив web сервизы замедляют работу процессов в целом и делают достаточно сложным разработку шаблона. Выше описанные недостатки позволяют мне не рассматривать движки работающие по спецификации BPEL.

На мой взгляд необходимо рассматривать движки Workflow шаблон которых представляет собой граф переходов между активностями. Активности должны иметь возможность выполнять любые действия в системе документооборота, используя его API. Рассмотрим такую систему.

**JBOSS jbpm**

У JBOSS есть свой движок workflow. Поверхностное его изучение показало что его вполне можно использовать в системе документооборота. Данный движок ориентирован на взаимодействие с пользователем, в отличие от BPEL. Шаблон процесса выглядит в виде графа переходов между активностями, поддерживается ветвление, условные переходы, циклы. Имеется плагин для eclipsa для создания, редактирования и отладки процесса. Существует API для интеграции данного движка в любую систему, есть интерфейс для мониторинга процессов. Шаблон процесса соответствует стандарту BPMN 2.0 что гарантирует переносимость шаблонов на другие движки бизнес процессов, реализующие данный стандарт. Так как данный движок функционирует на платформе jboss легко производится интеграция пользователей, путем выбора одного домена безопасности. Существует широкий набор простых активностей, которые позволяют реализовать любую логику работы системы, вместе с тем предоставляется возможность создать сколько угодно сложную активность, выполняющую специфические действия. Например смена статуса документа, рассылка уведомлений на электронную почту и.т.д.

**Вывод**

На мой взгляд jboss jbpm как нельзя лучше подходит к реализации процессов в системе в связи со своей гибкостью и легкостью интеграции в систему.

# Activiti

**Хранение истории процессов**

Таблицы Activiti для хранение сведений о процессах:

* act\_hi\_actinst – история исполнения активностей процессов;
* act\_hi\_attachment – история прикрепленный вложений (*в нашем проекте вложения процессов пока не используется*);
* act\_hi\_comment – история сообщений и комментариев к процессам;
* act\_hi\_detail – информация о переменных процессов;
* act\_hi\_procinst – информация о выполнении экземпляров процессов;
* act\_hi\_taskinst – информация о выполнении задач для пользователей (User Task).

**Настройка движка Actitviti для хранения истории:**

В файле **beans.xml** в bean processEngineConfiguration добавить

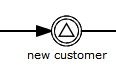
<property name="history" value="audit" />

где value – уровень хранения информации. Классификация уровней:

* **none :** пропускается вся история. Это самый производительный уровень для выполнения процессов, но не будет информации о процессах.
* **activity :** хранится история только о выполнении экземпляров процессов и активностей.
* **audit :** (значение по умолчанию) хранится история о выполнении экземпляров процессов и активностей, настройки форм и взаимодействие пользователей с формами (*в нашем проекте формы Activiti пока не используются*).
* **full :** сохраняется полностью вся история – информация о выполнении экземпляров процессов, активностей, изменения переменных, комментарии. (*данный уровень указан в текущей конфигурации Activiti в проекте*)

Работа с историей осуществляется через внутренний сервис Activiti – **HistoryService**. Основные методы: createHistoricProcessInstanceQuery(), createHistoricActivityInstanceQuery(), createHistoricDetailQuery() и createHistoricTaskInstanceQuery().

**Построение взаимодействия несвязанных процессов**

Взаимодействие процессов строится на событиях и сигналах. В Activiti за это отвечают элемент «Signal Intermediate catching event» [](file:///D:\CM5\br4jv2\@title=_25d0_25a4_25d0_25b0_25d0_25b9_25d0_25bb_3asignal.catch.event.png), которое создает событие ожидающие сигнал, и элемент «Signal Intermediate throwing event» [](file:///D:\CM5\br4jv2\@title=_25d0_25a4_25d0_25b0_25d0_25b9_25d0_25bb_3asignal.throw.event.png), которое посылает сигнал.

