Социальные API и протоколы

Рецепты социальных блюд, персональная публикация в Internet

Александр Загацкий

Социальные API и протоколы : Рецепты социальных блюд, персональная публикация в Internet

Александр Загацкий

Аннотация

Данная книга содержит примеры работы с технологиями и API социальных сервисов.

Содержание

Список таблиц

Об этой книге

По итогам моего опыта присутствия в социальных сетях у меня сформировалось стойкое убеждение, что не может быть среди них самой лучшей, самой безопасной или надежной . Даже если эти параметры близки к идеальным, проект является продуктом какой-либо компании, а та в свою очередь может изменить свою политику в отношении продукта, быть куплена другой компанией или даже может разориться.

После "судного дня" (авария в датацентре Amazon EC2) у меня появилось стойкое желание вынести центр своей активности из внешних сервисов, сохранив при этом мое присутствие в них и активность.

Очевидное решение - децентрализация моей сетевой активности. В этом направлении существует достаточное количество проектов: Diaspora, SatusNet, Friendika, OneSocialWeb. Данные проекты относятся к группе децентрализованных социальных сетей. Почти все они открыты и некоторые можно использовать в качестве альтернативы коммерческим аналогам. Например, платформу микроблоггинга StatusNet можно установить на собственном сервере и домене.

Тем не менее, для меня в первую очередь представляют интерес принципы их функционирования и протоколы взаимодействия.

Данная книга является скорее инструментом для меня самого. С ее помощью я рассчитываю систематизировать знания по интересующей меня теме и заодно сделать их доступными всем.

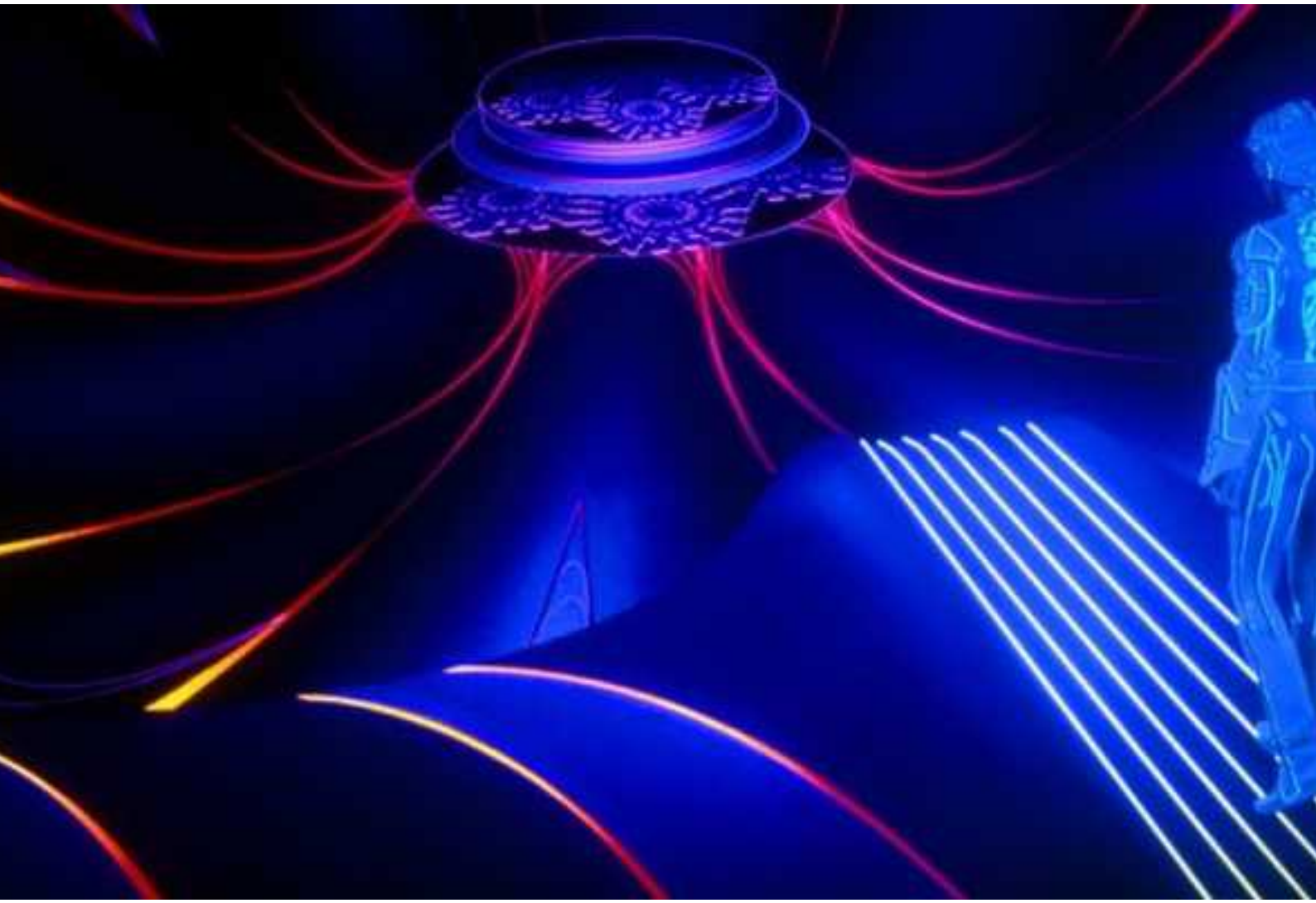
Александр Загацкий

Глава 1. Социальный робот

Два робота

У меня есть парочка роботов, которые очень помогают в жизни. С одним из них, я знаком несколько лет, а второй появился только в прошлом году. Один из них простой и надежный, а второй - сложный и его иногда приходится ремонтировать. Они являются примерами роботов разных поколений и сложности.

Робот для извлечения структур данных



Его я создал несколько лет назад. Его предназначением является обработка потока данных, идущего от некоего центрального сервиса. Выполняет он небольшую, но важную работу: в потоке ищет структуры данных и затем передает их на дальнейшую обработку.

Вскоре после его создания, мы с ним расстались. Я стал разрабатывать новые программы, а его передал вместе с остальной частью системы в управление отдельной команде разработки. До меня доходили слухи, что его разбирали, пытаясь понять как он работает, но в любом виде он справлялся со своей работой.

