Компонентная модель OAF*/Qt4

Sergey N. Yatskevich<snc@spectrum-soft.ru>

12 июня 2012 г.

1 Назначение

Компонентная модель OAF/Qt4 предназначена для поддержки разделения сложных программ на набор слабосвязанных и динамически линкуемых модулей. Компонентом в OAF/Qt4 называется разделяемая библиотека (*.so), экспортирующая один или несколько OAF-классов[1].

Одной из важных задач данной модели, является поддержка глобального пространства имён объектов (см. раздел 1.4 “Моникеры”). Такое пространство имён позволяет хранить стандартизованные текстовые ссылки на другие объекты системы и получать по ним ссылки на соответствующие программные объекты непосредственно в процессе работы программы с автоматической инициализацией всей необходимой для функционирования этих объектов инфраструктурой.

Ключевыми особенностями компонентной модели OAF/Qt4 являются:

- выделение разработки интерфейсов, протоколов и описаний OAF-классов в самостоятельную задачу, решаемую архитектором приложения;
- реализация поиска подходящих OAF-классов по их описаниям с помощью языка запросов;
- поддержка глобального пространства имён объектов (моникеров).

Язык запросов и глобальное пространство имён объектов обеспечивают возможность создания объекта не только через явное задание его OAF-класса, но и через описание желаемой функциональности, которую он должен реализовывать. То есть по сути OAF/Qt4 является расширением RTTI языка C++, реализующим дополнительные способы создания C++ объектов, реализация которых размещена во внешних библиотеках. Наличие описаний OAF-классов позволяет реализовать загрузку компонентов и создание объектов по требованию, а не в процессе начальной загрузки.

1.1 Интерфейс OAF/Qt4

OAF/Qt4 представляет собой библиотеку, линкуемую ко всем компонентам приложения. Данная библиотека реализует следующие функции:

- *OAF::getClassInfo* - функция получения информации об OAF-классе по его CID (Class IDentity - идентификатор класса). Например:

  ```
  const IPropertyBag* cinfo = OAF::getClassInfo ("OAF/IO/CDevice:1.0");
  ```

  возвратит интерфейс к описанию OAF-класса с заданным именем;

*OAF - расшифровывается как Object Activation Framework. Это название оставлено как ссылка на использованную в качестве прототипа библиотеку BonoboActivation, ранее носившую это название

[1]Здесь и далее под OAF-классом понимается класс C++, о котором известен только набор экспортирумых им интерфейсов и протоколов взаимодействия, но неизвестны никакие детали реализации

- *OAF::query* - функция подбора OAF-классов, соответствующих заданным критериям. Например:

```
QStringList clist;
OAF::query (clist, "repo_ids.has ('IIODevice')");
```

  после выполнения запроса список *clist* будет содержать CID всех установленных в системе OAF-классов, реализующих интерфейс *IIODevice*;

- *OAF::createFromCID* - функция создание объекта по CID его OAF-класса. Например:

```
URef<IObjectUnknown> c = OAF::createFromCID ("OAF/IO/CDevice:1.0");
```

  создаст объект заданного OAF-класса. Данная функция по смыслу аналогична оператору *new* языка C++;

- *OAF::createMoniker* - функция создания моникера с заданными параметрами. Например:

```
URef<IMoniker> m = OAF::createMoniker (NULL, "file:", "file-name.ext");
```

  создаст моникер для создания объектов, обеспечивающих доступ к заданному файлу;

- *OAF::parseName* - функция преобразования глобального имени объекта в цепочку моникеров. Например:

```
URef<IMoniker> m = OAF::parseName ("file:file-name.ext");
```

  создаст моникер для создания объектов, обеспечивающих доступ к заданному файлу;

- *OAF::unparseName* - функция восстановления глобального имени объекта по заданной цепочке моникеров. Например:

```
QString name = OAF::unparseName (m);
```

  восстановит имя *file:file-name.ext* для моникера из предыдущего примера;

- *OAF::createFromName* - шаблонная функция создания объекта, реализующего заданный интерфейс, по его глобальному имени. Например:

```
URef<IIODevice> c = OAF::createFromName<IIODevice> ("file:file-name.ext");
```

  создаст объект с интерфейсом *IIODevice*, обеспечивающий доступ к заданному файлу.

