На платформе Alfresco ECM

Моделирование контента в Alfresco ECM и Citeck EcoS

2015

Оглавление

Что такое моделирование контента

Системы класса ECM позволяют работать с произвольным контентом: документами, изображениями, аудио- и видео-записями, а также файлами любых других типов. Помимо собственно файлов такие системы хранят определенный объем дополнительной информации (метаданных, атрибутов). В общем случае в организации свой набор типов документов (например, входящие, служебные записки, приказы, …), и у каждого типа – свой набор атрибутов (регистрационный номер, контрагент, филиал, ответственный, …). Поэтому ECM-система должна позволять добавлять новые типы данных и атрибуты.

Система Alfresco ECM позволяет делать это с помощью функции моделирования контента. Администратор Alfresco задает модели (XML-файлы), в которых определяет типы данных и их атрибуты. Атрибуты также обладают некоторыми характеристиками (обязательность, значение по умолчанию и т.п.), которые также задаются при моделировании.

Кто занимается моделированием контента

В системе EcoS уже доступны модели для широкого круга применений. Однако при внедрении системы в любую организацию случается так, что каких-то типов и атрибутов в стандартных моделях не хватает. В этом случае инженеры внедрения должны уметь определить новые типы и атрибуты, а также задать для них требуемые значения характеристик.

Стандартные возможности моделирования Alfresco

В моделях Alfresco можно определять:

- Пространства имен (`namespaces`)
- Классы объектов: типы и аспекты (`types`, `aspects`)
- Свойства и ассоциации (`properties`, `associations`)
- Ограничения (`constraints`)

Сами модели – это XML-файлы следующей структуры:

```
<model name="letters:lettersModel"
       xmlns="http://www.alfresco.org/model/dictionary/1.0">
    <description>Citeck EcoS letters</description>
    <author>Citeck</author>
    <version>1.0</version>
    <imports> … </imports>
    <namespaces> … </namespaces>
    <constraints> … </constraints>
    <types> … </types>
    <aspects> … </aspects>
</aspect>
```

Обязательными элементами модели являются `name`, `xmlns` и `namespaces`. На практике в каждой модели также есть импортированные пространства имен (`imports`), а также определения типов (`types`) или аспектов (`aspects`), для чего эта модель и предназначена.

Типы

Модель позволяет определять типы данных (`types`). Например, можно определить тип «Входящий документ». Для типа можно определить свойства (например, входящий номер) и ассоциации (например, контрагент), а также аспекты (например, классифицируемый).

Каждый объект характеризуется одним (и только одним) типом (т.е. объект не может быть одновременно входящим и приказом). Однако для типа всегда указывается родительский тип (`parent`), от которого он наследует все его свойства, ассоциации и аспекты.

В модели типы располагаются в разделе `types`. Обязательным элементом определения типа является `name`, на практике в каждом определении присутствует также `parent`, остальные элементы можно опускать. Каждое определение типа выглядит следующим образом:

```
<type name="letters:income">
    <title>Входящий</title>
    <description>Входящий документ</description>
    <parent>idocs:doc</parent>
    <properties> ... </properties>
    <associations> ... </associations>
    <overrides> ... </overrides>
    <mandatory-aspects> ... </mandatory-aspects>
</type>
```

Свойства и ассоциации

Свойства и ассоциации можно считать атрибутами типа. Различие между ними заключается в том, что свойства ссылаются на некоторый примитивный тип данных (строка, число, дата, …), а ассоциации – на типы данных, определенные в моделях (входящий, юр.лицо, пользователь, …).

Помимо типа данных в модели можно указать, является ли свойство (ассоциация) обязательным (`mandatory`), и может ли свойство (ассоциация) содержать несколько значений (`multiple`, `many`). Для свойств также можно указать значение по умолчанию (`default`), ограничения (`constraints`) и директивы управления индексацией (`index`).
Определения свойств перечисляют в разделе `properties` модели. Обязательные элементы определения свойства – `name` и `type`, остальные элементы можно не включать. Пример:

```
<property name="letters:deliveryMethod">
    <title>Метод доставки</title>
    <type>d:text</type>
    <mandatory>true</mandatory>
    <multiple>false</multiple>
    <default>express</default>
    <constraints> ... </constraints>
    <index enabled="true"> … </index>
</property>
```