Я снова встретил его и восстановил в исходном виде. Выглядит он следующим образом:

```
                    #I, Robot#
my $reg_pat =           qr^
                 <(   [ \w  ]+ )>
                       ###
                      (.*?)
                      #####
                     < \/\1>
                     #     #
                     #     #
                     #     #
                     ^xs;  #xs;
#...

my %attr = $vals =~ m/$reg_pat/g; #Run robot, run !
```

Социальный робот

Еще одного робота [1] я создал, в прошлом году. Мне необходима была помощь в освещении хода конференции по свободному ПО [2], и поэтому получившийся робот обладает рядом полезных свойств:

- обучен работе в социальных сетях Twitter и FriendFeed;
- умеет отличить обычного пользователя группы FriendFeed от администратора;
- особое внимание уделяет записям в группе, которые понравились администраторам;
- местообитание этого робота - группа в FriendFeed [3].

На данный момент он требует починки, и я собираюсь перебрать его внутренности, чтобы запустить снова.

Предпосылки создания социального робота

Прежде чем перейти к структуре робота, я хотел бы рассказать вкраце о задачах, которые он помогает решать.

Самые необходимые вещи при освещении, пожалуй, любого мероприятия, будь то конференция или workshop - это :

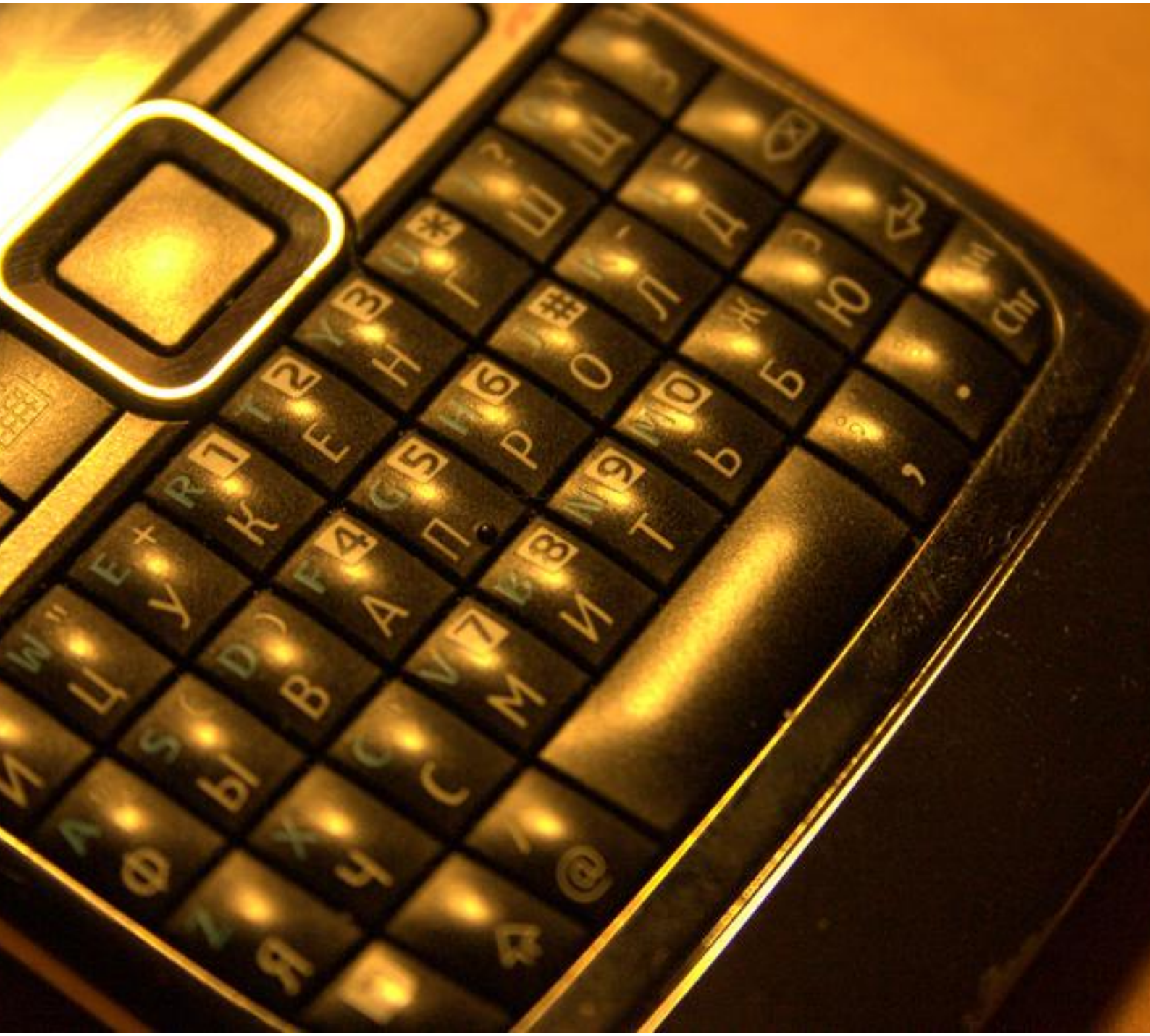
- простота публикации заметок о ходе конференции
- распространение этой информации в социальных сетях. В моем случае приоритетной социальной сетью являлся twitter [4].

Каждый из этих моментов выглядит следующим образом.

[1] Робот, обученный работе в FriendFeed. lveeboteg [http://friendfeed.com/lveeboteg]
[2] Международная конференция разработчиков и пользователей свободного программного обеспечения. http://lvee.org/
[3] Конференция Lvee в FriendFeed. http://friendfeed.com/lvee/
[4] Официальный twitter конференции Lvee.org. http://twitter.com/lveecon

Простота публикации

На конференции я всегда с собой беру блокнот и ручку. Из других подручных средств - мобильный телефон, который используется уже давно как телетайп [5] с атавизмом в виде функций голосовой связи. В моем случае это Nokia e71.



За почти трех летний срок у меня устоялся следующий набор способов публикации:

Почта Встроенный почтовый клиент полностью выполняет свои функции.

[5]ТЕЛЕТАЙП (от теле... и англ. type - писать на машинке), приемно-передающий буквопечатающий телеграфный аппарат с клавиатурой, как у пишущей машинки. http://www.slovopedia.com/2/210/266635.html

Jabber клиент — Jabber (XMPP) - открытый протокол обмена сообщениями, который присутствует во многих социальных сетях. В качестве Jabber клиента я использую talkonaut [http://www.talkonaut.ru/].