**Принцип работы**

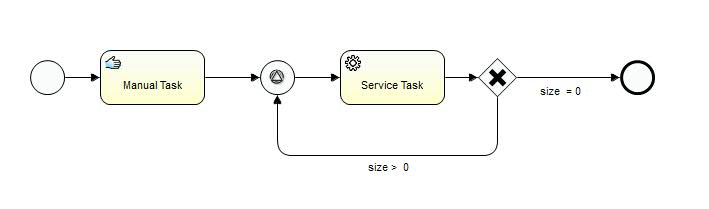
Есть два разных процесса, которые запустили независимо друг от друга. Через некоторое время первый процесс дошел до элемента «Signal Intermediate catching event». Здесь этот процесс остановился, создал событие (запись в таблице Activiti **act\_ru\_event\_subscr**), которое будет ожидать сигнал с именем «**signal**». В это время второй процесс дошел до элемента «Signal Intermediate throwing event», здесь происходит посылка сигнала с именем «**signal**». Второй процесс продолжает работу. Первый получает сигнал, завершает событие ожидания и так же продолжает работу.

При этом сигнал могут получить сразу несколько процессов, т.е. несколько процессов остановились, какой-то другой один процесс послал сигнал, и те процессы сразу все продолжали работу, что является достоинством и недостатком одновременно. Имя сигнала заносится в шаблон процесса и, следовательно, несколько экземпляров процесса будут иметь один и тот же сигнал.

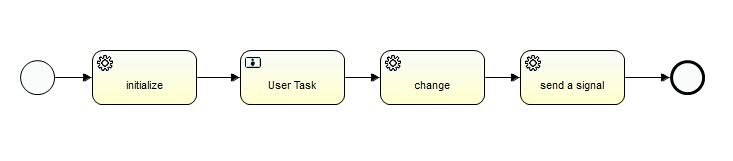
Есть возможность послать сигнал с использованием API Activiti (в том числе и одному определенному процессу). Для посылки сигнала используются методы signalEventReceived(String signalName) или signalEventReceived(String signalName, String executionId) у сервиса Activiti **RuntimeService**, где signalName – имя сигнала, executionId – Id исполнения активности (не путать с Id экземпляра процесса). В шаблоне процесса в том месте, где надо послать сигнал вместо элемента «Signal Intermediate throwing event» используется «Java Service Task», который вызывает класс с функцией посылки сигнала. После работы класса, оба процесса(получающий и посылающий сигнал) продолжат работ.

**Пример использования посылки сигнала определенному процессу**

Есть два шаблона процессов: движение документа и исполнение поручений по документу. Запускается экземпляр движения документа с переменной idSysObject (Id документа). В определенной активности создаются экземпляры не дочерних процессов исполнения поручений, которые имеют так же переменную idSysObject. Далее экземпляр движения документа доходит до того места, где необходимо дождаться завершения экземпляров выполнения поручений. Особенность в том, что во время выполнения поручения могут создаваться другие поручения (но они будут иметь переменную idSysObject) и конечное их число не известно. Для такого взаимодействия используется связка элементов «Signal Intermediate catching event» - «Java Service Task». Часть шаблона движение документа:

[](file:///D:\CM5\br4jv2\@title=_25d0_25a4_25d0_25b0_25d0_25b9_25d0_25bb_3atestprocessprimary.jpg)

Часть шаблона выполнения поручения:

[](file:///D:\CM5\br4jv2\@title=_25d0_25a4_25d0_25b0_25d0_25b9_25d0_25bb_3atestprocesssecondary.jpg)

Дополнительно в шаблоне поручения необходимо вести переменную «completed», которая показывает завершен или нет процесс.

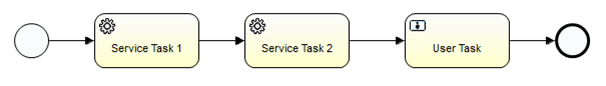
Процесс движения документа доходит до создания события ожидания сигнала «**signal**» и останавливается. Один из процессов выполнения поручения подходит к концу и становится завершенным - в активности «change» переменная completed = true, а в активности «send a signal» посылается сигнал только процессу движения с определенной переменной idSysObject. Процесс движения ловит сигнал, далее в активности «Service Task» подсчитывается количество незавершенных процессов выполнения поручения с определенной переменной idSysObject. Если есть незавершенные, тогда процесс возвращается в ожидание, если нет – то продолжает движение.

