|  |  |
| --- | --- |
| Логотип_Глобус_L | CM-Sochi |
|  |
| версия документа 1.0  редакция № от 03.09.2013 |

Эскизный проект

по интеграции компонент CMJ (CM4) с платормой CM-Sochi (CM4/5)

История изменений

|  |  |  |  |
| --- | --- | --- | --- |
| Дата | Версия | Описание | Автор |
|  | 1 |  |  |
| 30.08.2013 | 2 | Уточнения по результатам согласования.   * + 1. Уточнена работа с комплектами БД     2. Раздел про отключение AFSaveDocUtils |  |
|  |  |  |  |

Содержание

[1. Сокращения 4](#_Toc365647216)

[2. Исходные требования 4](#_Toc365647217)

[3. Компонентная модель 4](#_Toc365647218)

[4. Метод реализации 5](#_Toc365647219)

[5. СС и корпоративность 6](#_Toc365647220)

[6. Принципы реализации адаптера к платформе CM-Sochi 9](#_Toc365647221)

[6.1 Переиспользование кодовой базы CMJ-Server 9](#_Toc365647222)

[6.2 Принципы модификации ДОП 9](#_Toc365647223)

[6.3 Даты создания и модификаций 10](#_Toc365647224)

[6.4 Идентификатор сущности (UNID) 10](#_Toc365647225)

[6.4.1 Назначение UNID 11](#_Toc365647226)

[6.4.2 Дублирование UNID 12](#_Toc365647227)

[6.4.3 NoteId 13](#_Toc365647228)

[6.5 Evaluate 13](#_Toc365647229)

[6.6 Search 13](#_Toc365647230)

[6.7 Моделирование lotus.domino.Document 13](#_Toc365647231)

[6.7.1 Респонсы 14](#_Toc365647232)

[6.7.2 lotus.domino.Item 14](#_Toc365647233)

[6.7.3 lotus.domino.EmbeddedObject 14](#_Toc365647234)

[6.8 Моделирование lotus.domino.Database 15](#_Toc365647235)

[6.8.1 Сервер БД (Server) 15](#_Toc365647236)

[6.8.2 Путь к БД (fileName) 16](#_Toc365647237)

[6.8.3 Идентификатор БД (ReplicaId) 16](#_Toc365647238)

[6.8.4 Роли 16](#_Toc365647239)

[6.9 Моделирование lotus.domino.View 16](#_Toc365647240)

[7. Агенты 16](#_Toc365647241)

[8. Настраиваемый набор реквизитов 17](#_Toc365647242)

[9. Хранение форматированного текста 17](#_Toc365647243)

[10. Аутентификация 17](#_Toc365647244)

[11. Авторизация 18](#_Toc365647245)

[11.1 Моделирование авторизации доступа к документу в CM-Sochi 18](#_Toc365647246)

[11.1.1 Пользовательский доступ 18](#_Toc365647247)

[11.1.2 3 Доступ делопроизводителям 19](#_Toc365647248)

[11.1.3 Наведённый доступ 19](#_Toc365647249)

[12. Справочники 20](#_Toc365647250)

[13. Уведомления 20](#_Toc365647251)

[14. ПКД 20](#_Toc365647252)

[15. ChLog 21](#_Toc365647253)

[16. История изменений документа 21](#_Toc365647254)

[17. Сервис AFSaveDocUtils 21](#_Toc365647255)

[18. Полнотекстовый поиск 22](#_Toc365647256)

[19. Список задач для рефакторинга CMJ-Server 22](#_Toc365647257)

1. Сокращения

ДОП – доменный объект платформы CM-Sochi.

РБД – реляционная база данных

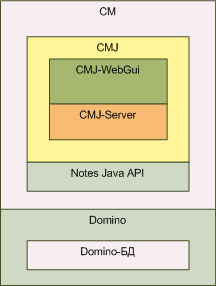
1. Исходные требования

Исходные требования для разработки дизайна кода:

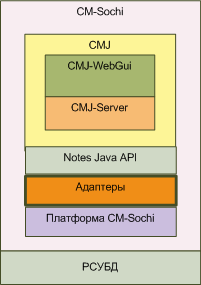
* Переиспользовать код СМJ-Server с минимальным изменением его кодовой базы.
* Слой сопряжения СМJ-Server и платформы CM-Sochi должен позволять подменять свою имплементацию прозрачно для СМJ-Server. Верифицируется возможностью замены платформы СМ-Sochi.
* Имена атрибутов хранимых объектов должны быть читабельны, в отличие от исторически сложившихся интуитивно непонятных имен используемых CMJ для полей Notes-документов.

1. Компонентная модель

В документе используется следующая компонентная декомпозиция имеющейся системы CompanyMedia 4.x (CM):



Компонентная модель разрабатываемой CompanyMedia-Sochi (CM-Sochi) она же CompanyMedia 4/5:



В рамках проекта CM-Sochi разрабатывается «Платформа CM-Sochi» и «Адаптеры к платформе CM-Sochi», призванные заменить платформу Domino. Предполагается переиспользовать существующие компоненты, входящие в CMJ.

1. Метод реализации

Основное направление для проектирования слоя сопряжения CMJ (CM4) с Платформой CM-Sochi – через собственную имплементацию основных интерфейсов Notes Java API. Слой, содержащий эту имплементацию, назвали "Адаптер к платформе".

Для реализации потребуется собственная имплементация всех интерфейсов и классов Notes Java API, используемых в CM4. Эту собственную имплементацию следует оформить в виде отдельного jar (допустима и иерархия jar). На уровне сборочных скриптов, подключать в classpath этот jar вместо библиотеки с Notes Java API. Тогда код CMJ-Server автоматически будет использовать классы адаптера. Правда, при этом исключается возможность использовать Notes Java API в рамках продукта, что было бы потенциально полезно, например, для интеграционных задач.