- *OAF::unloadUnusedLibraries* - функция выгрузки неиспользуемых компонентов;
- *OAF::unloadClassInfo* - функция выгрузки описаний OAF-классов;
- *IFunctionFactory* OAF::functionFactory ()* - фабрика стандартных функций языка запросов.

1.2 Описание OAF-классов

Описания OAF-классов хранятся в виде набора XML-файлов, которые должны иметь расширение *oaf*. Минимальным вариантом такого файла является:

```
<?xml version='1.0' encoding='UTF-8'?>
<oaf:info xmlns:oaf="http://www.spectrum-soft.ru/files/specs/oaf-syntax.dtd">
    <oaf:class cid="OAF/IO/CFactory:1.0" type="dll" location="/usr/lib/oaf-qt4/liboaf-io-1.0"/>
</oaf:info>
```

В случае, если библиотека компонента расположена в том же каталоге, что и соответствующий *oaf* файл, путь к библиотеке можно не указывать:

```
<?xml version='1.0' encoding='UTF-8'?>
<oaf:info xmlns:oaf="http://www.spectrum-soft.ru/files/specs/oaf-syntax.dtd">
    <oaf:class cid="OAF/IO/CFactory:1.0" type="dll" location="liboaf-io-1.0"/>
</oaf:info>
```

Стандартным каталогом, который сканируется по умолчанию является */usr/lib/oaf-qt4*. В случае необходимости дополнительные пути поиска *oaf* файлов можно указать с помощью переменной окружения *OAFPATH*. Отдельные каталоги в этой переменной разделяются с помощью символа ':'.

1.2.1 Тэги *oaf:info* и *oaf:class*

Тэг *oaf:info* является корнем всех описаний OAF-классов. Каждый тэг *oaf:class* описывает один OAF-класс. Этот тэг имеет три обязательных XML-атрибута:

- *cid* - уникальный идентификатор класса (Class IDentity). В принципе это любая строка, однако для единообразия предлагается использовать следующий формат: <пространство имён класса>/<имя класса, совпадающее с именем соответствующего C++ класса>:<версия реализации>;
- *type* - задаёт один из двух способов создания объектов данного OAF-класса:
  - *dll* - для создания объекта нужно загрузить заданную в location динамическую библиотеку и вызвать у неё функцию *URef<IObjectUnknown> createObject(const QString& _cid)*;
  - *factory* - для создания объекта нужно использовать интерфейс *IGenericFactory* объекта с идентификатором класса *location* (такой способ удобен если нужно где-то хранить информацию, разделяемую между всеми объектами, создаваемыми с помощью данной фабрики. Это может быть, например, кэш всех созданных объектов чтобы вместо повторного создания возвращать ссылки на подходящие из уже созданных);
- *location* - определяется значением атрибута *type*.

Например OAF-класс объектов, создаваемых с помощью фабрики можно описать так:

```
<?xml version='1.0' encoding='UTF-8'?>
<oaf:info xmlns:oaf="http://www.spectrum-soft.ru/files/specs/oaf-syntax.dtd">
    <oaf:class cid="OAF/IO/CFactory:1.0" type="dll" location="liboaf-io-1.0"/>

    <oaf:class cid="OAF/IO/CDevice:1.0" type="factory" location="OAF/IO/CFactory:1.0"/>
</oaf:info>
```