Определения ассоциаций перечисляют в разделе `associations` модели. Обязательными элементами каждой ассоциации являются `name` и `target.class`. В поле `target.class` необходимо указать требуемый тип или аспект target-объекта. На практике также обычно указывают отсутствие ограничений на source-объект (`mandatory=false, many=true`).

```
<association name="letters:addressee">
    <title>Адресат</title>
    <source>
        <mandatory>false</mandatory>
        <many>true</many>
    </source>
    <target>
        <class>idocs:contractor</class>
        <mandatory>false</mandatory>
        <many>false</many>
    </target>
</association>
```

Элементы `source` и `target` в определении ассоциации означают два объекта, которые связываются. Ассоциацию можно представить как стрелку, связывающую два объекта, и тогда `source` – это откуда идет стрелка, а `target` – куда. Как видно из определения, для объекта `target` класс должен быть задан, а для `source` – нет. Это потому, что класс для объекта `source` – это тип или аспект, в котором определена эта ассоциация.

Настройки mandatory и many регулируют арность связи с каждого конца. Например:

- если mandatory = true и many = true, то арность связи: 1…*;
- если mandatory = false и many = true, то арность связи: 0…*;
- если mandatory = false и many = false, то арность связи: 0…1;
- если mandatory = true и many = false, то арность связи: 1...1.

Дочерние ассоциации

Каждая ассоциация задает связь между двумя объектами, причем эта связь может быть иерархической, а может и не быть. Те ассоциации, для которых связь иерархическая, называются дочерними (child-association), а остальные – равноправными (peer-association). В дочерних ассоциациях соединяемые объекты являются родителем и ребенком, а в равноправных – это просто исходный объект и целевой.

Дочерние ассоциации – важный элемент системы Alfresco. В частности:

- если удалить родителя, то будут удалены все его дети;
- для дочерних объектов можно контролировать уникальность имени (это можно отключить при определении child-ассоциации);
- через child-ассоциацию распространяются права доступа к объекту (это можно отключить для отдельных объектов);
- через child-ассоциацию распространяются метки времени изменения (это можно отключить при определении child-ассоциации).

В модели дочерние ассоциации располагаются в разделе `associations`, также как и обычные ассоциации, но вместо `association` пишут `child-association`:

```
<child-association name="cm:contains">
    <source>
        <mandatory>false</mandatory>
        <many>true</many>
    </source>
    <target>
        <class>sys:base</class>
        <mandatory>false</mandatory>
        <many>true</many>
    </target>
    <duplicate>false</duplicate>
    <propagateTimestamps>true</propagateTimestamps>
</child-association>
```

В случае дочерних ассоциаций объект `source` обозначает родителя, объект `target` – ребенка. Настройка `duplicate=false` позволяет включить контроль уникальности по имени, а настройка `propagateTimestamps=true` – включить распространение меток времени изменения.

Аспекты

Аспекты позволяют добавлять к объектам дополнительные свойства и ассоциации, а также другие аспекты. В этом смысле они аналогичны типам, но в отличие от типов, объект может иметь множество различных аспектов. Аспект также может обладать родительским аспектом (parent), но в отличие от типов это необязательно. Если типы обычно называются существительным (приказ, договор), то аспекты – прилагательными или глаголами (классифицируемый, имеет автора).

Определения аспектов перечисляют в разделе aspects модели. Каждое определение выглядит аналогично определению для типа:

```
<aspect name="letters:hasDelivery">
    <title>…</title>
    <description>…</description>
    <parent>…</parent>
    <properties> ... </properties>
    <associations> ... </associations>
    <overrides> ... </overrides>
    <mandatory-aspects> ... </mandatory-aspects>
</aspect>
```

Единственным обязательным элементом определения аспекта является имя. Можно даже сделать аспект, у которого есть только имя – такой аспект называется маркерным. Благодаря этому можно различать объекты по наличию или отсутствию такого маркерного аспекта. Например, в стандартной модели есть аспект `cm:personDisabled`, который налагается на объект `cm:person` (пользователь), если он отключен.

Имена и пространства имен

Каждый объект, определенный в модели – включая типы, аспекты, свойства и ассоциации, а также саму модель – должны иметь уникальное имя (`name`). Для исключения конфликтов имен в Alfresco используются пространства имен (`namespace`): каждое имя (типа, аспекта, …) состоит из двух частей: пространства имен и локального имени, при этом локальное имя должно быть уникальным только в рамках своего пространства имен. Каждая модель может определить

несколько пространств имен, и каждое имя в модели должно содержаться именно в этих пространствах имен.