Web клиент — Для просмотра web установлен браузер Opera Mobile.

Указанные три способа публикаций:

- являются наиболее распространенными и присутствуют на любой мобильной платформе;
- базируются на открытых протоколах и достаточно библиотек, для их поддержки на серверной стороне.

Итак инструментарий для публикации имеется.

Остается сервис, который публикует сообщения, отправленные одним из указанных способов и распространяет информацию в социальных сетях. В свое время я остановился на уже существующем сервисе friendfeed.com [http://friendfeed.com]. Публикация в нем возможна всеми из доступных мне способов: с помощью email, jabber сообщений и web интерфейса. К тому же friendfeed обладает замечательным API, благодаря которому можно расширить его возможности.

Распространение информации

Вся информация о конференции, отправляемая с мобильного телефона, публикуется в открытой группе friendfeed [6]. Это позволяет другим пользователям FriendFeed также размещать в группе свою информацию, комментировать или отмечать понравившиеся заметки. Туда же импортируется RSS поток новостей с официального сайта конференции *(встроенными средствами FriendFeed)*.

Итак группа конференции в FriedFeed является основным агрегатором потока информации о мероприятии. Далее необходимо связать официальный twitter аккаунт конференции с этой группой. Именно этот функционал реализует рассматриваемый социальный робот.

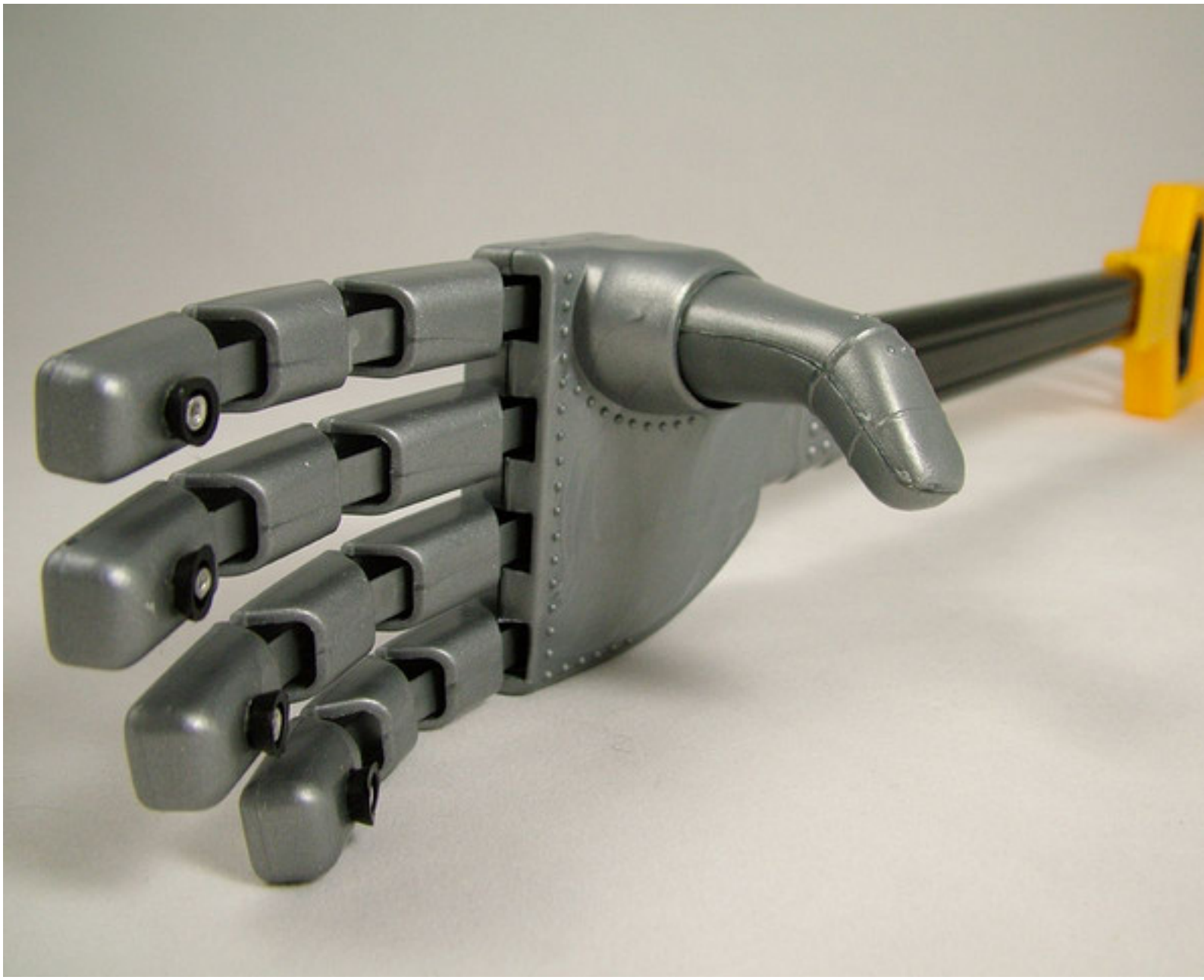
Flow - потоковая обработка данных

В разработке используются в основном 2 приема обработки данных: либо загрузка всех данных в оперативную память, либо обработка небольшими порциями по мере чтения. Первый способ прост и годится для небольших объемов, второй - сложнее, но является единственным решением при обработке больших объемов данных.

Оба подхода к обработке данных можно встретить, например, при обработке XML. К первому типу относится библиотека `XML::LibXML`. К потоковым относятся `XML::Parser`, `XML::SAX`, `XML::ExtOn`. На основе потоковой библиотеки `XML::ExtOn` построена, например, реализация `Perl6::Pod`.

Однако все больше приходится использовать потоковую обработку не только в отношении XML. И тут такие сущности, как пространства имен и тэги оказываются лишними. Для подобных случаев я создал библиотеку Flow [7]. По сути `FLow` - это `XML::SAX::Machines` [8] в максимально упрощенном виде: выброшен XML и SAX, а осталась лишь идея композиции - Machines *(то, что требуется для построения роботов!)*.