*Использование посылки сигнала определенному процессу обусловлено тем, что при посылке просто сигнала будут срабатывать все экземпляры процесса движения документа, что влияет на производительность.*

**Особенности выполнение активностей в синхронном и асинхронном режиме**

**Принцип работы**

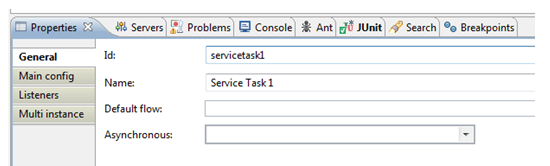
Описание будет построенное на примере простого процесса, схема показана на рисунке:

[](file:///D:\CM5\br4jv2\@title=_25d0_25a4_25d0_25b0_25d0_25b9_25d0_25bb_3asimplepro.png)

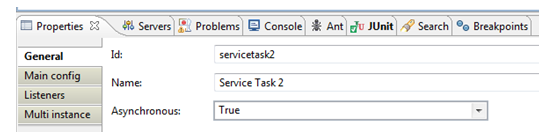
Процесс выполняет простые действия:

После запуска, в активности Service Task 1 происходит вызов класса ClassSerTask1, в котором происходит создание SysObject’а. Далее в Service Task 2 происходит изменение только что созданного объекта при помощи класса ClassSerTask2, после чего в User Task генерируется задача для пользователя. После выполнения задачи, процесс успешно завершается.

У некоторых активностей есть свойство «Asynchronous» (по умолчанию пустое), которое отвечает за синхронность/асинхронность выполнения.

[](file:///D:\CM5\br4jv2\@title=_25d0_25a4_25d0_25b0_25d0_25b9_25d0_25bb_3aprop1.png)

В синхронном режиме активность выполняется в том же потоке, что и инстанс процесса. В случае возникновения ошибки (например, при выполнении действий в классе ClassSerTask2) инстанс процесса полностью останавливается, т.е. процесс не остается в БД. Также происходит откат транзакций, например, отменяется создание объекта в ClassSerTask1 и его изменение в ClassSerTask2. Так же при потери связи с БД в синхронном режиме во время выполнения инстанса процесса этот инстанс не сохраняется, т.е. **процесс пропадает**.

[](file:///D:\CM5\br4jv2\@title=_25d0_25a4_25d0_25b0_25d0_25b9_25d0_25bb_3aprop2.png)

В асинхронном режиме активность выполняется в новом потоке. При этом создается специальная точка остановки (объект Job в Activiti), к которой может произойти откат в случае ошибки в новом потоке. Например, в классе ClassSerTask2 были изменения SysObject’а, возникла ошибка, при этом инстанс процесса не пропадает, а откатывается к моменту исполнения Service Task 2. Специальными средствами Activiti можно повторно запустить выполнение этой активности. При этом все изменения SysObject’а в Service Task 1 остаются, изменения в Service Task 2 отменяются. В случае потери связи с БД, инстанс **процесса не пропадает** и откатывается к моменту исполнения Service Task 2.

Стоит отметить, что можно настроить обработку исключения в ClassSerTask2 таким образом, что откат в процессе не происходит и запускается активность User Task.

В ходе исполнения активности (Asynchronous=true) создается новая транзакция, в которой выполняются действия внутри класса(например, изменения SysObject’a), а так же системные действия активности (например, сохранение истории инстанса процесса). При возникновении ошибки (например, неудачная запись об истории процесса) транзакция откатывается и SysObject не меняется.

**Замечания**

1. Существует возможность изменять параметры движка Activiti без использования beans.xml с помощью вызова функция у processEngine (processEngineImpl), что дает возможность получать параметры из главной/серверной конфигурации (например, временной период обращения движка к БД).
2. Для работы Asynchronous=true у активностей необходимо чтобы JobExecutorActivate был включен (true).

**Дополнительные настройки движка Activiti**

**Активация работы Job’ов без использования beans.xml**

@Autowired

private processEngine processEngine;

//...

ProcessEngineImpl processEngineImpl = (ProcessEngineImpl) processEngine;

ProcessEngineConfigurationImpl conf = processEngineImpl.getProcessEngineConfiguration();

conf.setJobExecutorActivate(true);

conf.buildProcessEngine();

//...

**Установка таймера проверки Job’ов**

@Autowired

private ProcessEngine processEngine;

//...

ProcessEngineImpl processEngineImpl = (ProcessEngineImpl) processEngine;

ProcessEngineConfigurationImpl conf = processEngineImpl.getProcessEngineConfiguration();

conf.getJobExecutor().setWaitTimeInMillis(60000);

//...