Пространства имен задаются достаточно длинной строкой (URI), например: `http://www.alfresco.org/model/content/1.0`. Имена также получаются довольно длинными, например, имя типа «папка»: `{http://www.alfresco.org/model/content/1.0}folder`. Для того чтобы сократить запись используются префиксы пространств имен, например, предыдущее пространство имен имеет префикс `cm` (что означает content model), а предыдущее имя в префиксной записи более краткое: `cm:folder`.

Пространства имен определяются в секции `namespaces` следующим образом:

```
<namespace uri="http://www.citeck.ru/model/letters/1.0" prefix="letters" />
```

Если в модели используются имена из других моделей (а это обычный случай), то они подключаются аналогично в секции `imports`:

```
<import uri="http://www.alfresco.org/model/dictionary/1.0" prefix="d" />
<import uri="http://www.alfresco.org/model/content/1.0" prefix="cm" />
```

Ограничения

На значения свойств можно задавать ограничения (`constraints`). В моделях поддерживаются следующие виды ограничений:

- `LIST` – свойство должно иметь значение из списка разрешенных значений (`allowedValues`)
- `MINMAX` – свойство должно иметь значение не меньше (не больше) заданного (для числовых свойств)
- `LENGTH` – свойство должно иметь длину не меньше (не больше) заданной (для строковых свойств)
- `REGEX` – свойство должно соответствовать регулярному выражению

Пример ограничения, заданный в модели:

```
<constraint name="letters:deliveryMethod" type="LIST">
    <parameter name="allowedValues">
        <list>
            <value>snailmail</value>
            <value>courier</value>
            <value>fax</value>
            <value>express</value>
        </list>
    </parameter>
</constraint>
```

Ограничения можно задавать как внутри модели, так и непосредственно внутри свойства, значение которого ограничивается. Если ограничение задается внутри модели, тогда для него обязательно задается имя, чтобы потом можно было сослаться на это ограничение по имени:

```
<property name="letters:deliveryMethod">
    ...
    <constraints>
        <constraint ref="letters:deliveryMethod" />
    </constraints>
</property>
```

Переопределения

Переопределения позволяют уточнить характеристики свойств в дочерних типах или аспектах.

Например, если есть тип данных «документ», для которого определено свойство «номер», то в дочернем типе данных «приказ», можно сделать это свойство обязательным, указать ему значение по умолчанию или добавить на него ограничение.

Пример переопределения, делающего свойство обязательным:

```
<overrides>
    <property name="letters:deliveryMethod">
        <mandatory>true</mandatory>
    </property>
</overrides>
```

Локализация моделей

Если достаточно работы модели на одном языке, то можно использовать атрибуты `title` и `description` – их можно задать для всех именованных объектов модели: типов, аспектов, свойств и ассоциаций.

```
<type name="letters:income">
    <title>Входящий</title>
    <description>Входящий документ</description>
    ...
```

Если требуются переводы модели на несколько языков, то эти переводы располагаются в отдельных файлах с расширением `*.properties` – по одному файлу на язык перевода.

Также properties-файлы потребуются для локализации ограничений типа `LIST`.

Пример properties-файла:

```
listconstraint.letters_deliveryMethod.snailmail=Почта
letters_lettersModel.type.letters_income.title=Входящий
letters_lettersModel.property.letters_deliveryMethod.title=Метод доставки
letters_lettersModel.association.letters_addressee.title=Адресат
```

Наследование

Очень часто возникает задача сделать одно и то же свойство (ассоциацию) в нескольких разных типах. Однако механизм моделей позволяет определить каждое свойство (ассоциацию) только один раз – в каком либо типе или аспекте. Поэтому для решения такой задачи необходимо прибегнуть к наследованию.

Существует два варианта наследования свойства (ассоциации) – наследование от родительского типа (аспекта) – указывая элемент `parent` – или наследование от обязательных аспектов – `mandatory-aspects`.

Основное отличие в том, что родительский тип может быть только один, а обязательных аспектов – несколько, поэтому аспекты – более удобный способ структурирования данных.

Загрузка моделей

Действующие модели в Alfresco хранятся в оперативной памяти репозитория, поэтому для того, чтобы XML-файл с определением модели начал действовать, его необходимо загрузить в Alfresco. Для этого система предлагает два варианта:

- dictionary bootstrap – загрузка модели при старте сервера;
- dynamic models – загрузка модели при работе сервера.