[6] Группа в социальной сети FriendFeed, посвященной конференции Ivee.org [http://Ivee.org]. http://friendfeed.com/Ivee
[7] Flow - библиотека для обработки потока данных. http://search.cpan.org/perldoc?Flow
[8] Introducing XML::SAX::Machines. http://www.xml.com/pub/a/2002/02/13/sax-machines.html



Простое должно оставаться простым

Концептуально библиотека `Flow` похожа на набор привычных утилит: grep, tail, head, которые используются ежедневно в командной строке. Они ориентированы на построчную *потоковую* обработку и будучи объединенными в *pipe* реализуют необходимую логику. Например:

```
cat test_file.txt | grep MyLine
```

Данная команда выведет строки из файла, содержащие "MyLine".

Идея компоновки простых инструментов в "pipe" используется в Flow. Указанный ниже код реализует аналогичную логику:

```
use Flow;
my $f = create_flow(
Grep=>qr/MyLine/,
```

```
                sub {print @_; \@_}
);
 open FH, "<test_file.txt";
 $p = $f->parser;
 $p->begin;
 while (my $str = <FH> ) {
  $p->flow($str)
 }
 $p->end;
```

Особенностью `Flow`, является то, что процесс обработки потока позволяет применять логику к буферу, содержащим несколько элементов данных.

```
 my $f = create_flow( Splice=>20, Grep=>qr/MyLine/)
```

Благодаря этому достигается компромисс в виде блочной обработки данных. Это позволяет снизить процессорные затраты на вызовы методов. К тому же варьируя размер блока данных (`Splice`) можно регулировать использование памяти.

Пример использования библиотеки Flow

Чтобы продемонстрировать возможности библиотеки, рассмотрим небольшую задачу. Пусть нам необходимо найти произведение чисел, хранящихся в файле. Содержимое файла следующее:

```
1
2
0
3
1
5
```

Таким образом имеем простой файл, в каждой строке которого находится число. Код, решающий эту задачу, выглядит следующим образом:

```
 use Flow;
 my $res = 1;
 my $f = create_flow(
           #Печатаем каждую прочитанную строку
    sub { print "process:",@_,"\n" ;\@_},
           #Результат произведения накапливаем в
           # переменной $res
    sub { $res *= $_ for @_}
);
 open FH, "<test_file.txt";
 $p = $f->parser;
 $p->begin;
 while (my $str = <FH> ) {
  $p->flow($str)
 }
 $p->end;
 # вывод результата
 print "$res \n";
```

Результат работы скрипта следующий:

```
process:1
```

```
process:2
process:0
process:3
process:1
process:5
result: 0
```

Результат произведения чисел равен 0. Для этого была прочитана каждая строка файла и произведено действие умножения.

В приведенном примере обработка завершиться как только будет достигнут конец файла. Для нашей задачи результат становится очевидным, если будет встречен 0, так как если среди множителей есть 0, то и произведение будет равно 0. Дальнейшая выборка чисел и обработка не нужна.

Особенностью `Flow` является обратная связь, существующая между циклом формирования потока данных и элементами, обрабатывающими этот поток. Возможна ситуация, когда дальнейшая обработка не имеет смысла. Например, необходимая информация была найдена или встречены данные, при которых продолжение вычислений теряет смысл *(как в рассмотренном примере)*.

Реализуется данный механизм исходя из следующего соглашения. Если блок кода или метод объекта в последовательности обработчиков возвращает `undef` или ссылку на массив, то производится дальнейшая обработка данных. Иначе данный результат рассматривается как особый и возвращается в инициирующий поток цикл. В нашем примере таким циклом является построчное чтение из файла.

С учетом описанных особенностей модифицируем код примера:

```
  use Flow;
  my $res = 1;
  my $f = create_flow(
     sub { print "process:",@_,"\n" ;\@_},
     sub {
  for (@_) {
    #Возвращаем 1 как признак особого случая
  $res *= $_  or return 1
   }
  \@_ # передача данных далее "по цепочке"
  }
 );
  open FH, "<test_file.txt";
  $p = $f->parser;
  $p->begin;
  while (my $str = <FH> ) {
  # прекращаем дальнейшую обработку
   last if $p->flow($str)
  }
  $p->end;
  # вывод результата
  print "$res \n";
```

Результат будет следующим:

```
process:1
process:2
```

```
process:0
result: 0
```

Как видим возможно управлять процессом формирования потока данных из компонент, обрабатывающих сам поток данных.

Именно наличие подобной управляющей обратной связи [9], позволяет достигать оптимальных результатов при обработке данных.

Данный пример прост, но общее понимание принципа работы библиотеки `Flow` поможет при ее использовании в более сложных задачах.

Структура социального робота

Социальный робот реализован на Perl 5. Основная его задача - поддерживать обмен сообщениями между группой в социальной сети FriendFeed и аккаунтом конференции в twitter.

Правила, которые он реализует следующие:

- экспортировать в twitter сообщения, которые опубликованы определенными пользователями FriendFeed или администраторами группы;
- экспортировать в twitter записи, отмеченные like администраторами группы;
- импортировать в группу friendfeed новые сообщения из аккаунта twitter.

И кончно же он соблюдает обязательные законы робототехники [10]

Структура робота

Общую последовательность действий при обмене сообщениями можно представить в виде следующих шагов:

- получаем записи из friendfeed и twitter;
- пропускаем ранее опубликованные записи;
- публикуем оставшиеся сообщения в необходимом сервисе;
- запоминаем новые опубликованные сообщения.