где 60000 – период обращения к БД в миллисекундах.

# ToDo

Данная страница содержит весьма неупорядоченный список высокоуровневых и низкоуровневых требований к системе, о которых важно не забыть.

**Утилиты**

В рамках разработки системы планируется разработка следующих утилитарных систем, целью которых является упрощение настройки и модернизации системы интеграторами и администраторами системы.

**Инсталлятор**

При помощи инсталлятора производится установка системы и ее первоначальная настройка. Инсталлятор работает в двух режимах - командной строки и графического интерфейса. Инсталлятор производит все необходимые действия по настройке СУБД и сервера приложений в зависимости от выбранной конфигурации, избавляя интегратора/администратора от необходимости запуска скриптов для БД и правки XML-настроек сервера приложений.

**Модификатор системы**

Модификатор системы может являться под-системой Инсталлятора. Его целью является автоматическое осуществление модификации системы при выпуске патчей или новых версий. Инсталлятор и модификатор, в случае поддержки различных серверов приложений или СУБД, могут выполнять различные действия в зависимости от версии серверов, но это остается прозрачным для конечного пользователя утилиты.

**Конфигураторы системы**

Конфигурация системы (пользовательские формы, списки и прочее) описывается XML-документами. Для каждой конфигурации должен присутствовать редактор с удобным и понятным графическим интерфейсом пользователя, позволяющий в интерактивном режиме создавать или редактировать существующие конфигурации. Конфигураторы могут быть представлены как в виде самостоятельных, не обязательно веб-, приложений, так и виде приложений, встраиваемых в Document Box.

**API**

1. Серверный, универсальный для всех клиентов системы
2. Web Services
3. Клиентский GXT-API

**Свойства приложений**

**Дерево папок в Document Box**

1. Ленивая загрузка дерева папок
2. Счетчики системных объектов в дереве папок можно сделать конфигурируемыми (например, "показывать", "не показывать", "обновлять раз в Х минут")
3. Предоставление возможности приложениям форсированного обновления дерева папок (возможно при помощи скрытого Action)

**Desktop**

1. В Desktop будут доступны все приложения, которые могут быть открыты в Document Box

**Виджеты**

1. Виджет (кнопка), позволяющий выполнить некоторое действие, свойственное данному приложению. Актуально для приложений, предоставляющих возможность выполнить действие не только из панели инструментов

**Доработки ядра**

1. Добавить возможность пересчета acl при смене статуса карточки, от которой зависят права на другие объекты. Сейчас такая возможность едсть только при смене атрибутов карточки.
2. Права на атрибуты. В случае если атрибут readOnly при попытке его изменить формировать Exception или игнорировать данные изменения.
3. Права на атрибуты. В случае если атрибут скрытый не добавлять его в sysobject при его зачитывание.
4. Сервис вложений. Модифицировать механизм так чтобы системные поля вложений (path, mimetype) невозможно было изменить пользователю через API модифицирования карточек. Возможность изменения данных полей должна остатся только через сервис вложений.
5. Необходимо "визуализировать" процедуру расчета прав. Тоесть админскую функцию, которая выдаст отчет какой пользователь и почему имеет те или иные права.
6. findSysobjectFromDocbase - реализация завязана только на один супертип

# Версионность данных

Сначала надо определится зачем нам нужна версионность, и какие операции с версиями нам понадобятся. Сначала опишем версионность карточек (документов и справочников).

1. Должна быть возможность отобразить в карточке ту версию связанного документа или справочника, на которую ссылался документ на момент установки связи.
2. Должна быть возможность отобразить в карточке последнюю версию связанного документа или справочника, на которую ссылается документ, даже если после момента установки связи появились новые версии объекта на который ссылается документ.
3. В списках документов необходимо отобразить поле из связанного документа, которое было на момент установки этой связи.
4. В списках документов необходимо отобразить поле из связанного документа, которое соответствует последней версии связанного объекта.
5. Должна быть возможность отобразить документ или справочник любой версии.
6. Должна быть возможность посмотреть список всех версий документа.
7. При формировании отчета в списках необходимо отобразить поле из связанного документа, которое было на момент установки этой связи.
8. При формировании отчета в списках необходимо отобразить поле из связанного документа, соответствующему последней версии связанного объекта.