Для использования подхода dictionary bootstrap необходимо:

1. положить модель (например, файл `lettersModel.xml`) в папку `tomcat/shared/classes/alfresco/extension/model`;
2. положить локализацию (например, файлы `letters*.properties`) в папку `tomcat/shared/classes/alfresco/extension/messages`;
3. создать файл `tomcat/shared/classes/alfresco/extension/lettersModel-bootstrap-context.xml` следующего, содержимого:

```
<?xml version='1.0' encoding='UTF-8'?>
<beans xmlns="http://www.springframework.org/schema/beans">
  <bean id="letters.dictionaryBootstrap" parent="dictionaryModelBootstrap">
    <property name="models">
      <list>
        <value>alfresco/extension/model/lettersModel.xml</value>
      </list>
    </property>
    <property name="labels">
      <list>
        <value>alfresco/extension/messages/letters</value>
      </list>
    </property>
  </bean>
</beans>
```

4. перезагрузить репозиторий Alfresco.

Для использования подхода dynamic models необходимо:

1. положить модель (например, файл `lettersModel.xml`) в папку `tomcat/shared/classes/alfresco/extension/model`;
2. положить локализацию (например, файлы `letters*.properties`) в папку `tomcat/shared/classes/alfresco/extension/messages`;
3. зайти по адресу http://localhost:8080/alfresco/faces/jsp/login.jsp
4. войти в систему как администратор;
5. зайти по адресу http://localhost:8080/alfresco/faces/jsp/admin/repoadmin-console.jsp
6. для загрузки модели выполнить команду:
   `deploy model alfresco/extension/model/lettersModel.xml`
7. для загрузки локализации выполнить команду:
   `deploy messages alfresco/extension/messages/letters`

8. для выгрузки модели выполнить команду:
   `undeploy model lettersModel.xml`
9. для выгрузки локализации выполнить команду:
   `undeploy messages letters`

Динамическая загрузка модели (во время работы репозитория) может быть довольно капризной. Например, из модели нельзя ничего удалять (типы, свойства и т.п.), можно только добавлять новое, а также нельзя накладывать новые ограничения (например, делать свойства или ассоциации обязательными, или пополнять список обязательных аспектов). Для того чтобы это можно было сделать, необходимо сначала выгрузить модель, а затем загрузить ее снова. Однако выгрузить модель не удастся, если в репозитории имеются объекты с типами или аспектами, указанными в этой модели – придется сначала удалить все эти объекты.

Официальная документация Alfresco

Более подробно о моделировании контента средствами Alfresco можно прочесть в официальной документации Alfresco и других ресурсах.

http://docs.alfresco.com/4.2/concepts/content-modeling-about.html

http://wiki.alfresco.com/wiki/Data_Dictionary_Guide

http://wiki.alfresco.com/wiki/Constraints

http://ecmarchitect.com/images/articles/alfresco-content/content-article.pdf

Ниже мы приводим рекомендации по моделированию на основе нашего опыта.

Правила и рекомендации

Пространства имен

- Модель может задавать несколько пространств имен.
  Хорошее разделение имен по пространствам имен способствует улучшению качества кода (его становится проще читать и т.п.)
- Разные модели не могут задавать одно и то же пространство имен
  Попытка загрузить модель, определяющую пространство имен, которое уже определено, приводит к ошибке.
- Одно пространство имен – один префикс
  В большей части системы префикс используется как заменитель пространства имен, поэтому они должны быть взаимозаменяемы. При этом используется префикс, указанный в модели, которая определяет пространство имен. Во всех остальных моделях рекомендуется использование того же самого префикса, это повышает качество кода.
- В модели можно определять имена только с пространством имен, определенным в этой же модели.
  Определение имени из импортированного пространства имен разрешается, но может привести к конфликтам, если в другой модели будет добавлен объект (например, свойство) с таким же именем.

Именование

- Имена должны содержать только латинские буквы.
- Имена типа и аспекта не могут совпадать.
- Имена типа (аспекта) и ассоциации не рекомендуется дублировать.
  В противном случае не будет работать поиск CMIS, а значит и интеграция с многими приложениями.
- Имена свойств и ассоциаций не рекомендуется дублировать
  В некоторых частях системы (например, на формах) на свойства и ассоциации ссылаются по имени, предполагая, что оно уникально для свойств и ассоциаций. Если это неявное предположение не выполняется, то это может привести к непредсказуемому поведению таких компонентов.
- Следствие из всего вышеперечисленного: все имена лучше делать уникальными. Для того чтобы достичь этого, можно использовать какую-нибудь систему при формировании имен.