Чтобы обеспечить возможность использования и Notes Java API, можно сделать другую имплементацию класса NotesFactory, создающую Session с адаптерной имплементацией. Для этого необходимо отрефакторить CMJ-Server для возможности конфигурацией фабрик определить, какую имплементацию NotesFactory использует CMJ-Server. Также необходимо собственную декларацию интерфейсов Notes Java API оформить в виде отдельного jar, который будет исключаться из проекта, если присутствует оригинальное Notes Java API с собственной декларацией интерфейсов.



1. СС и корпоративность

В зависимости от потребностей внедрения при инсталляции системы создаётся конфигурация состава модулей, входящих в данную инсталляцию и режим их совместного использования разными организациями. Справочник, хранящий описание такой конфигурации называется «Структура системы» (СС). Если модуль сконфигурирован как общий для нескольких организаций, он называется «корпоративным». Корпоративным чаще всего делается справочник внешних организаций (СпО), Входящие документы (ВхД), Исходящие документы (ИсхД). Ещё существует понятие «комплект модулей». Модуль одного комплекта взаимодействует только с модулями своего комплекта. Если модуль одного комплекта ищет другой модуль, для которого в конфигурации нет привязки к тому же комплекту, тогда взаимодействие осуществляется с модулем, для которого не задан комплект. Однотипные базы разных комплектов в целом похожи, но могут быть небольшие отличия, не меняющие интерфейс взаимодействия между модулями.

СС в CM-Sochi должна быть общей для всех организаций, работающих с Системой. Для описания СС в CM-Sochi, через разрабатываемый административный клиент должна быть задана следующая структура данных.



* ModuleType – объекты данного класса описывают множество всех типов модулей поставляемых с Системой. Например, их состав задаётся в xml-файле, а административный клиент считывает их от туда.
  + Id используется для поиска модулей заданного типа. (Аналогичная сущность в СМ4 называется "Тип программы", а id – это нынешние “Missions”, “InputDocs”). Значения id захардкожены в CMJ-Server.
  + isCorporabale – признак, могут ли модули данного типа быть «корпоративными». Если false, Module может быть связан лишь с 1 SystemOrganization
  + isComplectable – признак, могут ли модули данного типа помещаться в комплекты. Если false, в Module нельзя заполнять поле Complect.
* SystemOrganization – описатель организации. Объекты данного типа должны создаваться через АРМ администратора и сохраняются в РБД.
  + SOModule\_Id – идентификатор модуля «справочник организаций» (СО) для данной SystemOrganization.
  + shortName – краткое наименование организации.
* Module – описывает изолированную инсталляцию модуля. Если объект связан более чем с одной SO, то модуль «корпоративный». Объекты данного типа должны создаваться через АРМ администратора и сохраняются в РБД.
  + Id – сгенерированный уникальный идентификатор инсталляции. Он не должен изменяться на протяжении всего периода эксплуатации системы.
  + Complect – имя «комплекта», к которому приписан модуль.
  + FilePath – фиктивное имя файла модуля, для имитации получения модуля по пути к файлу. Значение должно быть уникальным среди всех модулей.
  + Title – отображаемое пользователям уникальное наименование модуля.
* RKK – пример хранимой сущности, принадлежащей некоторому модулю. Принадлежность определяется наличием ссылки на объект Module.

Один из вариантов схемы развёртывания CM-Sochi:



Сейчас важно отметить, что каждая инсталляция CMJ-Server привязывается настройками к одной СО. Т.о. пользователи, заходя на сервер с инсталляцией CMJ-Server работают в контексте известного модуля СО и должны быть сотрудниками данной организации. В первой версии CMJ-Sochi не планируется обеспечивать федеративный режим работы Системы, поэтому экземпляры CMJ-Server, входящие в изолированную Систему, должны работать с одной РБД. Администратор, формируя СС, сохраняет его описание в общей РБД, т.о. обеспечивается единый СС для всех организаций (СО).

Поиск модуля по его ReplicaId моделируется в CM-Sochi поиском Module по Module.Id. Поиск модуля по системному идентификатору БД и идентификатору СО моделируется поиском Module, связанного с заданным ModuleType.Id и SystemOrganization.SOModule\_Id.

В CMJ-Server некоторые Database не регистрируются в СС, а значит, нет возможности считать их ModuleTypeId из СС (СО, СС, СОК, Классификатор связей). Такие Database всегда получаются по их ReplicaId, получаемых из настроек.

При заведении через АРМ администратора состава организаций, входящих в систему, также в РБД должен сохраниться Module, характеризующий СО для данной организации и связанный с ModuleType=“SO”. В поле SystemOrganization. SOModule\_Id должен быть вписан идентификатор созданного Module для СО. Комплект для СО задавать запрещено. Модуль СО должен быть привязан к одной SystemOrganization. Тогда при поиске Database по ReplicaId единообразно с прочими БД будет находиться и модуль СО и определяться его ModuleTypeId. В настройки CMJ-Server в качестве cmj.af.so.replicaID должен заноситься именно Module.Id.

Похожим образом следует поступить и с СС. Только Module для СС должен создаваться при первом запуске АРМ администратора, а при добавлении SystemOrganization, создаваться и ассоциация Module СС с этой новой SystemOrganization.

Модули СОК и Классификатор связей в CM-Sochi должны определяться через АРМ Администратора, как и все прочие модули. Это приведёт к их занесению в общий список Module. При получении их ReplicaId из «Настроек СО» и последующим поиском модуля по ReplicaId сработает общий механизм поиска модулей.

Понятие «Сеть организации» относится к конфигурированию распределённой работы СМ. В первом релизе CM-Sochi организовывать распределённость не требуется. В связи с этим понятие «Сеть организации» в CM-Sochi не реализуем. Адаптеры в качестве имени «сети» должны возвращать пустую строку.

1. Принципы реализации адаптера к платформе CM-Sochi
   1. Переиспользование кодовой базы CMJ-Server

Чтобы обеспечить переиспользование CMJ-Server без изменений кода, можно включать код CMJ-Server в Java-проект, как внешние библиотеки. При этом допустимо специфичные CM-Sochi варианты имплементации переключать через конфигурации, такие как web.xml и xml-дескрипторы бинов.