Опишу способ, который используется в платформе Documentum, на мой взгляд удовлетворяющий всем требованиям. Каждый тип со всеми версиями хранится в отдельной таблице. В таблице есть два идентификатора r\_object\_id и i\_chronicle\_id и флаг последней версии is\_last. Есть еще поле метка версии, но оно принципиальной роли не играет. Поле r\_object\_id - уникальный идентификатор, однозначно определяющий объект в системе, является первичным ключем. Поле i\_chronicle\_id - идентификатор документа, определяющий идентификатор всех версий одного документа. То есть у всех версиях одного и того же документа идентификатор i\_chronicle\_id одинаковый. Причем у первой версии документа атрибуты r\_object\_id и i\_chronicle\_id совпадают. Поле is\_last - флаг, устанавливается в 1 только у последней версии документа. Итак рассмотрим как данная система ведет себя при реализации задач, описанных выше.

1. В случае, когда надо отображать атрибут объекта с определенной, не последней версии мы в ссылочный атрибут записываем идентификатор r\_object\_id того объекта, который в данный момент является последней версией (where is\_last=1). При этом, в любое время, получив объект по r\_object\_id мы получим атрибуты той версии, которая была последней, на момент установки связи.
2. В случае когда надо отображать атрибут объекта всегда последней версии, мы в ссылочный атрибут записываем идентификатор i\_chronicle\_id того объекта, который в данный момент является последней версией (where is\_last=1). При этом, в любое время, получив объект по двум условиям (where i\_chronicle\_id = 'id' and is\_last=1) мы получим атрибуты последней версии.
3. Получить запросом поле связанного объекта определенной версии select \* from type1 inner join type2 on (type1.type2\_id = type2.r\_object\_id)
4. Получить запросом поле связанного объекта последней версии select \* from type1 inner join type2 on (type1.type2\_id = type2.i\_chronicle\_id and type2.is\_last=1)
5. Получение атрибутов любой версии по идентификатору select \* from type where r\_object\_id = 'id'
6. Просмотр всех версий select \* from type where i\_chronicle\_id = 'id'
7. Аналогично п.3
8. Аналогично п.4

Недостаток данного подхода - пухнущая таблица, так как в ней будут содержатся все версии объектов. Этот недостаток будет сказываться на документах с большим количеством версий, а справочники не будут иметь большое количество версий, поэтому данный подход вполне приемлем для справочников. Для документов, на мой взгляд, требуются только следующие действия из тех которые я перечислил ранее:

1. Должна быть возможность отобразить документ или справочник любой версии.
2. Должна быть возможность посмотреть список всех версий документа.

Для реализации данной функции можно организовывать хранение в отдельной таблице, при этом данные в таблицу версий записывать сразу при создании или изменении документа в основной таблице, то есть в таблице версий у нас будет хранится и последняя и все предыдущие версии, что сделает код для работы с версиями более универсальным и простым. Что касается версионности вложения. На мой взгляд работа с вложениями должна полностью быть пересмотрена. Каждый документ должен иметь возможность содержать один или несколько вложений, причем каждое из этих вложений может иметь несколько версий. При этом на сервере, в бизнес слое, должно быть API для работы с вложениями. Реализация данного API может быть в модуле и быть разной от модуля к модулю.

# Наследование типов

**Конфигурация**

У конфигурации типов появился атрибут supertype. В нём можно указать тип от которого наследуется данный. Поддерживается иерархия наследований.

В системе данный тип ведёт себя как единое целое, то есть он включает в себя как свои атрибуты, так и атрибуты супертипа.

Если acl не указан у наследника, то он будет взят у супертипа. Аналогично для указания жизненного цикла по умолчанию.

**Хранение в БД**

Атрибуты "Тип", "Дата создания", "Статус" и т.д. хранятся в таблице супертипа, в таблице наследника хранится только идентификатор и новые атрибуты. Идентификатор в таблице супертипа совпадает с идентификатором в таблице наследника.

# Серверные точки расширения

На сервере существует механизм точек расширения, выполненный в виде сервиса ru.intertrust.cm5.core.api.extention.ExtentionService. Сервис позволяет в любом месте сервера, вызвать классы расширений, имплементящих определенный интерфейс.