Ограничения

- Не рекомендуется задавать ограничение обязательности (`mandatory`), особенно для ассоциаций.
  Если в модели задано, что ассоциация обязательна, то невозможно будет создать объект без этой ассоциации. Если же это ограничение было наложено после создания объекта (т.е. модель была обновлена), то работа с объектами без этой ассоциации может быть полностью блокирована.
- Не рекомендуется задавать ограничения на source-объект ассоциаций.
  Обычно в качестве атрибута рассматривается target-объект ассоциации. Если налагаются ограничения на source-объект, это может приводить к невозможности создания объектов с не вполне понятными сообщениями об ошибке. Отсутствие ограничений на source-объект предполагает, что `source.mandatory = false`, `source.many = true`.

Наследование

- Для типа `parent` обязателен и должен быть другим типом.
  Создание типа без родительского типа формально разрешено, но система будет плохо работать с таким объектом, вплоть до того что они не будут доступны даже администратору системы.
- Для аспекта `parent` необязателен, но если задан – должен быть другим аспектом.
- Если свойство (ассоциацию) необходимо использовать в нескольких типах, лучше вынести такое свойство (ассоциацию) в отдельный аспект и указать этот аспект как обязательный.
- Переопределения (`overrides`) работают только для свойств, унаследованных от типа
  Если необходимо сделать свойство, унаследованное от аспекта обязательным, то можно использовать функционал моделирования Citeck EcoS.

Разное

- Не стоит пытаться добавить новый примитивный тип данных (data-type).
  В настоящий момент создать объект, у которого свойство имеет нестандартный тип данных, не получится, даже если этому типу данных соответствует поддерживаемый класс Java (например, строка).

- В моделях важен порядок следования элементов. Например, в модели элементы следуют в таком порядке: imports, namespaces, constraints, types, aspects. Более подробно посмотреть порядок элементов можно в XSD-схеме моделей – в файле modelSchema.xsd. Если перепутать порядок элементов, модель не сможет быть прочитана и загружена.

Возможности моделирования в Citeck EcoS

Чего не хватает в моделях Alfresco

Модели Alfresco позволяют определять новые типы и атрибуты, но не лишены недостатков. Среди основных недостатков можно отметить следующие:

- небольшое количество характеристик атрибутов (`mandatory, multiple, protected, default, title, description, constraints`);
- характеристики можно задавать только в виде констант (`true/false`, строка, число), нельзя использовать выражения;
- неполная поддержка характеристик в ассоциациях (в частности, отсутствует возможность задать значение по умолчанию и задать ограничения);
- неполная поддержка переопределений (ассоциации переопределять нельзя, свойства можно переопределять только унаследованные от родительского типа/аспекта).

Система Citeck EcoS содержит возможности моделирования, которые исправляют все эти недостатки:

- расширенный перечень характеристик атрибутов (помимо стандартных поддерживаются также характеристики `relevant, value-title, value-description`);
- характеристики одинаково хорошо поддерживаются как в свойствах, так и в ассоциациях;
- их можно задавать в виде выражений на нескольких поддерживаемых языках;
- гибкие возможности переопределения любых характеристик свойств и ассоциаций.

Для этого в Citeck EcoS разработана функциональность *инвариантов*.

Что такое инварианты

Инвариант – это некоторое утверждение о характеристике атрибута, которое должно выполняться. Пример инварианта: «свойство Номер обязательно, если статус документа – не Новый». В этом примере атрибут – «Номер», характеристика – обязательность (mandatory), утверждение – «обязательно, если …».

С помощью инвариантов можно задавать следующие характеристики атрибутов:

- `mandatory` – обязательность – является ли атрибут обязательным
- `multiple` – множественность – может ли атрибут содержать несколько значений
- `relevant` – релевантность – имеет ли атрибут смысл (если он не имеет смысла, то он не отображается и никак не учитывается)
- `protected` – системный – если `true`, то атрибут обслуживается системой, если `false`, то пользователь может задать значение этого атрибута
- `valid` – корректность – является ли значение атрибута корректным
- `title` – название атрибута
- `description` – описание атрибута
- `value` – значение атрибута – можно задать значение атрибута явно (вычислимые значения)

- `default` – значение атрибута по умолчанию
- `options` – возможные значения атрибута
- `value-title` – название для значения атрибута
- `value-description` – описание для значения атрибута

Различные характеристики неявно связаны между собой. Например, если атрибут нерелевантный (`relevant = false`), то он всегда корректный (`valid = true`). Если атрибут множественный (`multiple = true`), то его значение (`value`) – это массив, а если единичный (`multiple = false`) – то единичное значение. Эти и некоторые другие логичные правила закодированы в Citeck EcoS для обеспечения удобства моделирования.