В CMJ-Server всё же придётся немного вмешаться для обеспечения такого способа переиспользования. В частности, перенести из web-модуля “rest” все классы в какой-нибудь нижестоящий jar-модуль.

* 1. Принципы модификации ДОП

Все сущности в CMJ сохраняются через объект с интерфейсом lotus.domino.Document. При отображении Document в объекты платформы, как правило, необходимо будет задействовать множество разнотипных доменных объектов платформы (ДОП), например, для сохранения множественных значений полей. Т.е. одному Document соответствует N ДОП, которые связаны между собой в древовидную структуру. Корень дерева будем называть «корневым ДОП».

Возможны два подхода для реализации модификаций ДОП, ассоциированных с экземпляром Document. Первый – «считать, сравнить, модифицировать, записать». Т.е. адаптер при сохранении Document получает все свои ДОП, сравнивает какие поля модифицировались, определяет по ним какие именно ДОП надо обновить и обновляет только их.

При втором подходе можно обойтись без предварительного чтения и сравнения. Т.е. при условии, что есть полный набор значений всех атрибутов, можно сформировать в памяти все ДОП в их новом состоянии и сохранить. В том числе надо не забыть установить и идентификатор версии ДОП для срабатывания оптимистической блокировки. Операций чтения из СУБД при этом нет (нет SELECT), а только обновления (UPDATE).

При втором подходе особо надо отметить операцию удаления из состава связанных ДОП. С уровня REST приходит информация о новом наборе связанных ДОП, но нет данных какие элементы были удалены. Поэтому, чтобы определить, есть ли удалённый элемент, надо считать идентификаторы связанных объектов из БД, определить какие лишние, сравнив с данными из REST, и удалить их.

Второй подход с одной стороны эффективнее, т.к. длительность операций обращения к РБД больше при передаче и выполнении большего числа пакетов SQL-операторов, нежели при увеличении числа SQL-операторов в пакете. С другой стороны, его применение практически редко возможно из-за сложностей с удалением ассоциированных ДОП, и неэффективностью если чтение ДОП по каким-либо причинам было произведено. А причин для чтения множество, например, необходимость ведения истории изменений РКК, основанной на сопоставлении хранимой в БД информации, с переданной через REST.

Т.е. для адаптеров можно ограничиться применением лишь первого подхода – «считать, сравнить, модифицировать, записать».

Сохранение всех ДОП, соответствующих одному NotesDocument должно происходить атомарно в одной транзакции. Но допустима транзакция, включающая сохранение нескольких NotesDocument. Т.е. для имплементации метода Document.save() нужен propagation = REQUIRED

* 1. Даты создания и модификаций

У объектов Document есть свойства, указывающее дату его создания и модификации. При его отображении на иерархию ДОП, в качестве даты создания можно использовать дату создания корневого ДОП.

Дата модификации используется CMJ-Server как интегральный идентификатор версии сущности. Платформа CMJ-Sochi отслеживает дату модификакции каждого ДОП независимо. Поэтому, для получения даты модификации средствами платформы, необходимо обходить всю иерархию ДОП и выбирать максимальную дату среди всех ДОП. Это для многих операций неэффективно, поэтому предлагается на уровне адаптера, при сохраеннии любого ДОП из иерархии, выполнять фиктивное сохранение корневого ДОП, чтоб его дату модификации обновить и использовать в дальнейшем, как интегральный идентификатор версии всей иерархии ДОП.

У нового, ещё не сохранённого ДОП, дата создания и модификации определена, в отличие от id.

* 1. Идентификатор сущности (UNID)

Notes Java API при создании нового Document в памяти, даже если он ещё не сохранён, позволяет получить его UNID. После сохранения Document, его UNID не меняется.

Адаптер, для реализации операций получения по UNID (Database. getDocumentByUNID()) должен смочь на основании переданного UNID определить идентификатор корневого ДОП. Удобно закодировать внутри UNID и идентификатор ДОП (в первые 16 символов UNID). Но API платформы CM-Sochi, при создании нового ДОП (CrudService.createDomainObject()) не назначает ему идентификатор до его сохранения. Таким образом, не удастся обеспечить св-во неизменности UNID нового Document, и его же, после сохранения.

Платформа CM-Sochi генерирует уникальные идентификаторы для ДОП.

В CMJ-Server есть места, полагающиеся на структуру UNID, в частности, что в последних 16 символах закодирована дата создания документа.

CMJ принципиально важно, чтобы UNID был строкой длинной в 32 символа. Набор символов любой, лишь бы символы могли без перекодирования использоваться в URL, и были регистронезависимы.

Есть «техническое» создание Document, не предназначенное для получения или модификации хранимой в БД информации. Такое, как генерация уникальных идентификаторов, через получение UNID нового документа. Создание поля типа ResponseReferenceList, через MakeResponse и модификацию UNID. Значит, нужно позволять создавать Document без связи с ДОП.

Т.о. в 32х символах UNID должно быть закодировано с возможностью обратного раскодирования:

* идентификатор ДОП
* в последних 16 символах – дата создания документа в том же формате, что и Notes Java API.

При получении коллекции, платформа в качестве id отдает id ДОП. Чтобы адаптер смог сформировать полноценный UNID, требуется ещё дата создания, поэтому в коллекции в обязательном порядке требуется добавлять поле, содержащее дату создания корневого ДОП.

* + 1. Назначение UNID

CMJ-Server практически всегда (но не в 100% случаев) применяет самостоятельное назначение UNID. К тому же, в нём реализован свой слой доменных объектов, которые хранят UNID без связи с Document. При создании доменного объекта, UNID генерируется и сохраняется в поле доменного объекта, а когда он сохраняется, то UNID назначается создаваемому Document. При чтении доменного объекта, UNID считывается из Document и запоминается в поле.

В CMJ-Server есть и другие случаи назначения UNID, например, при создании листа согласования, резервируются UNID’ы для будущих виз. При создании визы, ей назначается UNID из списка ранее зарезервированных.