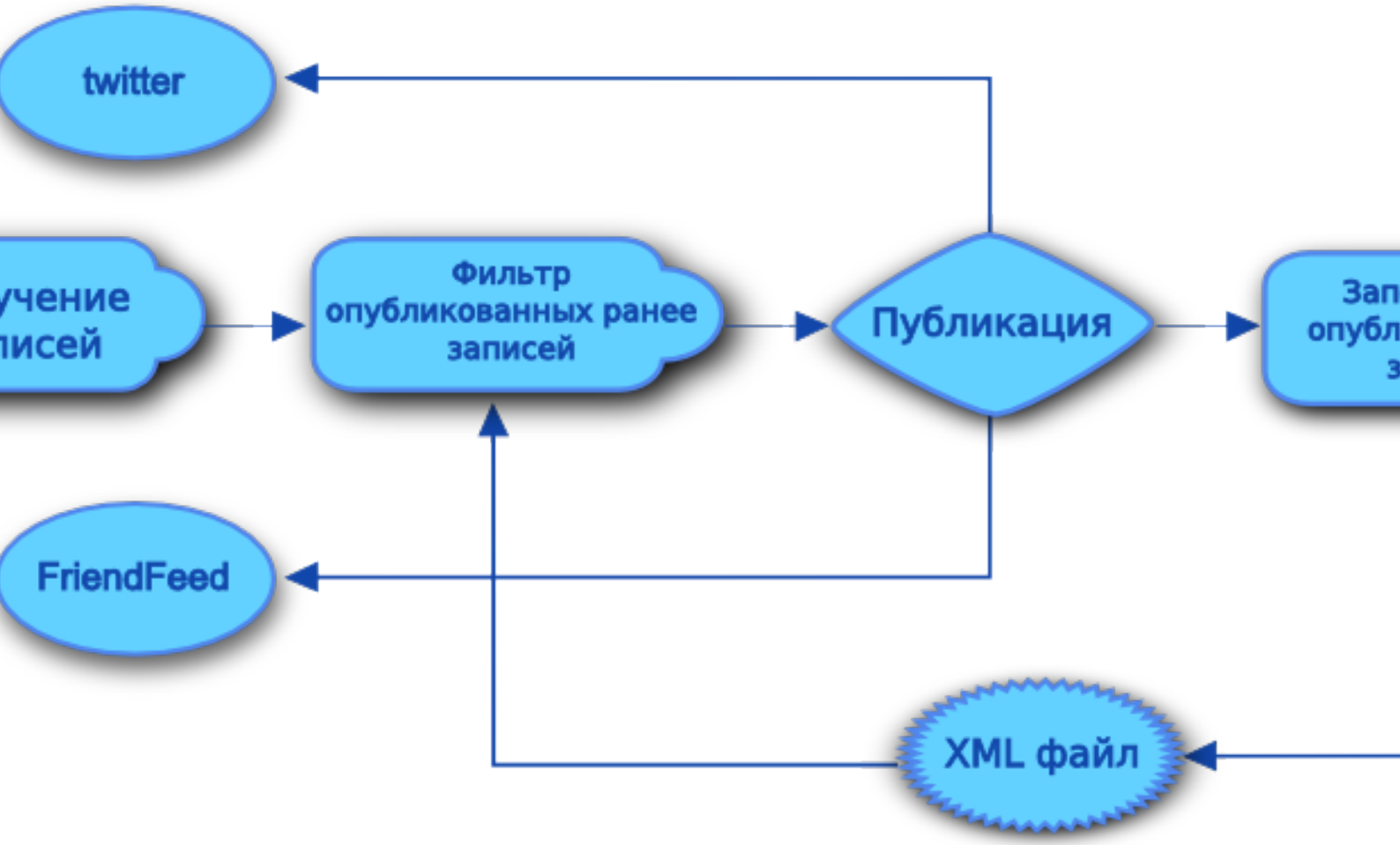
Как видно, нам необходимо:

- знать как получить список сообщений из обоих социальных сетей;
- потребуется где-то хранить информацию о ранее опубликованных сообщениях
- уметь публиковать новые сообщения из группы friendfeed в twitter и наоборот.

Более близкая к реализации схема выглядит следующим образом:

[9] Обратная связь. http://ru.wikipedia.org/wiki/Обратная_связь [http://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D0%B0%D1%8F_%D1%81%D0%B2%D1%8F%D0%B7%D1%8C]. Наиболее полную информацию об обратных связях можно почерпнуть из курса по "Автоматизированным Системам Управления".

[10] Три закона робототехники. http://ru.wikipedia.org/wiki/Три_закона_роботехники [http://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B8_%D0%B7%D0%B0%D0%BA%D0%BE%D0%BD%D0%B0_%D1%80%D0%BE%D0%B1%D0%BE%D1%82%D0%B5%D1%85%D0%BD%D0%B8%D0%BA%D0%B8]



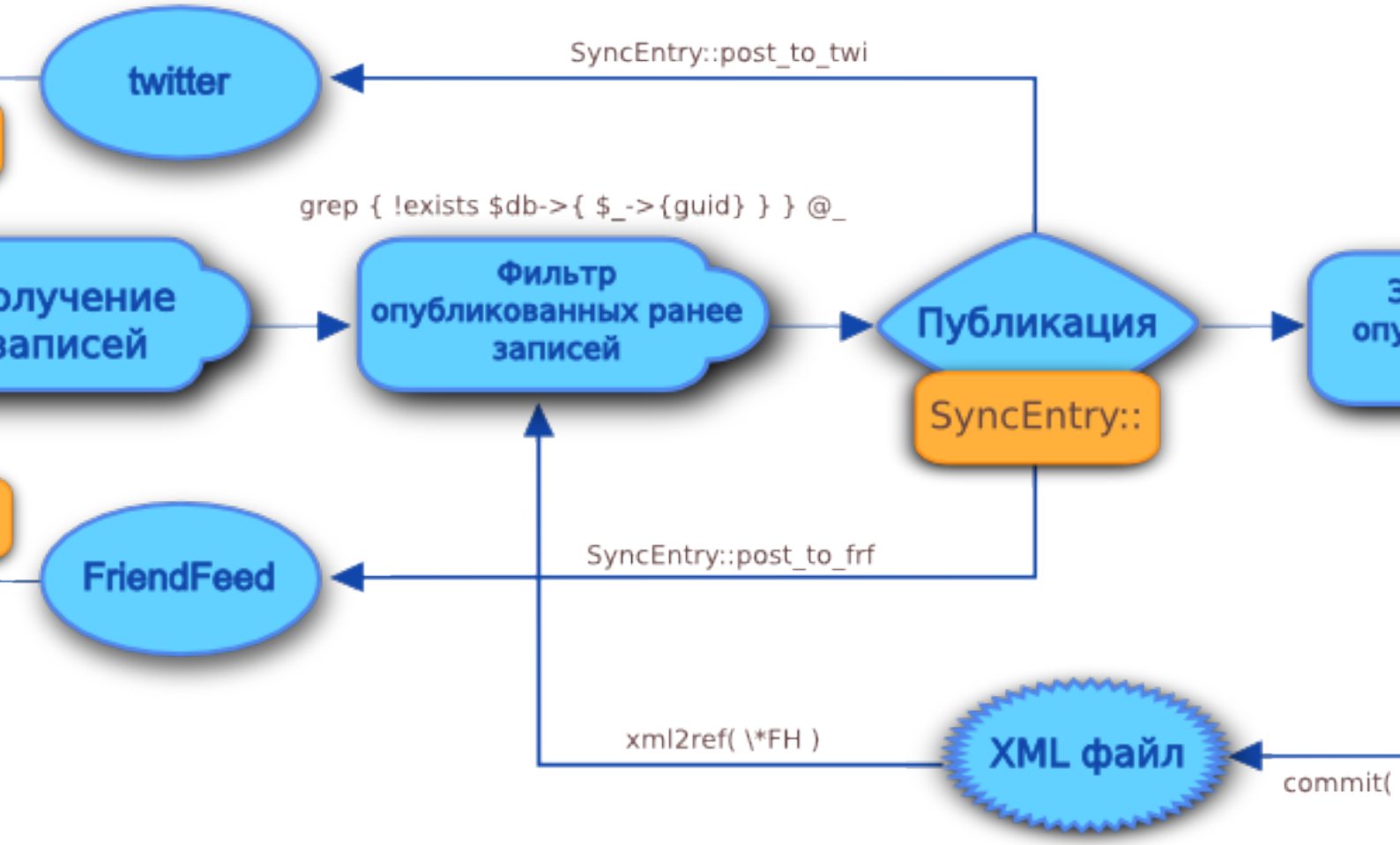
В качестве хранилища используется XML файл. Его содержимое при чтении преобразуется в Хэш, а по завершении работы сохраняется обратно в виде XML. Данный способ прост и нагляден.