Помимо характеристик свойств и ассоциаций можно управлять характеристиками специальных атрибутов, таких как:

- `parent` – родительский элемент
- `parentassoc` – дочерняя ассоциация, связывающая с родительским элементом

Таким образом, можно указать где должны создаваться объекты (или где они должны создаваться по умолчанию).

Значения характеристик можно задавать на одном из поддерживаемых языков:

- `javascript`
- `freemarker`
- `criteria`
- `explicit`

Языки JavaScript и FreeMarker стандартны для Alfresco, в системе есть достаточно разнообразное API для этих языков. Язык `criteria` – язык поиска – введен в EcoS для упрощения поиска как альтернатива стандартным языкам Alfresco, используемым для поиска, таким как Lucene и FTS. Язык `explicit` – это просто явно заданные константные значения – `true`, `false`, произвольные строки, числа или списки значений.

Например, инвариант «свойство Номер обязательно, если статус документа – не Новый» удобно описать на языке JavaScript:

```
<property name="idocs:registrationNumber">
    <invariant on="mandatory" language="javascript">
        node.properties['idocs:documentStatus'] != 'new'
    </invariant>
</property>
```

На языке `javascript` удобно описывать произвольные значения, в том числе логические, числовые, объекты репозитория, а также массивы таких значений. Язык `freemarker` более удобен для задания строк, например:

```
<invariant on="value" language="freemarker">
 Договор №${node.properties["idocs:registrationNumber"]!'б/н'} с
 ${node.assocs['contracts:contractor'][0].properties['idocs:juridicalTitle']
</invariant>
```

В настоящий момент поддержка языка FreeMarker пока что сильно ограничена, поэтому рекомендуется использовать язык JavaScript.

Для описания инвариантов доступно подмножество JavaScript API и FreeMarker API:

- Node API: `nodeRef, typeShort, aspects, parent, properties, assocs, name, isSubType(), hasAspect(), hasPermission();`
- Utilities: `message()`.

Описание этих свойств и методов доступно в официальной документации Alfresco:

http://docs.alfresco.com/4.2/concepts/API-JS-intro.html

http://docs.alfresco.com/4.2/references/API-FreeMarker-intro.html

Доступные в инвариантах свойства и методы доступа позволяют делать только операции чтения, и этого вполне достаточно для того чтобы задать произвольные выражения для инвариантов.

Файлы инвариантов

Использование инвариантов не отменяет необходимости загрузки моделей, а напротив – расширяет его. Инварианты загружаются в Alfresco в виде отдельных XML-файлов, сопутствующих файлам моделей.

Файл инвариантов имеет следующую структуру:

```
<?xml version="1.0" encoding="UTF-8"?>
<invariants xmlns="http://www.citeck.ru/ecos/invariants/1.0">
    <imports> ... </imports>
    <!-- определения инвариантов -->
</invariants>
```

Вместо комментария `<!-- определения инвариантов -->` в файле перечисляются определения инвариантов в следующем виде:

```
<type name="letters:income">
    <association name="idocs:legalEntity">
        <invariant on="default" language="javascript">
            node.assocs["letters:outcome"][0]
                .assocs["idocs:legalEntity"][0]
        </invariant>
    </association>
</type>
```

В этом инварианте записано, что юр.лицо входящего документа (получатель) по умолчанию равно юр.лицу исходящего документа (отправителю).

Инварианты располагаются в определенной области действия (scope), которую можно ограничивать с помощью специальных элементов:

- `type` – ограничивает область действия инварианта типом, заданным именем (`name`);
- `aspect` – аспект, заданный именем;
- `property` – свойство, заданное именем;
- `association` – ассоциация, заданная именем;
- `properties` – все свойства с указанным типом (`type`), например, все строковые свойства, или вообще все свойства, если тип не указан;
- `associations` – все ассоциации с указанным типом, например, все ассоциации с пользователями, или вообще все ассоциации, если тип не указан.