1.2.2 Тэг *oaf:attribute*

Полноценное использование OAF/Qt4 возможно только при использовании атрибутов в описании OAF-классов (в противном случае создание объектов можно будет выполнить только с помощью задания идентификатора OAF-класса, что фактически свяжет компоненты друг с другом ровно настолько, насколько их связало бы прямое использование C++-классов друг друга). Например:

```
<?xml version='1.0' encoding='UTF-8'?>
<oaf:info xmlns:oaf="http://www.spectrum-soft.ru/files/specs/oaf-syntax.dtd">
    <oaf:class cid="OAF/IO/CFactory:1.0" type="dll" location="liboaf-io-1.0">
        <oaf:attribute name="repo_ids" type="stringv">
            <oaf:item value="IObjectUnknown"/>
            <oaf:item value="IGenericFactory"/>
        </oaf:attribute>
        <oaf:attribute name="name" type="string" value="I/O objects factory"/>
        <oaf:attribute name="description" type="string" value="I/O objects factory"/>
    </oaf:class>

    <oaf:class cid="OAF/IO/CDevice:1.0" type="factory" location="OAF/IO/CFactory:1.0">
        <oaf:attribute name="repo_ids" type="stringv">
            <oaf:item value="IObjectUnknown"/>
            <oaf:item value="IIODevice"/>
        </oaf:attribute>
        <oaf:attribute name="name" type="string" value="I/O device"/>
        <oaf:attribute name="description" type="string" value="I/O device"/>
    </oaf:class>

    <oaf:class cid="OAF/IO/CIOMoniker:1.0" type="factory" location="OAF/IO/CFactory:1.0">
        <oaf:attribute name="repo_ids" type="stringv">
            <oaf:item value="IObjectUnknown"/>
            <oaf:item value="IMoniker"/>
        </oaf:attribute>
        <oaf:attribute name="name" type="string" value="I/O moniker"/>
        <oaf:attribute name="description" type="string" value="I/O moniker"/>
```

```
        <oaf:attribute name="monikers" type="stringv">
            <oaf:item value="file:"/>
        </oaf:attribute>
    </oaf:class>
</oaf:info>
```

Для каждого OAF-класса может быть определено множество атрибутов. Каждый атрибут имеет обязательные XML-атрибуты *name* и *type*. Возможные типы атрибутов определены в описании языка запросов OAF/Qt4. Имена типов должны быть записаны в нижнем регистре. Значения атрибутов всех типов, кроме *stringv*, записываются в XML-атрибуте *value*. Значения типа *stringv* должно быть записано как набор *oaf:item*:

```
<oaf:attribute name="repo_ids" type="stringv">
    <oaf:item value="IObjectUnknown"/>
    <oaf:item value="IIODevice"/>
</oaf:attribute>
```

Смысл атрибутов с тем или иным именем определяется архитектором приложения. Например на уровне самой OAF/Qt4 определены следующие атрибуты:

- *repo_ids*(*stringv*) - перечень всех интерфейсов, предоставляемых объектами данного класса;
- *description*(*string*) - понятное человеку полное описание OAF-класса;
- *name*(*string*) - понятное человеку короткое описание OAF-класса (например для использования в качестве строки меню);
- *mime_types(stringv)* - перечень mime-типов, которые может обрабатывать объект данного OAF-класса;
- *monikers*(*stringv*) - перечень префиксов имён, которые обрабатывает данный моникер.

1.3 Язык запросов OAF/Qt4

OAF/Qt4 поддерживает язык запросов, который может быть использован для отбора OAF-классов, соответствующих заданным критериям. Запрос на этом языке передаётся в параметре *_req* при вызове функции *OAF::query*. Запрос вычисляется на основании атрибутов, заданных в описаниях OAF-классов. Например запрос:

```
repo_ids.has('IMoniker') AND monikers.has('file:')
```

отберёт CID всех OAF-классов, которые поддерживают заданный моникер (то есть OAF-класс, который экспортирует *IMoniker* интерфейс и обрабатывает имена с заданным префиксом).

Язык запросов поддерживает так же сортировку и отбор нескольких записей из общего результата с помощью выражений, похожих на SQL. Например:

```
priority.defined() ORDER BY priority DESC, name LIMIT 2
```

отберёт CID всех OAF-классов, в которых определён атрибут *priority*, упорядочит CID по уменьшению значения атрибута *priority* и увеличению (в лексикографическом порядке) значения атрибута *name* и выдаст в качестве результата первые два CID.