Остальные функциональные блоки робота реализованы в виде модулей библиотеки `Flow` и скомпонованны затем в *pipe*.

Реализация социального робота

Полностью код робота опубликован по адресу http://github.com/zag/lveeboteg [11]. На следующей схеме отмечены фактические названия модулей и подпрограмм. Это поможет ориентироваться при изучении исходных кодов.

[11] Инструментарий для синхронизации группы в FriendFeed с twitter аккаунтом. http://github.com/zag/lveeboteg



Я пройдусь по основным моментам реализации.

API FriendFeed

На мой взгляд у этой социальной сети одно из самых простых и доступных в использовании API [12].

Вот пример запроса содержимого группы:

```
http://friendfeed-api.com/v2/feed/lvee?pretty=1&num=60
```

Формат вывода - JSON. Его легко преобразовать в массив записей с помощью модуля `JSON`:

```
use JSON;

sub fetch_frf {
#...
my $res=[]
if ( $response->is_success ) {
    $res = decode_json( $response->content );
 }
#...
}
```

[12] FriendFeed API. http://friendfeed.com/api/

Для публикации в сообщений потребуется специальный ключ и login пользователя. Поэтому в friendfeed заведен специальный пользователь, от имени которого сообщения из twitter публикуются в friendfeed. При отправке подобных запросов используется http заголовок авторизации:

```
    $r->header( Authorization => 'Basic '
          . encode_base64( $self->{frf_usr} . ':' . $self->{frf_key} ) );
    my $response = $ua->request($r);
```

В FriendFeed есть свой собственный сервис сокращения ссылок. Для доступа к нему необходим запрос с идентификатором заметки:

```
  my $res  = $self->post_frf(
    "http://friendfeed-api.com/v2/short",
    [ entry => $e->{entry}->{id} ] );
  $res = $res->{shortUrl} if $res;
```

Сокращение ссылок необходимо при отправке сообщений в twitter.

Так как в twitter попадают сообщения администраторов группы, то требуется получить этот список. Для этого используется запрос: http://friendfeed-api.com/v2/feedinfo/lvee?pretty=1.

API Twitter

При работе с этим сервисом необходимо уметь получать список последних записей, а также - публиковать новые. Для подобных запросов к API потребуются зарегистрировать робота [13].

Для работы с API twiiter используется Perl 5 библиотека Net::Twitter [http://search.cpan.org/dist/Net-Twitter/].

Подключение и использование библиотеки выглядит следующим образом:

```
 use Net::Twitter
 my $nt = Net::Twitter->new(
    traits              => [qw/OAuth API::REST/],
    consumer_key        => TWI_CONSUMER_KEY,
    consumer_secret     => TWI_CONSUMER_SECRET,
    access_token        => TWI_ACCESS_TOKEN,
    access_token_secret => TWI_ACCESS_TOKEN_SECRET,
 );
```

Значения констант `TWI_CONSUMER_KEY`, `TWI_CONSUMER_SECRET`, `TWI_ACCESS_TOKEN`, `TWI_ACCESS_TOKEN_SECRET` доступны после регистрации приложения в twitter.

Библиотека упрощает выполнение запроса на получение сообщений:

```
   $nt->user_timeline( { count => 30 } )
```

А также публикацию:

```
    $nt->update( { status => $text } )
```

При публикации сообщений в twitter требуется соблюдать ограничение в 140 символов. Если не получается опубликовать полностью, размещается сокращенное сообщение, снабженное ссылкой на оригинальное в группе friendfeed.

[13] Страница регистрации приложений для Twitter. https://dev.twitter.com/apps

Инструкция к социальному роботу

Как и в случае любого сложного устройства, к роботу должна прилагаться инструкция. В данном случае она будет следующей.

Сперва необходимо установить необходимые библиотеки:

Таблица 1.1. Необходимые Perl 5 библиотеки

Имя библиотеки	Примечание
Net::Twitter	FreeBSD: `portupgrade -N net/p5-Net-Twitter`, счастливым обладателям Debian (squeeze): `libnet-twitter-perl`
Flow	http://search.cpan.org/dist/Flow/
XML::Flow	FreeBSD: `portupgrade -N textproc/p5-XML-Flow`, http://search.cpan.org/dist/XML-Flow/
JSON	FeeBSD: `portupgrade -N converters/p5-JSON`, http://search.cpan.org/dist/JSON/

Далее потребуется зарегистрировать пользователя FriendFeed, от имени которого будет создавать сообщения робот. На данном этапе потребуется "Remote Key", предоставляемый сервисом FriendFeed для доступа к своему API [14]. Этот ключ и имя пользователя необходимо указать в качестве значений для констант `FRF_USR` и `FRF_RKEY`.