В области действия инварианта обязательно должно быть одно (и только одно) ограничение по атрибуту (одно из `property, association, properties, associations`). Несколько ограничений по атрибуту не допускается. В области действия инварианта может быть также ограничение по классу (`type` или `aspect`), несколько ограничений по классу не поддерживается. Если ограничения по классу для инварианта нет, то он распространяется для указанных атрибутов вне зависимости от класса (типа или аспекта).

Области действия могут быть произвольно вложены друг в друга, например:

```
<type name="type1">
    <association name="assoc1">
        <!-- определения инвариантов для type1.assoc1 -->
    </association>
    <property name="prop1">
        <!-- определения инвариантов для type1.prop1 -->
    </property>
    <properties type="d:text">
        <!-- определения инвариантов для текстовых свойств в типе type1 -->
    </properties>
</type>
<property name="prop2">
    <!-- определения инвариантов для *.prop2 -->
    <type name="type2">
        <!-- определения инвариантов для type2.prop2 -->
    </type>
    <aspect name="aspect2">
        <!-- определения инвариантов для aspect2.prop2 -->
    </ aspect >
</property>
```

Все имена, указанные в атрибутах `name` и `type`, должны быть из пространств имен, перечисленных в разделе `imports`. В этом разделе перечисляют пространства имен аналогично тому, как это делается в моделях:

```
<import uri="http://www.alfresco.org/model/dictionary/1.0" prefix="d" />
<import uri="http://www.alfresco.org/model/content/1.0" prefix="cm" />
<import uri="http://www.citeck.ru/model/content/idocs/1.0" prefix="idocs" />
<import uri="http://www.citeck.ru/model/letters/1.0" prefix="letters" />
```

В самих инвариантах обязательно задается для какой характеристики этот инвариант (`on`) и на каком языке он описан (`language`). Выражение на указанном языке располагается как текст внутри элемента `invariant`.

Для описания выражений доступны следующие объекты:

- `node` – объект репозитория, для которого инвариант задает характеристику атрибута;
- `value` – текущее значение атрибута;

а также другие стандартные объекты:

- `person` – текущий пользователь;
- `companyhome` – корень репозитория;
- `userhome` – домашняя папка пользователя;
- `message` – функция для разыменования кодов локализации.

Объекты `node`, `person`, `companyhome` и `userhome` – это объекты класса Node, поэтому они представляют одинаковое API для доступа (`nodeRef`, `parent`, `properties`, `assocs`, …).

Загрузка инвариантов

Загружать инварианты можно при старте сервера через механизм bootstrap (аналогично тому, как загружаются модели). Для этого в файле `*-context.xml` нужно указать следующие строки:

```
<bean parent="invariantsDeployer">
    <property name="location"
              value="alfresco/extension/invariants/letters-invariants.xml"
/>
    <property name="priority" value="EXTEND" />
</bean>
```

В свойстве `location` указывается, где располагается XML-файл с инвариантами относительно корня classpath (папка `tomcat/shared/classes`). В свойстве `priority` указывается приоритет файла инвариантов. Приоритет позволяет переопределять существующие инварианты в файлах с более высоким приоритетом. В инвариантах поддерживаются следующие приоритеты (в порядке повышения):

- `COMMON` – наиболее общий приоритет, инварианты для широкого множества атрибутов;
- `MODULE` – приоритет модулей EcoS – инварианты для атрибутов различных типов документов;
- `EXTEND` – приоритет расширений
- `CUSTOM` – приоритет внедрений

Разработчики EcoS используют приоритеты `COMMON` и `MODULE`, инженеры внедрения EcoS – приоритеты `EXTEND` и `CUSTOM`.

Для запрета переопределения можно использовать в инварианте атрибут `final`. Однако final-инварианты также могут переопределить другие final-инварианты.

```
<invariant on="…" language="…" final="true"> ... </invariant>
```

Обработка инвариантов

Поддержка инвариантов заключается в том, что в определенных местах системы их значения вычисляются и учитываются в логике обработки документа.

Например, при сохранении документа учитываются следующие инварианты:

- value, default, multiple – для вычисления значений атрибутов;
- relevant, mandatory, valid – для проверки релевантности, обязательности и корректности.

Если в процессе сохранения выясняется, что какой-то атрибут некорректный, то сохранение запрещается.

Также инварианты поддерживаются в формах системы EcoS, при этом выражения инвариантов вычисляются по мере заполнения формы пользователем, поэтому пользователь может видеть в действии все правила, записанные в виде инвариантов.