## Создание новой точки расширения

Для создания новой точки расширения необходимо разработать интерфейс, который будет имплементить классы, реализующие точки расширения, и расширить этот интерфейс от ru.intertrust.cm5.core.api.extention.ExtentionPointBase

Далее в том месте, где будет точка расширения вызвать метод сервиса ExtentionPointBase.getExtentionPoint(Class), в который передается интерфейс точки расширения. Получаемый класс приводится к интерфейсу точки расширения. Далее у точки расширения вызывается соответствующий интерфейсу метод. При этом вызовется данный метод у всех классов, имплементящих интерфейс точки расширения и аннотированных @ExtentionPoint.

## Пример реализации точки расширения

Интерфейс точки расширения:

public interface BeforeSave extends ExtentionPointBase {

void onBeforeSave(SysObject sysobject, User user) throws ServerException;

}

Вызов точки расширения

BeforeSave beforeSaveExtention = (BeforeSave) extensionService.getExtentionPoint(BeforeSave.class);

beforeSaveExtention.onBeforeSave(document, user);

Пример реализации класса точки расширения

@ExtentionPoint

public class TestOnBeforeSave implements BeforeSave {

@Override

public void onBeforeSave(SysObject sysobject, User user) {

if (sysobject.getType().equals("test\_extention\_point")){

StringDataValue value = (StringDataValue) sysobject.getAttribute("name").getDataValue();

value.setValue(value.getValue() + "\_test\_before\_save");

}

}

}

## Точки расширения включеные в ядро

В ядро включены следующие точки расширения (пакет ru.intertrust.cm5.core.api.extention):

* AfterCreate - можно менять переданный sysobject в точке расширения
* AfterCreateUserTask - момент создания задачи для пользователя
* AfterPromote
* AfterSave
* BeforePromote
* BeforeRemove
* BeforeSave - можно менять переданный sysobject в точке расширения
* ReceivingNotification - момент принятия уведомления пользователем
* SendingNotification - момент отправки уведомления пользователю

# Автоматическое заполнение атрибутов

В системе реализована точка расширения, позволяющая заполнять атрибуты определенными значениями, вычисляемыми по специальным алгоритмам. Точка расширения имеет свою конфигурацию - тэг compuitedAttributeValue.

Пример конфигурации:

<config name="test\_attr\_computed">

<core:compuitedAttributeValue>

<core:rule valueTemplate="{CUR\_ATR}-{LINK}-{QUERY}-{CLASS}-{BACKLINK}" type="test\_doc" computed="once" attribute="computed\_once">

<core:pattern name="CUR\_ATR">

<core:value>

<core:link>name</core:link>

</core:value>

</core:pattern>

<core:pattern name="LINK">

<core:value>

<core:link>dictionary\_id.name</core:link>

</core:value>

</core:pattern>

<core:pattern name="QUERY">

<core:value>

<core:query>select name from test\_dictionary d where id=(:dictionary\_id)</core:query>

</core:value>

</core:pattern>

<core:pattern name="CLASS">

<core:value>

<core:className>ru.intertrust.cm5.core.test.resolver.NameAttributeHashValueResolver</core:className>

</core:value>

</core:pattern>

<core:pattern name="BACKLINK">

<core:value>

<core:link>dictionary\_id.test\_dictionary\_2(dictionary\_id).name</core:link>

</core:value>

</core:pattern>

</core:rule>

<core:compuitedAttributeValue>

</config>

Внутри compuitedAttributeValue определяются правила заполнения атрибутов. Правил может быть много.

Одно правило определяет как заполняется один атрибут определенного типа. Для зажания имени атрибута и типа служат атрибуты "type" и "attribute"

Атрибут valueTemplate опреджеляет формат значения поля. Внутри формата могут быть фрагменты, которые подлежат замене, в процессе формирования значения. Фрагменты обрамляются фигурными скобками. Текст не обрамленный фигурными скобками переносится в атрибут без изменений.

Атрибут computed служит для определения когда надо пересчитывать поле. Допустимые значения once - один раз при создании, always - каждый раз при сохранении.

Внутри тэга "rule" могут вставлятся один или более тэгов "pattern". Тэг "pattern" определяет правило замены одного фрагмента в valueTemplate. Какой именно фрагмент будет замещатся указывается в атрибуте "name" тэга "pattern"

Внутрии тэга "pattern" определен тэг "value", который определяет правило замены. Существует 3 алгоритма замены. link - ссылка на поле, с помощью языка ссылок query - в значение подставляется результат выполнения запроса. Подставляется первое поле перврй строки. className - в значение подставляется результат работы метода getValue класса, имя которого указано в теле тэга "className". Класс должен имплементить интерфейс CustomValueResolver.