Создается группа в FriendFeed. Ее имя указывается в `FRF_GROUP`.

Следующим шагом производим регистрацию робота в twitter [15]. На данном этапе заполняются значения констант: `TWI_CONSUMER_KEY`, `TWI_CONSUMER_SECRET`, `TWI_ACCESS_TOKEN`, `TWI_ACCESS_TOKEN_SECRET`.

Также есть дополнительные настройки:

FRF_POST_ADMIN — Разрешает публиковать в twitter сообщения пользователей, являющихся администраторами группы. По умолчанию : 1.

FRF_POST_ADMIN_LIKED — Публикация сообщений, отмеченных администраторами как 'like'. По умолчанию: 1.

FRF_POST_USERS — Перечислены идентифиакторы пользователей, сообщения которых будут публиковаться в twitter. Например: ['lvee', 'lveeboteg'].

FRF_SKIP_TWITTER_SRC — Сообщения импортируемые средствами FriendFeed из twitter будут исключены. По умолчанию: 1.

Теперь осталось создать базу сообщений:

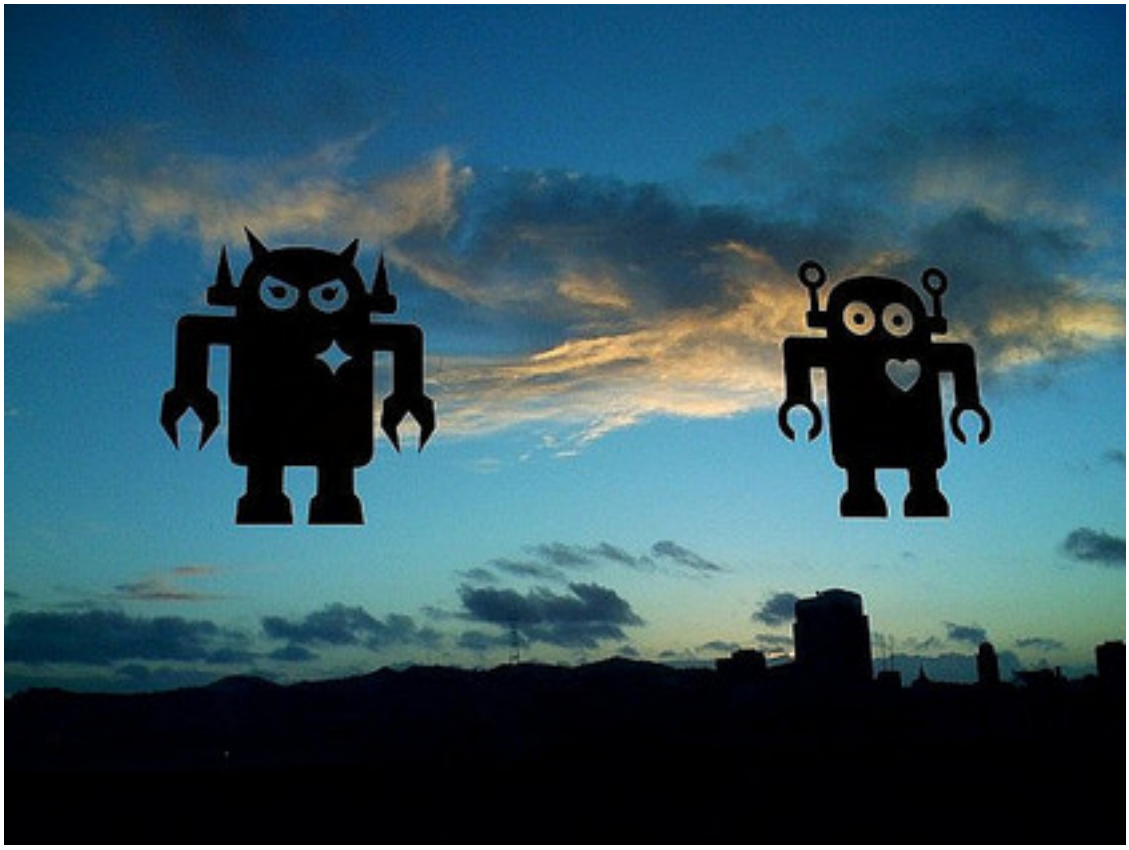
```
twifrf.pl -f lveecon.db  -fromfrf -fromtwi -init
```

[14] Страница с информацией о ключе для доступа к API FriendFeed. https://friendfeed.com/account/api
[15] Страница регистрации приложений для Twitter. https://dev.twitter.com/apps

и периодически запускать следующую команду:

```
twifrf.pl -f lveecon.db  -fromfrf -fromtwi
```

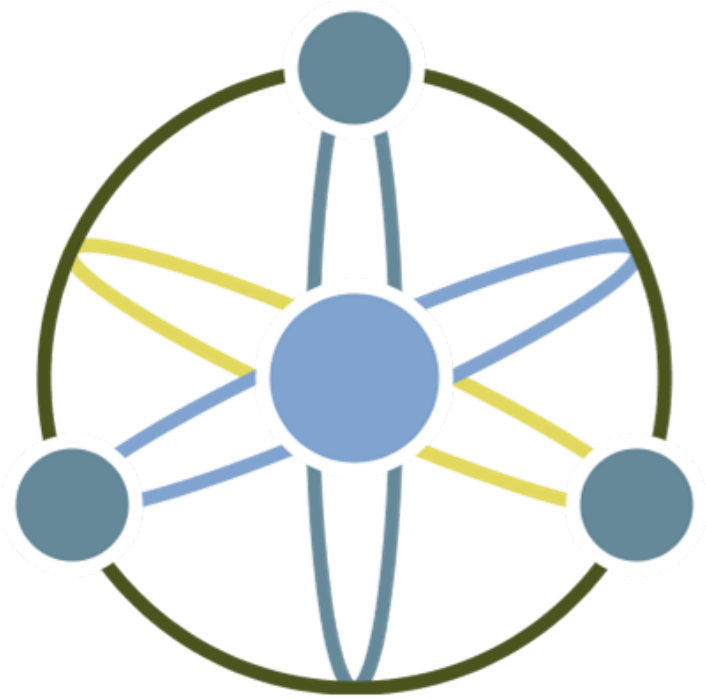
При ее выполнении новые сообщения, появившиеся в FriendFeed будут публиковаться в twitter и наоборот: из twitter попадать в группу FriendFeed. Если необходимо игнорировать новые сообщения в twitter - из командной строки удаляется опция `-fromtwi`.