Для решения проблем c назначением UNID CMJ-Server’ом, предлагается иметь возможность однозначного мэппинга UNID на id ДОП. Для чего завести специальный ДОП для мэппинга, содержащего 2 поля:

* UNID (уникальный первичный ключ)
* соответствующий id Доп (уникальный).

Чтобы сэкономить на количестве обращений к таблице мэппинга можно реализовать ряд оптимизаций.

Можно в UNID, формируемый адаптером на основании id ДОП, установить специальный бит, идентифицирующий, что UNID выдан платформой CM-Sochi . Назовём такой тип UNID – «P-UNID», а назначенный тип – «N-UNID» При получении документа по P-UNID, будет происходить проверка этого бита, и если бит установлен, то корневой ДОП получается сразу. Если не установлен, то предварительно по таблице мэппинга происходит поиск id ДОП.

Если изначально объект создается с назначенным CMJ-Server’ом UNID, то как мы избавимся от обращения к таблице мэппинга? До тех пор, пока «жив» доменный объект CMJ-Server'а, который порадил новый Document, им должен предоставляться сгенерированный N-UNID, требующий мэппинга. Но когда доменный объект порождается путём считывания из платформы, он должен получить P-UNID, собранный на основании id ДОП. Предварительный анализ показал, что отличие запрошенного UNID, от предоставляемого доменным объектом, не повлияет на логику CMJ-Server, приводя лишь к редким случаям кэш-промаха у HTTP-КЭШей.

В общем случае, на момент обращения к методу Document.getUniversalId() в адаптере могут ещё не быть заполнены поля, по которым можно определить тип корневого ДОП, который необходим для получения id от платформы. В этом случае адаптер должен сгенерировать N-UNID самостоятельно. Сгенерированный N-UNID адаптер должен запомнить и отдавать на запросы getUniversalId(). При первом сохранении, будет получен id корневого ДОП и в таблицу мэппинга должно быть занесено соответствие сгенерированного N-UNID и id ДОП. До уничтожения объекта Document, getUniversalId() должен продолжать возвращать N-UNID. Если же создаётся новый объект Document, через считывание из платформы той же самой сущности, то по запросу getUniversalId() должен возвращаться уже её P-UNID.

Если же при вызове метода Document.getUniversalId() в адаптере заполнены поля, по которым можно определить тип корневого ДОП, то можно получить сразу P-UNID. При сохранении такого Document запись в таблицу мэппинга не требуется.

* + 1. Дублирование UNID

В CMJ-Server используется операция присваивания UNID для быстрого нахождения сущностей из одного модуля, однозначно соответствующих сущности из другого модуля. Например, сущность, содержащая информацию о блокировке документа. Т.о. UNID становится уникальным лишь в контексте модуля (Database).

Надо учесть, что копироваться может как P-UNID, так и N-UNID. Получение второго Document может запрашиваться как P-UNID, так и N-UNID первого Document. При этом, P-UNID у второго Document должен быть свой. На запрос ко второму Document getUniversalId() должен возвращаться P-UNID первого Document если Document получен из платформы. Если второй Document не получен из платформы, то getUniversalId() должен возвращать тот UNID, который ему был назначен.

Для решения предлагается захардкодить список ModuleTypeId баз данных, сущностям которых могут назначаться дублирующиеся UNID и для операций получения и сохранения сущностей в Database из списка, применять альтернативный алгоритм. Предварительный анализ показал, что CMJ-Server если и назначает дублирующийся UNID, то делает это для всех сущностей Database. Поэтому, решение о применении альтернативного алгоритма можно делать на основании принадлежности к Database. Документы из таких специальных Database получаются только по UNID, т.е. не по представлениям (коллекциям) и не поиском.

Альтернативный алгоритм заключается в следующем. Если скопирован N-UNID, то при сохранении Document необходимо сначала по первой таблице мэппинга получить P-UNID исходного Document и занести в ещё одну таблицу мэппинга соответствие назначенного P-UNID и id корневого ДОП, соответствующего сохраняемому Document. Получить P-UNID возможно, т.к. CMJ-Server всегда выполняет копирование UNID из уже сохранённого Document. Если же скопирован P-UNID, то сразу заносим соответствие во вторую таблицу.

При получении документа по N-UNID исходного Document, сначала получается P-UNID по первой таблице мэппинга, затем по P-UNID по второй таблице находится id ДОП. При получении документа по P-UNID делаем тоже самое, пропуская шаг получения P-UNID. P-UNID запоминаем в поле адаптера, чтобы возвращать его на запрос getUniversalId().

Вместо второй таблицы мэппинга, можно задать дополнительный атрибут для ДОП целевого Document. В атрибут записывать исходный P-UNID и выполнять поиск ДОП по этому атрибуту.

* + 1. NoteId

У Document помимо UNID есть другой идентификатор – NoteId. Этот идентификатор не является глобально уникальным. Уникален лишь в контексте модуля (Notes БД) и его конкретной реплики. CMJ-Server воспринимает NoteId, как строку произвольной длины. Адаптер в качестве NoteId вполне может использовать тот же UNID.

* 1. Evaluate

В CMJ-Server используется функция Evaluate (51 раз), которая исполнет язык @-формул. Вариант написать свой интерпретатор @-формул не пройдёт по экономическим соображениям. Поэтому, без подключения Notes Java API мы практически не можем использовать Evaluate. Во многих местах код можно переписать на аналоги без Evaluate. Но некоторые места требуют изменения концепции реализации, такие как:

* Вычисление списка получателей уведомлений
* Вычисление информации о связанном документе (DocInfo)
* Вычисление данных для синхронизации между документами (AFSaveDocUtils)

Два первых случая считывают формулы из хранимых настроек. В CM-Sochi можно в настройках вместо @-формул задавать выражения на другом языке (Groovy/SpEL+DOEL), а адаптерами выполнять их вычисления при вызове Evaluate.