Формальное определение запроса:

```
<запрос>: <выражение отбора>
            [ORDER BY <список выражений сортировки>]
            [LIMIT <выражение ограничения>]

список выражений сортировки: <выражение> [ASC|DESC]
список выражений сортировки: <выражение> [ASC|DESC], <список выражений сортировки>
```

1.3.1 Типы данных

Язык OAF/Qt4 поддерживает следующие типы данных в описании OAF-класса:

- *string* - произвольная строка (константы данного типа задаются как 'строка');
- *stringv* - список строк (константы данного типа задаются как ['строка1','строка2', ...]);
- *boolean* - булевское число (константы данного типа задаются с помощью ключевых слов TRUE/YES и FALSE/NO);
- *integer* - 64-х битное целое число (константы данного типа - десятичные целые числа);
- *double* - число с плавающей точкой (константы данного типа - десятичные числа с плавающей точкой в фиксированном или инженерном форматах);
- *null* - неопределённое значение.

1.3.2 Функции

В языке запросов OAF/Qt4 определены следующие функции:

- *defined(<выражение>)* - возвращает TRUE если заданное выражение определено для текущего описания. Например если <выражение> представляет собой идентификатор атрибута, то результат выражения будет показывать определён или нет данный атрибут в описании OAF-класса;
- *has(stringv v, string s)* - возвращает TRUE если в списке строк v присутствует строка s;
- *has_ one(stringv v1, stringv v2)* - возвращает TRUE если хотя бы одна из строк из списка v2 присутствует в списке v1;
- *has_ all(stringv v1, stringv v2)* - возвращает TRUE если все строки списка v2 присутствуют в списке v1;
- *prefer(string s, stringv v)* - возвращает -1, если строка s не присутствует в списке v. В противном случае возвращает номер строки в списке вычисленный относительно конца списка. Таким образом чем ближе строка к началу списка, тем больше это число. Эта функция предназначена для использования в выражениях сортировки;
- *if(v1, v2, v3)* - возвращает результат вычисления выражения v2 если результат вычисления выражение v1 равен true и результат вычисления выражения v3, если false. Если результат вычисления выражения v1 не определён, то возвращается NULL;
- *ifnull(v1, ...)* - возвращает первый слева на право отличный от NULL результат вычисления выражения в списке.

Имена функций нечувствительны к регистру. Для атрибутов определены два эквивалентных способа вызова функций:

- *funcname (<attribute_ name>, <список доп.аргументов>)* - функциональный стиль;
- *<attribute_ name>.funcname (<список доп.аргументов>)* - объектный стиль.

1.3.3 Операторы

В языке запросов OAF/Qt4 используются операторы, похожие на SQL. Этот выбор сделан для того, чтобы в них не встречались специальные знаки (например ! или &), которые могут использоваться для других целей. Таким образом запросы на языке OAF/Qt4 могут произвольно смешиваться с URL или другими формами моникеров (например - !) не усложняя разбор моникеров на составные части.

- = (равно, аналогичная функция имеет имя *EQ*);
- <> (не равно, аналогичная функция имеет имя *NE*);
- < (меньше, аналогичная функция имеет имя *LT*);
- > (больше, аналогичная функция имеет имя *GT*);
- <= (меньше или равно, аналогичная функция имеет имя *LE*);
- >= (больше или равно, аналогичная функция имеет имя *GE*);
- *AND* (логическое 'И', аналогичная функция имеет имя *AND* и принимает на вход два и более аргументов);
- *OR* (логическое 'ИЛИ', аналогичная функция имеет имя *OR* и принимает на вход два и более аргументов);
- *XOR* (исключающее 'ИЛИ', аналогичная функция имеет имя *XOR* и принимает на вход два и более аргументов);
- *NOT* (отрицание, аналогичная функция имеет имя *NOT*);
- / (разделить, аналогичная функция имеет имя *DIV*);
- + (сложить, аналогичная функция имеет имя *ADD*);
- - (вычесть, аналогичная функция имеет имя *SUB*);
- * (умножить, аналогичная функция имеет имя *MUL*);
- - (минус в значении унарного арифметического оператора, аналогичная функция имеет имя *NEG*).