Глава 2. Открытые протоколы

В данной главе, я расскажу об открытых протоколах и форматах, которые лежат в основе функционирования социальных сетей. Данные протоколы открыты и в их разработке принимали участие инженеры и специалисты ведущих компаний: Google, Microsoft, Yahoo!, Nokia и т.д.

Формат Atom



Формат Atom является во многих сервисах контейнером для дополнительной информации. Он представляет собой XML файл и содержит в структурированном виде информацию о ресурсе и его содержимом. Если в качестве ресурса выступает персональный блог, то Atom файл содержит информацию о самом блоге, а также содержимое и свойства каждой записи.

Простой пример Atom выглядит следующим образом:

```
<?xml version="1.0" encoding="utf-8"?>
  <feed xmlns="http://www.w3.org/2005/Atom">
    <title>Example Feed</title>
    <link href="http://example.org/"/>
    <updated>2003-12-13T18:30:02Z</updated>
    <author>
      <name>John Doe</name>
    </author>
    <id>urn:uuid:60a76c80-d399-11d9-b93C-0003939e0af6</id>
    <entry>
      <title>Atom-Powered Robots Run Amok</title>
      <link href="http://example.org/2003/12/13/atom03"/>
      <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
      <updated>2003-12-13T18:30:02Z</updated>
      <summary>Some text.</summary>
    </entry>
  </feed>
```

В общем он схож с более простым форматом RSS, но появился Atom позже и является более современной заменой RSS.

Так как в основе формата синдикации Atom лежит XML, это позволяет ему быть контейнером дополнительной информации, сохраняя при этом соответствие своей спецификации. Расширение формата возможно с помощью пространства имен XML [1]. Следующий пример демонстрирует определение пространства имен и параметров протокола Salmon [2].

```
<link rel="hub" href="http://myhub.example.com/endpoint" />
```

На формате Atom базируется протокол публикации Atompub [3]. Этот стандарт унифицирует последовательность действий при публикации данных на ресурсе (например, Фотографий или заметок в блоге). При этом Atom является основным форматом для размещения информации.

ActivityStreams - формат представления социальной активности пользователя

Помимо материалов, публикуемых пользователем в сети, еще одним ресурсом является его сетевая активность. Она включает в себя следующие действия: изменение статуса, создание заметки, публикация фотографии или альбома, отметка понравившихся материалов других пользователей и т. д. Данная активность собранная с различных аккаунтов позволяет быть в курсе действий пользователя. Чтобы формализовать эту активность был создан формат ActivityStreams [4].

[1] Namespaces in XML 1.0. http://www.w3.org/TR/REC-xml-names/
[2] Salmon Protocol. http://www.salmon-protocol.org/
[3] The Atom Publishing Protocol. http://tools.ietf.org/html/rfc5023
[4] Формат представления сетевой активности ActivityStreams. http://activitystrea.ms/

Участие в разработке спецификации ActivityStreams принимали участие представители известных компаний. На следующем снимке представлена одна из рабочих встреч, посвященная обсуждению этого формата [5]:

Данный формат основывается на следующих составных частях: автор *(actor)*, действие *(verb)*, объект *(object)* и цель *(target)*.

Примеры активности, представлены ниже:

Таблица 2.1. Сетевая активность

actor	verb	object	target
Person	*shared*	link	target
Person	*started following*	Person	target
Иван Васильевич	*меняет*	профессию	царские палаты
Вася	*разместил*	фотографию	picasaweb

[5] Встреча в офисе Six Apart, посвященная формату ActivityStreams. http://www.readwriteweb.com/archives/google_facebook_myspace_activitystreams.php

Форматы представления сетевой активности могут быть следующие: `JSON` и `Atom`. В случае `Atom` возможна интеграция в уже существующий feed.

Вот простой пример записи в формате JSON, взятый из спецификации [6] :

```
{
   "published": "2011-02-10T15:04:55Z",
   "actor": {
     "url": "http://example.org/martin",
     "objectType" : "person",
     "id": "tag:example.org,2011:martin",
     "image": {
       "url": "http://example.org/martin/image",
       "width": 250,
       "height": 250
     },
     "displayName": "Martin Smith"
   },
   "verb": "post",
   "object" : {
     "url": "http://example.org/blog/2011/02/entry",
     "id": "tag:example.org,2011:abc123/xyz"
   },
   "target" : {
     "url": "http://example.org/blog/",
     "objectType": "blog",
     "id": "tag:example.org,2011:abc123",
     "displayName": "Martin's Blog"
   }
 }
```

В следующем примере демонстрируется интеграция в формат `Atom`:

```
<entry xmlns="http://www.w3.org/2005/Atom" xmlns:activity="http://activitystre
 <id>tag:photopanic.example.com,2009:activity/4859/4352</id>
 <title>Geraldine posted a Photo on PhotoPanic</title>
 <published>2009-11-02T15:29:00Z</published>
 <link rel="alternate" type="text/html" href="http://example.com/geraldine/act
 <activity:verb>post</activity:verb>
 <activity:object>
   <id>tag:photopanic.example.com,2009:photo/4352</id>
   <title>My Cat</title>
   <published>2009-11-02T15:29:00Z</published>
   <link rel="alternate" type="text/html" href="http://example.com/geraldine/p
   <activity:object-type>photo</activity:object-type>
 </activity:object>
 <content type="html">
   &lt;p&gt;Geraldine posted a Photo on PhotoPanic&lt;/p&gt;
   &lt;img src="/geraldine/photos/4352.jpg"&gt;
 </content>
</entry>
```

Определение пространства имен для формата ActivityStreams производится следующей строкой:

```
<entry xmlns:activity="http://activitystrea.ms/spec/1.0/" >
```

[6] Спецификация JSON Activity Streams 1.0. http://activitystrea.ms/specs/json/1.0/

Формат AcvtivityStreams используется в распределенных социальных сетях, как формат передачи пользовательских событий. Так в протоколе Salmon, при создании пользователем комментария к статье на каком-либо ресурсе, производится идентификация автора и последующий импорт текста комментария или сообщения. Таким образом формат ActivityStreams вполне можно сопоставить с системой событий в операционной системе. Только вместо операционной системы - глобальная сеть Internet.