Интерфейс CustomValueResolver

public interface CustomValueResolver {

Object getValue(SysObject sysObject) throws ServerException;

}

Пример реализации (в поле подставляется hash значение поля name)

public class NameAttributeHashValueResolver extends ApplicationContextSetter implements CustomValueResolver {

@Override

public Object getValue(SysObject sysObject) throws ServerException {

String name = (String) sysObject.getAttribute("name").getDataValue().getValue();

return String.valueOf(Math.abs(name.hashCode()));

}

}

# Требования к ядру

Ядро - это совокупность базовых объектов, сервисов для работы с ними и GUI-компонентов, на основе которых, как из кубиков, может быть реализован функционал системы документооборота (любого рода, включая канцелярский, деловой или др.) или смежных областей (например, архив).

|  |  |
| --- | --- |
| **Сервисы/компоненты** | **Требования** |
| Сервис конфигурации | Декларативное описание доменных объектов системы (документ, согласование…) без необходимости написания кода взаимодействия с хранилищем данных (в данный момент реляционной БД). |
| Сервис хранения данных | Максимизация производительности операций взаимодействия с хранилищем (в случае реляционной БД – отказ от вертикальной структуры хранения данных в пользу горизонтальной).  Минимизация времени отклика системы |
| Сервис для работы с карточками | Расширяемость системы без какой бы то ни было модификации кода ядра  Предоставление Public API для третьесторонних интеграторов системы |
| Инфраструктура поддержки модулей расширения |
| Сервис поиска | Поиск по атрибутам и содержимому карточек |
| Сервис для работы с ЖЦ | Декларативное/визуальное описание жизненных циклов объектов без необходимости написания/изменения кода |
| Интеграция движка Activiti |
| Сервис для работы с правами доступа. Механизм динамических ролей | Расчет прав доступа к доменным объектам системы (чаще всего – к документам, однако, с точки зрения ядра, это безотносительно). Максимизация производительности расчета прав |
| Сервис для работы с вложениями, версионность вложений | Добавление файловых вложений к объектам системы, обеспечение версионности вложений |
| Сервис для работы с системным журналом | Ведение журнала действий пользователей и действий с объектами системы |
| Сервис генерации отчетов | Генерация отчетов, установка шаблонов, формирование отчета по шаблону  Автоматическая фильтрация прав при использовании системы построения отчетов |
| GUI-инфраструктура, Document Box | Максимальное абстрагирование GUI от объектов системы.  Возможность использования ядра GUI в других системах документооборота и смежных областей |
| Приложение GUI «Список объектов» | Декларативное определение списков, отображаемых в системе, без необходимости написания/изменения кода.  Возможность создания сложных списков при помощи кода, в тех случаях, когда декларативных средств недостаточно |
| Сервис конфигурируемых списков |
| Составное приложение «список/редактор системных объектов» | Наличие приложения, состоящего из двух частей – списка объектов и детальной информации по выбранному объекту |
| Виртуальный рабочий стол | Концепция виртуального рабочего стола пользователя |
| Приложение «Редактор системных объектов» | Декларативное создание форм (как редактирующих, так и справочных), отображающих сущности системы, без необходимости написания/изменения кода.  Возможность создания сложных форм при помощи кода, в тех случаях, когда декларативных средств недостаточно |
| Сервис конфигурируемых форм |
| GUI компоненты отображения конфигурируемых форм |
| Основыне виджеты, отображающие атрибуты объектов системы |
| Сервис импорта/экспорта объектов системы из/в xml | Интеграция с третьими системами  Перемещение данных из одной инсталляции системы в другие |

# Экспорт и импорт объектов

**Экспорт**

Для экспорта объектов в DataService имеются два метода:

1. public String exportToXml(List<SysObjectKey> keys) - выгружает все объекты, переданные в качестве параметра, и объекты на которые они ссылаются
2. public String exportToXmlByType(String type) - выгружает все объекты указанного типа и объекты на которые они ссылаются

Оба метода возвращают xml закодированный в Base64

**Импорт**

Синтаксис схож с синтаксисом описания типа.