В третьем случае @-формулы задаются LotusScript-кодом либо кодом CMJ-Server’а, а вычисляются CMJ-Server’ом через Evaluate.

* 1. Search

Метод поиска объектов Document (Database.search) тоже использует @-формулы для определения критериев поиска. Его применение в адаптерах так же проблематично.

Возможным решением, обеспечивающим минимальное расхождение кода СМ4 и CM-Sochi, может быть рефакторинг CMJ-Server в CM4, чтобы операции, использующие @-формулы были вынесены в отдельные программные компоненты, подключаемые через конфигурацию, например, через xml-дескрипторы Spring Beans.

* 1. Моделирование lotus.domino.Document

Имплементация в адаптере интерфейса Document может быть независима от типа моделируемой сущности. На момент создания экземпляра Document тип сущности, которая будет описана этим Document, ещё не известен. Тип можно определить на основании поля Form (в некоторых случаях и ещё от ряда полей) и типа Database, в которой создаётся Document. Т.е. тип сущности становится определён только после заполнения её полей. Поэтому имплементация Document должна самостоятельно хранить значения полей, по крайней мере, до момента первого её сохранения. В момент сохранения Document, на основании полей, можно определить тип сущности и динамически подгрузить программную компоненту, специфичную типу сущности (именованный Spring Bean), которая и будет выполнять дальнейшую работу по взаимодействию с платформой CM-Sochi (Document реализует паттерн Bridge).

Для каждого типа сущности, в платформе будет соответствовать иерархия ДОП. Компонента, при сохранении Document, должна разложить свои поля по соответствующим ДОП. После сохранения ДОП, платформа предоставит Id для каждой ДОП. Document должен включить идентификатор корневого ДОП в свой UNID. Тогда в дальнейшем, при чтении Document по предоставленному UNID, можно будет найти корневой ДОП и по нему получить всю иерархию ДОП.

Компонента, реализующая взаимодействие с платформой, может представлять собой композицию бинов, каждый из которых реализует свой алгоритм сериализации полей Document в ДОП. Алгоритмы такие, как конвертировать %-имя в ссылки на ДОП справочника организации, или сохранить файл вложения, или просто переместить значение поля Document в соответствующее поле указанного ДОП.

Платформе должны быть предоставлены xml-конфигурации с описанием типов ДОП, моделирующих сущность в платформе.

* + 1. Респонсы

В Notes Java API есть поддержка специализированного типа отношений 1:N между объектами Document – «респонс» (response). Операции:

Document.getResponses() – получение коллекции документов-респонсов

Document.getParentDocumentUNID() – получение UNID документа к которому данный Document является респонсом.

Document.makeResponse(Document parent) – сделать Document респонсом к parent.

Чтобы смоделировать данные св-ва для сущности, которая может быть респонсом, в описателе типа её корневого ДОП, необходимо задать поле c зарезервированным именем «\_\_Ref».

* + 1. lotus.domino.Item

Интерфейс представляет одно поле Domino-документа. Объекты этого класса получаются только через объект-Document и используются как альтернативный способ чтения/записи значений полей в документ (сам документ дает также "прямой" способ). Адаптер, реализующий данный интерфейс, должен использовать обращения к "родительскому" Document для операций чтения/записи значений в поле, представляемое данным Item’ом.

* + 1. lotus.domino.EmbeddedObject

Работа с файлами вложений в Domino осуществляется либо через интерфейс EmbeddedObject, получаемый из Item, либо через методы интерфейса Document.

Платформа CM-Sochi даёт возможность создать специализированный ДОП для хранения файлов. Набор возможных типов файловых ДОП указывается при определении типа контейнерного ДОП. К файловому ДОП нельзя добавлять дополнительные поля. Имя типа файлового ДОП должно быть уникальным в рамках всей системы. Имя типа ДОП не должно превышать 25 символов.

Для сохранения файлов через адаптер, на каждое поле, которое может содержать вложение, должен быть определён отдельный тип файлового ДОП. Имя типа файлового ДОП должно содержать в себе имя соответствующего поля и тип контейнерного ДОП, например «DSP\_InputDocs\_Rkk\_**Body**». Если требуется хранить файл без привязки к полю, следует использовать зарезервированный суффикс для имени типа «$Attach», например «InputRkk\_$Attach»

**Примечание.** Платформав качестверазделителя в имени типа ДОП допускает только знак подчёркивания «\_». Возможность использовать знак «$» под вопросом.

Если сохранение происходит через интерфейс EmbeddedObject, по нему можно определить имя поля, через родительский Item и тип корневого ДОП через родительский Document и использовать их для формирования имени типа файлового ДОП. Имя файла сохранять в системном поле Name. Оно не требует уникальности имён файлов.

При получении файлов через EmbeddedObject, адаптер должен получить все файловые ДОП соответствующего типа.

Сохранениие файла через интерфейс Document, производится аналогично, но в качестве имени типа файлового ДОП используется тип корневого ДОП для данного Document, плюс зарезервированное имя «$Attach».

* 1. Моделирование lotus.domino.Database

Объект класса lotus.domino.Database представляет в Notes Java API отдельную БД Notes. Это значит, что все lotus.domino.Document, логически хранятся внутри Database и доступны через её методы. Коллекции (представления) и атрибутивный поиск (Search) предоставляют выборку только среди множества сущностей, хранимых в этой БД Notes. Можно сказать, что объект Database определяет отдельное пространство имён для хранимых в ней сущностей.

Бизнес-модули в CMJ моделируются отдельными Database. Каждый Database может содержать как разнотипные сущности, так и сущности одинаковой структуры (резолюции, карточки исполнения и т.п.)

Моделировать Database в платформе можно двумя способами. Первый – заведение отдельных типов ДОП для сущностей из разных типов БД, даже если у них одинаковая структура. Второй – для одинаковых сущностей не заводить отдельный тип ДОП, но при выборках задавать дополнительный фильтр, идентифицирующий БД.