1.4 Глобальное пространство имён объектов (моникеры)

Моникеры (moniker - кличка) - это способ именования, который позволяет реализовать глобальное пространство имён объектов. Создание моникера возможно либо вручную, либо из его строкового представления. Приведём примеры строкового представления моникеров и их интерпретацию:

- *cid:OAF/IO/CFactory:1.0* - фабрика объектов соответствующего OAF-класса;
- *cid:OAF/IO/CFactory:1.0#new:OAF/IO/CDeviceFile:1.0* - объект OAF-класса *OAF/IO/CDeviceFile:1.0*, созданный с помощью заданной фабрики;
- *file:/tmp/a.gz* - локальный файл;
- *file:/tmp/a.gz#gunzip:* - распакованный локальный файл из предыдущего примера.

Для создания объекта с помощью строкового представления моникера используется вызов:

```
URef<IIODevice> d = OAF::createFromName<IIODevice> ("file:file-name.properties");
```

Важной особеностью моникера является возможность создания цепочек объектов в зависимости от того, какой интерфейс требуется. Например:

```
URef<IPropertyBag> p = OAF:createFromName<IPropertyBag> ("file:file-name.properties");
```

создаст объект, представляющий заданный файл как список атрибутов. При какая именно цепочка объектов будет построена зависит от требуемого интерфейса и типа файла.

1.4.1 Создание объектов с помощью строкового представления моникеров

Сначала строковое представления моникера разбивается на части по опорным символам '!' и '#'. При этом символ '!' считается именем моникера, а символ '#' - просто разделителем. Каждая из таких частей должна состоять из префикса и, возможно пустого, суффикса, которые разделены символом ':' (при этом символ ':' считается частью префикса, а префиксом для моникера с именем '!' считается сам этот символ). Префикс является собственно именем моникера, а суффикс - набором параметров для создаваемого данным экземпляром моникера объекта. Цепочка моникеров создаётся начиная с самого левого элемента. Создание цепочки состоит из следующих шагов:

1. ищется и создаётся объект самого моникера с помощью запроса к подсистеме OAF/Qt4:
   `repo_ids.has('IMoniker') AND monikers.has(<prefix>) ORDER BY ifnull(priority, 0) LIMIT 1`
2. для созданного моникера задаются ранее созданный левый моникер и набор параметров:
   `moniker->set(left, <префикс>, <суффикс>);`

В качестве результата процедуры создания цепочки моникеров возвращается самый правый моникер цепочки.

Для создания цепочки объектов используется метод *IMoniker::resolve*. Моникеры по цепочке запрашивают друг у друга создание объектов с нужными им интерфейсами, связывают их и как результат работы возвращают составной объект, обладающий запрошенным интерфейсом.

1.4.2 Специальный моникер '!'

Данный моникер имеет своё собственное представление исключительно из соображений удобства записи и из-за его широкого использования. Он служит для создания объектов с заданными параметрами. Аргументом, описывающим параметры создания объекта, служит часть строкового представления моникера от символа '!' до следующего символа '!', '#' или конца строки. В процессе создания объектов данный моникер обращается к моникеру, стоящему от него слева для получения объекта, поддерживающего интерфейс *IObjectCollection* и, с помощью вызова его метода *IObjectCollection::getObject* с передачей ему строки аргументов, получает от него объект с нужными характеристиками. При этом объект, реализующий интерфейс *IObjectCollection*, сам определяет способ разбора строки аргументов и создания соответствующего объекта. Он может вернуть новый объект или существующий объект с таким же характеристиками или ссылку на себя самого, но с изменёнными в соответствии с аргументами параметрами.

2 Стандартные интерфейсы

TBD. Здесь должно быть описание всех интерфейсов, поставляемых с библиотекой OAF.

3 Стандартные реализации

TBD. Здесь должно быть описание всех реализаций C++-классов, поставляемых с библиотекой OAF.

4 Стандартные компоненты

TBD. Здесь должно быть описание всех компонентов, поставляемых с библиотекой OAF, включая детали их реализации.