**Концепция использования Activiti в CM5**

Целью интеграции Activiti в платформу CM4.5 является оценка возможности использования данного движка для решения задач маршрутизации и обработки доменных объектов в платформе CM5 и последующих, оценки удобства написания и понятности процессов используя нотацию BPMN2, оценки быстродействия. Данная цель будет достигаться путем внедрения движка Activiti в платформу CM4.5, написания обертки движка для удобства работы с ним и создания необходимой инфраструктуры для удобного взаимодействия движка Activiti и остальных сервисов платформы, написания тестовых процессов, которые должны охватывать максимальное количество механизмов, которые потребуются при реализации реальных бизнес процессов в последующих версиях платформы.

Перечень задач, которые будут решаться 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 CM5
2. Данные Activiti хранятся в той же базе данных, что и данные CM5
3. Activiti использует информацию о пользователях и группах из CM5
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).

## Обще используемые 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 |
|  |  |  |
|  |  |  |

## Тебования к интеграции Activiti

При встраивание движка необходимо максимально облегчить работу с ним для конечных 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.