Первый способ, с определением отдельных типов ДОП , эффективнее и в плане производительности, и в плане точности повторения поведения Notes БД, но делает задачу описания типов ДОП более избыточной. Избыточность частично компенсируется возможностью наследовать тип ДОП от другого, без добавления новых полей. При этом, такая конфигурация должна приводить к формированию отдельной таблицы в РБД, соответствующей описываемому типу, которая по структуре будет аналогична таблице прототипа.

Для моделирования поведения Database видится достаточным, чтобы адаптер интерфейса Database имел поля, идентифицирующее её тип –“moduleType.Id” и Module.complect. Module.complect+ModuleType.id должен использоваться, как префикс к именам коллекций и типам ДОП, запрашиваемых от платформы посредством объектов, полученных из Database. Если тип Module.complect+ModuleType.id отсутствует, то адаптер должен использовать только ModuleType.id. Такой подход позволит легко определять модули идентичной структуры в разных комплектах и, в тоже время, допускать разработку немного отличающихся однотипных модулей. От платформы необходим сервис для определения существования типа ДОП по его имени.

Надо учитывать ограничение платформы на длину имён типов ДОП в 25 символов. Максимальная длина ModuleType.id, используемая CMJ-Server – 18 символов. Выходит, необходимо сжимать Module.complect+ModuleType.id+тип ДОП, желательно так, чтобы результат был человеко-понимаемым.

* + 1. Сервер БД (Server)

В качестве имени сервера предлагается использовать DNS-имя хоста, где работает адаптер.

* + 1. Путь к БД (fileName)

Путь к файлу Database (Database.getFileName(), Session.getDatabase()) в CMJ-Server всегда берётся из каких-либо настроек. Предлагается в качестве имени файла использовать формат:

<*SystemOrganization.shortName*>[**/**<*Complect*>]**/**<*ModuleTypeId*>**.nsf**.

В настройках путь указывать в аналогичном формате.

* + 1. Идентификатор БД (ReplicaId)

ReplicaId в CMJ-Server всегда берётся из каких-либо настроек. Важно, чтобы для Database с одним ModuleTypeId, его ReplicaId была неизменной всегда. ReplicaId должна быть длиной 16 символов, применимых в URL без конвертации и не чувствительной к регистру. Предлагается в качестве ReplicaId использовать ModuleId в шестнадцатеричном представлении в верхнем регистре, дополненном лидирующими нулями до 16 символов.

* + 1. Роли

В CMJ-Server пользователь для каждой Database в отдельности может назначаться на роль, предоставляющую или ограничивающую ему права на видимость документов, либо определяющую состав разрёшенных операций, либо состав отображаемых реквизитов документов. Определение состава ролей пользователя выполняется через метод Database.queryAccessRoles(String userName).

Для моделирования поведения queryAccessRoles в CM-Sochi, для каждой роли каждого Module должна быть заведена статическая группа. Имя группы должно содержать префикс с ModuleTypeId далее разделитель «точка» и имя роли в квадратных скобках. Пример: “InputDocs.[Supervisor]”

От платформы CM-Sochi требуется возможность через API запросить состав групп, в которые входит пользователь. Если не будет специализированного API можно воспользоваться системными ДОП «USER\_GROUP» и «GROUP\_MEMBER». Платформа вернёт все группы, для всех Database и прочие группы. Метод queryAccessRoles должен отфильтровать их по префиксу ModuleTypeId и наличию квадратных скобок.

* 1. Моделирование lotus.domino.View

View используется для предоставления сортированных списков документов и поиска документов по ключу. View содержит колонки, содержащие значения полей Notes-документов, либо результат вычислений над полями Notes-документа. View получается через Database.

View в CM-Sochi надо моделировать через «коллекции» платформы. Имя коллекции при её запросе через Database.getView(name) составляется как Module.complect плюс ModuleType.Id плюс имя вида name. Если такой коллекции нет, то только ModuleType.Id + name. Поиск по виду – это применение фильтра коллекции.

1. Агенты

Логику работы агентов необходимо портировать на Java. Эмуляцию синхронного запуска агента через Notes Java API в CM-Sochi можно реализовать получением именованного Spring-bean. Имя бина состоит из Module.complect+ModuleType.id для Database, от куда получен агент, плюс имя агента. Все бины, реализующие агентов активируются через единый Java-интерфейс.

Есть унифицированный агент, который считывает из контекстного Document поля, указывающие какой Notes-класс подгрузить динамически и какой метод в нём активировать. Бин для такого агента должен на основании тех же параметров из полей получить имя другого бина и запустить его, передав параметры.

Для агентов «по расписанию» необходимо обеспечить возможность определения расписаний и привязку выполнения агентов к серверу приложений через АРМ администратора. Сейчас в СМ4 проектируется диспетчер задач, запускаемых по расписанию (AFServer), поэтому способ его портирования в CM-Sochi необходимо прорабатывать позже.

1. Настраиваемый набор реквизитов

В CMJ-Server часть атрибутов (реквизитов) сущности может быть настраиваемой администратором после внедрения системы. Атрибуты могут быть следующих типов:

* Строковый
* Числовой
* Дата-время
* Элемент справочника организаций (СО, СпО, СпП)
* Элемент классификатора

Атрибуты могут принимать множественное значение, либо одиночное. Атрибут типа «Элемент классификатора» может быть ссылкой на классификатор, либо заданный вручную?

Адаптер к платформе CM-Sochi должен уметь считывать конфигурацию, описывающую дополнительные атрибуты и уметь их отображать на ДОП. Бины, выполняющие мэппинг на ДОП, должны быть универсальными, зависящими от типа атрибута.

1. Хранение форматированного текста

В CM-Sochi предполагается хранение форматированного текста только в виде файла. Формат файла либо MIME, либо HTML, т.е. не RichText. Вложения при этом хранятся в отдельных фалах. Ветки кода CMJ-Server, работающие с полями типа RichText, при этом не должны исполняться.

С клиента CMJ-WebGUI форматированный текст поступает как HTML и конвертируется в MIME CM-Server’ом. Было бы удобно отрефакторить CM4, чтобы в CM-Sochi конфигурацией можно было подменить реализацию на прямое сохранение HTML в виде потока данных в файл платформы.