При вычислении инвариантов для конкретного атрибута конкретного объекта неподходящие инварианты откидываются, а подходящие упорядочиваются следующим образом:

- сначала инварианты с атрибутом `final`, затем без него;
- сначала инварианты для конкретных атрибутов (`property`, `association`), затем для произвольных атрибутов (`properties`, `associations`);
- сначала инварианты с приоритетом `CUSTOM`, затем `EXTEND`, затем `MODULE`, затем `COMMON`;
- сначала инварианты, подходящие к необязательным аспектам объекта, затем к типу объекта, затем к обязательным аспектам типа, затем к родительскому типу, затем к обязательным аспектам родительского типа и т.п. и наконец – инварианты без ограничения по классам.

Язык criteria

Язык `criteria` – это язык поиска, с помощью него удобно задавать инварианты для характеристик `value`, `default`, `options` в атрибутах, содержащих ссылки на объекты репозитория (в основном, это ассоциации). Выражение на языке criteria – это поисковый запрос, который состоит из критериев поиска. Каждый критерий состоит из трех частей:

- `attribute` – имя атрибута (`name`), по которому нужно делать поиск, или специальный атрибут (см. ниже);
- `value` – искомое значение атрибута;
- `predicate` – идентификатор предиката, который осуществляет сравнение фактического значения атрибута с искомым (`value`).

Пример инварианта, заданного на языке `criteria`:

```
<type name="letters:income">
    <property name="parent">
        <invariant on="default" language="criteria">
            <criterion attribute="path" predicate="path-equals"
value="/app:company_home/st:sites/cm:letters/cm:documentLibrary/cm:income"
/>
        </invariant>
    </property>
</type>
```

В данном случае поиск осуществляется по специальному атрибуту `path`, предикат `path-equals` (равенство пути) и значение (`value`) содержит путь к папке `income` (Входящие). Таким образом, мы задаем в какой папке по умолчанию должны создаваться входящие документы.

Возможные значения предикатов зависят от типа атрибута:

- строковые (`d:text, d:mltext`):
  - `string-contains, string-starts-with, string-ends-with`
  - `string-equals, string-not-equals`
  - `string-empty, string-not-empty`
- числовые (`d:int, d:long, d:float, d:double`):
  - `number-equals, number-not-equals`
  - `number-less-than, number-less-or-equal`
  - `number-greater-than, number-greater-or-equal`
- дата и время (`d:date, d:datetime`):
  - `date-equals, date-not-equals`
  - `date-greater-or-equal, date-less-than`
- логические (`d:boolean`):
  - `boolean-empty, boolean-not-empty`
  - `boolean-true, boolean-false`
- ассоциации:
  - `assoc-contains, assoc-not-contains`
- специальный атрибут `type`:

    - `type-equals, type-not-equals`
- специальный атрибут `aspect`:
    - `aspect-equals, aspect-not-equals`
- специальный атрибут `path`:
    - `path-equals` – равенство по пути
    - `path-child` – содержится непосредственно внутри указанного пути
    - `path-descendant` – содержится внутри (на произвольной глубине вложенности)
- специальные атрибут `parent`:
    - `parent-equals, parent-not-equals`

В инварианте можно указать несколько критериев, тогда они все должны быть выполнены (т.е. используется логика И).

Также в значение можно делать вставки на языке FreeMarker, например, для свойств Тип и Вид заданы следующие инварианты:

```
<property name="tk:type">
    <invariant on="options" language="criteria">
        <criterion attribute="type" predicate="type-equals"
                   value="cm:category" />
        <criterion attribute="parent" predicate="parent-equals"
                value="workspace://SpacesStore/category-document-type-root"
/>
    </invariant>
</property>
<property name="tk:kind">
    <invariant on="options" language="criteria">
        <criterion attribute="type" predicate="type-equals"
                   value="cm:category" />
        <criterion attribute="parent" predicate="parent-equals"
                   value="${node.properties['tk:type'].nodeRef}" />
    </invariant>
    <invariant on="relevant" language="javascript">
        node.properties["tk:type"] != null
    </invariant>
</property>
```

Другими словами для свойства Тип возможные значения – это подкатегории специальной категории «category-document-root-type», а для свойства Вид – возможные значения – это подкатегории выбранного Типа. Если же тип не выбран, то свойство Вид не релевантное.