<importConfig xmlns="http://jaxb.core.cm5.intertrust.ru/ImportConfig">

<sysObjectInfo name="testchild1" type="dt\_test\_child">

<fields>

<field name="name">

<stringDataValue>child1</stringDataValue>

</field>

</fields>

</sysObjectInfo>

<sysObjectInfo name="testchild2" type="dt\_test\_child">

<fields>

<field name="name">

<stringDataValue>child2</stringDataValue>

</field>

</fields>

</sysObjectInfo>

<sysObjectInfo name="parent" type="dt\_test">

<fields>

<field name="name">

<stringDataValue>test</stringDataValue>

</field>

<field name="attr\_id1">

<idDataValue>testchild1</idDataValue>

</field>

<field name="attr\_time">

<datetimeDataValue>2001-10-26T21:32:52+02:00</datetimeDataValue>

</field>

<field name="attr\_bool">

<booleanDataValue>true</booleanDataValue>

</field>

<field name="attr\_long">

<longDataValue>123</longDataValue>

</field>

<field name="attr\_double">

<decimalDataValue>1.23</decimalDataValue>

</field>

</fields>

</sysObjectInfo>

<relation src="testchild2" dest="parent" destAttrName="attr\_id2" />

</importConfig>

Атрибут name - уникальное имя объекта в пределах конфигурации, которое используется для указания атрибутов типа ID.

Так же указать на атрибут типа ID можно с помощью тега relation:

dest- уникальное имя объекта в котором есть атрибут destAttrName, значением которого станет идентификатор объекта с уникальным именем src.

В DataService для импорта есть метод - void importFromXml(String xml), в качестве параметра которому передается xml закодированный в Base64.

# Язык ссылок

В системе разработан специальный язык ссылок. Это символьная строка, обозначающая алгоритм поиска одной карточки(ек) или ее атрибута(ов) относительно другой, базовой карточки. При написании ссылки используются имена типов карточек и имена атрибутов, которые описаны в конфигурации. Различают два вида ссылок: прямая и обратная.

## Прямая ссылка

С помощью данной ссылки можно указать на карточку, на которую ссылается базовая карточка с помощью своих атрибутов. Например предположим в карточки department(подразделение) есть поле boss\_id(руководитель). Нам необходимо сослатся на карточку руководителя относительно карточки подразделения. В этом случае ссылка будет иметь вид "boss\_id" иными словами имя атрибута в базовой карточке, в котором хранится идентификатор искомой карточки. Если нам надо получить какой то определенный атрибут в карточке, которую мы нашли по ссылке то имя данного атрибута надо написать после имени ссылочного атрибута. Например имя руководителя в предыдущем примере мы получим с помощью ссылки "boss\_id.name". По прямой ссылке мы получаем ноль или одну карточку или ноль или один атрибут.

## Обратная ссылка

С помощью данной ссылки мы указываем на карточки, которые в своих атрибутах имеют ссылку на базовую карточку. Например предположим существует тип organization(организация) и тип department(подразделение). У типа department есть атрибут organization\_id(идентификатор организации) типа ID. Необходимо сослатся на все подразделения организации относительно организации. В этом случае ссылка будет иметь вид department(organization\_id). Снавала указывается имя типа, который ссылается на базовый, затем в круглых скобках имя атрибута, в котором хранится ссылка на базовую карточку. Если после закрывающейся скобки мы укажем имя не ссылочного атрибута, мы получим значение атрибута в ссылающейся на базовую карточку карточке. Например для получения имен подразделения из вышеописанного примера ссылка будет иметь вид department(organization\_id).name. По обратной ссылке мы можем получить ноль или много карточек или ноль или много значений атрибутов.

## Особый вид ссылки: ссылка на самого себя

Иногда необходимо в конфигурации написать особый вид ссылки на самого себя. Для этого существует символ точки "." или имя атрибута с типом отличным от ID. Напрмер объект, который вернется относительно карточки сотрудника по ссылке "." будет сома карточка сотрудника, а по ссылке "name" вернется имя текущего сотрудника.

## Группировка ссылок

Ссылки разного типа можно группировать неограниченное количество. Например данная ссылка относительно организации "department(organization\_id).boss\_id.name" вернет массив имен руководителей подразделений для текущей организации.