1. Аутентификация

Механизм аутентификации должен быть переделан на использование JAAS. Компонент аутентификации в CMJ-Server реализуется через Spring Security, позволяющей через изменение конфигурации делегировать часть процесса HTTP-аутентификации в JAAS.

1. Авторизация

CMJ-Server использует следующие контуры доступа, обеспечиваемые Domino и самим CMJ. Для доступа к нижестоящему контуру, необходимо пройти вышестоящий.

* Доступ к серверу Domino
* Наличие пользователя в текущем СО
* Доступ к БД
* Доступ к документу на чтение
* Доступ к документу на редактирование
* Доступ на редактирование к полям документа и операциям с ним

Доступ к серверу Domino – эквивалентен прохождению аутентификации платформой CM-Sochi.

Наличие пользователя в текущем СО – переиспользуется этап аутентификации CMJ-Server, проверяющий наличие пользователя в СО.

Доступ к БД – можно динамической группой, привязанной к соответствующему модулю.

Доступ к документу на чтение – в Domino задаётся наличием идентификатора пользователя, либо группы, в которую он входит, либо роли в полях типа Readers или Authors, на которую назначен пользователь.

CMJ-Server в СМ4 на основании идентификаторов пользователя, заданных в документе, ищет в СО Domino-идентификатор пользователя – «Notes-имя», также может запросить и Notes-имена замещающих или аудиторов сотрудника. Notes-имена сотрудника, его замов и аудиторов, соответствующих одной роли в документе (адресаты, подписывающий) записываются в отдельное поле с префиксом “AF$R\_”, либо “AF$A\_”. Суммированием значений таких полей автоматически формируется ещё 2 поля AFReaders и AFAuthors имеющие тип Readers и Authors, по которым Domino и выполняет авторизацию.

* 1. Моделирование авторизации доступа к документу в CM-Sochi

Адаптеры модуля СО программируются так, чтобы в качестве значений "Notes-имя" для CMJ-Server, возвращался UNID, соответствующий субъекту СО «назначение». В качестве замещающих и аудиторов должен возвращаться тот же самый UNID. Для делопроизводителей подразделений тоже должен возвращаться UNID назначений.

* + 1. Пользовательский доступ

«Пользовательским доступом» назовём случай определения состава пользователей, которым должен быть предоставлен доступ к документу, через перечисление идентификаторов пользователей в самом документе.

Для организации в CM-Sochi «Пользовательского доступа», вводим ДОП "Доступ пользователя" (Access). Он связан с корневым ДОП для документа и с ДОП субъекта СО. Кроме того, этот ДОП имеет поле «Роль в документе» (role) в качестве значения которого будем помещать имя AF$R/ AF$A поля.

Для каждого типа ДОП документа, придется иметь и 1 такой ДОП для доступа, т.к. мы не можем из него поставить связь N:1 на произвольный тип ДОП документа.



При сохранении Document, адаптер анализирует заданный набор его полей с префиксом AF$R или AF$A и удаляет/добавляет объекты ДОП-доступ (Access). Ключом для поиска является id ДОП для документа, id ДОП субъекта СО и имя AF$R/AF$A поля. Получается, что для каждого AF$R/AF$A поля будет свой набор записей Access. Привязка к имени поля необходима, т.к. при записи полей в Document CMJ-Server формирует не все поля, а только изменённые и поэтому, при чтении Document важно в исходном виде сформировать и AF$R/AF$A поля.

Для каждого типа ДОП документа должна быть задана контекстная роль, предоставляющая полный доступ (вне зависимости от статусов), и вычисляющаяся через динамическую группу "пользователь+замы+аудиторы" по списку субъектов СО, с которыми связан данный ДОП документ при помощи ДОП доступа с любым значением в поле "Роль в документе" (role).

Не использование в механизме полей AFReaders и AFAuthors делает не нужным их хранение в РБД. Но CMJ-Server изредка читает эти поля. Для таких редких операций можно предусмотреть «ленивое» их формирование по ДОП доступа.

* + 1. 3 Доступ делопроизводителям

«Доступ делопроизводителям» от «пользовательского доступа» отличается тем, что в Document для них нет списка UNID’ов сотрудников-делопроизводителей. А вместо этого есть UNID подразделения. Делопроизводители этого подразделения в СО указываются через связи с сотрудниками. Разница с «пользовательским доступом» лишь в определении контекстной роли, вычисляющейся через динамическую группу «делопроизводители модуля» по списку субъектов СО. Отметим, что для каждого типа модуля группа своя, а экземпляры для неё соответствуют подразделениям.

Но поскольку не эффективно и не удобно определять роль через фильтрацию записей ДОП "Доступ пользователя" по именам специфических полей, лучше обобщить механизм, заведя отдельный ДОП «Доступ делопроизводителя». Тогда контекстная роль «Делопроизводители» легко определяется по связям, без фильтрации.

* + 1. Наведённый доступ

Правила определения доступа к документу, зависящие от значений полей самого документа называем «собственный доступ». Правила определения доступа к документу, зависящие от значений полей других документов, называем «наведённый доступ». «Пользовательский доступ» и «доступ делопроизводителям» являются примерами «собственного» доступа.

Примерами «наведённого доступа» могут быть:

"Участники исполнения документа" – предоставления доступа к РКК исполнителям резолюций по РКК

"Участники согласования" – все визирующие документ.

"Участники ознакомления" – аналогично.

"Читатели связанных документов" – совокупность вышеперечисленных правил, применённых к связанным документам.

В CMJ-Sochi для любого такого правила можно определить еще несколько контекстных ролей, определяющих субъектов по связанным ДОП.

1. Справочники

Справочник СО и частично СпО задаётся через административного клиента CM-Sochi. Справочники СпО, СпП заполняются через CMJ-WebGUI.

Справочник СО в CM4 считывается из Domino и сохраняется в РБД. В CM-Sochi менять эту логику не предполагается. Нужны адаптеры для считывания СО из платформы CM-Sochi.

Реализация определения справочника «Каталог» и «Портал» в части профилей пользователя, тоже должна быть частью АРМ администратора.

1. Уведомления

Уведомление пользователей системы о событиях системы в СМ4 организовано через формирование записей в БД «Уведомления» и последующей отправкой e-mail пользователям на основании этих записей. За одним исключением, уведомления формируются агентами по-расписанию.

В CM-Sochi отправка e-mail будет осуществляться средствами платформы. Логика агентов должна быть переведена на язык Java и включена в CMJ-Server. При переводе, формирование уведомлений должно заключаться в отправке e-mail и прямом создании записей в ПКД. Возможность синхронной отправки уведомлений при сохранении документов зависит от производительности этих операций. Если производительность окажется не достаточной, можно задействовать «аудит» платформы для отложенной отправки уведомлений.

1. ПКД

Подсистема «Персональные каталоги данных» периодически считывает и сохраняет свои данные из Domino-БД «Уведомления». Считывание происходит в «кэш», построенный на РБД.

Для минимизации изменений не предполагается переводить хранилище ПКД на использование средств платформы, а сохранить существующий механизм объектно-реляционного отображения. Т.е. оставить за ПКД право непосредственной работы с РБД.

В CM-Sochi должны быть отключены задачи по расписанию, выполняющие синхронизацию данных ПКД с Domino-БД и обновления статусов задач по изменениям в Domino-БД,.

1. ChLog

В CM Domino-БД «Протоколы изменений» содержит записи об изменениях полей документов –«ChLog». ChLog служат для:

* Фиксации факта изменения документа – кем, когда, что.
* Сигнал для отправки уведомлений
* Сигнал для обновления доступа в документах при изменениях в СО
* Сигнал для обновления отображаемых в документах элементах справочников СО, СпО, СпП
* Сигнал для обновления «наведённого доступа»
* Сигнал для обновления статусов документов при изменениях в дочерних
* Сигнал для синхронизации атрибутов в зависимых документах

Как обеспечить эти функции CMJ-Sochi. Фиксация факта изменений будет обеспечиваться платформой в подсистеме «аудит»

Сигнал для обновления доступа в документах при изменениях в СО не нужен, также, как и для обновления «наведённого доступа», т.к. обновление обеспечивается контекстными ролями платформы.

Сигнал для обновления отображаемых в документах элементах справочников СО, СпО, СпП не нужен, т.к. извлекается непосредственно из справочников. Отображение в документах исторической информации должно обеспечиваться версионированием элементов справочника и ссылками в ДОП прикладных документов на конкретные версии.

Остаётся необходимость в:

* Сигнал для обновления статусов документов при изменениях в дочерних
* Сигнал для синхронизации атрибутов в зависимых документах

Данный функционал и вообще любой, использующий ChLog, заложен в агентах по-расписанию, а значит их необходимо переписать на Java. В рамках этой работы необходимо переориентировать логику либо на использование «аудита» платформы, либо на синхронное выполнение при поступлении сигнала.

Выходит, наличие модуля аналогичного «Протоколам изменений» в CM-Sochi не требуется. Появляется задача на уровне адаптеров или конфигураций отключить функционал создания ChLog.

1. История изменений документа

По-хорошему, историю изменений документа надо перевести на использование сервиса «аудит» платформы CM-Sochi. Но это требует значительных модификаций CMJ-Server. Предлагается сделать перевод в более поздних версиях CM-Sochi.

1. Сервис AFSaveDocUtils

Сервис AFSaveDocUtils выполняет следующие функции:

* Создание ChLog
* Формирование поле AFReaders/AFAuthors по AF$R/AF$A полям
* Синхронизация атрибутов в зависимых документах

Как было показано, первые 2 функции в CM-Sochi не востребованы. Синхронизацию атрибутов предлагается сделать на уровне адаптеров, либо организуя при считывании зависимых Document, получение атрибутов из основного. Либо запрограммировав копирование атрибутов при сохранении основного Document в зависимые.

Первый способ предпочтительней для РБД, но хорошо подходит лишь когда требуется просто считать атрибут из основного документа. Второй способ потребуется, чтобы задать алгоритм для вычисления зависимых атрибутов (замена случая задания @-формулы для вычисления синхронизируемых атрибутов в AF$ConnDocFields)

1. Полнотекстовый поиск

Для реализации предполагается задействовать сервисы платформы CM-Sochi, базирующиеся на Solar. Сервисы ещё не спроектированы, поэтому вопросы интеграции с CM-Server потребуется решать позже. Есть надежда, что гибкости конфигурирования подсистемы поиска CM-Server хватит для интеграции с сервисами платформы. Определённо можно сказать, что реализация интеграции не будет задействовать адаптеры Notes Java API к платформе CM-Sochi, а будет непосредственно взаимодействовать с сервисами платформы.

1. Список задач для рефакторинга CMJ-Server
2. Иметь возможность конфигурацией подменять NotesFactory на нашу фабрику сессий, например, через xml-дескрипторы Spring Beans (1 чел\*час) – Ноздрачёв+Иванов.
3. Перенести из web-модуля “rest” все классы в какой-нибудь нижестоящий jar-модуль (0,5 чел\*час) – Болотов.
4. Где возможно, заменить Evaluate на аналоги через Java API (8 чел\*час) – Ноздрачёв, Новикова
5. Оставшиеся операции, использующие @-формулы вынести в отдельные программные компоненты, подключаемые через конфигурацию (8 чел\*час) – Ноздрачёв, Болотов, Новикова, Иванов
6. Иметь возможность подменять имплементацию сохранения форматированного текста на прямое сохранение HTML (8 чел\*час). – Болотов
7. Доработка API ПКД в части создания/удаления уведомлений и входящих задач (0,5 чел\*час) – Левковская.
8. Возможность отключения AFSaveDocUtils (4 чел\*час) – Ноздрачёв

Итого 26 чел\*час