What is Vumi?

Vumi is a scalable, multi channel messaging platform. It has been designed to allow large scale mobile messaging in the majority world. It is actively being developed by the Praekelt Foundation and other contributors. It is available as Open Source software under the BSD license.

Why We Build on Open Source

Vumi is released under an open source license. It's also built upon many other pieces of existing open source packages. This allows the Vumi project to stand on the shoulders of giants and make use of tried and tested software that is already widely used in production instead of starting from scratch.

Building on open source software brings many benefits:

1. Risk of vendor lock-in is greatly reduced since access to software (and its source) is unrestricted
2. Availability of source code improves integration abilities and improves quality of technical support
3. Easy integration with server operating systems

No part of this is particularly new. Facebook Google and Amazon are all known to have built significant pieces of their infrastructure on Open Source software while also actively contributing improvements and releasing whole new projects under existing Open Source licenses.

The ability to collaborate in this wider Open Source software community is something we are passionate about.

Why we release under an Open Source Licence

Vumi passes the same benefits listed above to its users. It's released under a 3-clause BSD license which allows its users to use the Vumi software for any reason the user sees fit on a perpetual basis. External developers who use Vumi often submit improvements back to the Vumi codebase, resulting in a mutually beneficial relationship between the organisation and users who contribute.

What were the design goals?

The Praekelt Foundation has a lot of experience building mobile messaging campaigns in the areas such as mobile health, education and democracy. Unfortunately, a lot of this experience comes from having built systems that caused problems in terms of scale and/or maintenance.

Key learnings from these mistakes led to a number of guiding principles in the design of Vumi, such as:

1. The campaign application logic should be decoupled from how it communicates with the end user.
2. The campaign application and the means of communication with the end-user should each be re-usable in a different context.

3. The system should be able to scale by adding more commodity machines, i.e. it should scale horizontally.

The above mentioned guiding principles resulted in a number of core concepts that make up a Vumi application.

A Vumi Message

A Vumi Message is the means of communication inside Vumi. Esentially a Vumi Message is just a bit of JSON that contains information on where a message was received from, who it was addressed to, what the message contents were and some extra metadata to allow it to be routed from and end-user to an application and back again.

Transports

Transports provide the communication channel to end users by integrating into various services such as chat systems, mobile network operators or even traditional voice phone lines.

Transports are tasked with translating an inbound request into a standardized Vumi Message and vice-versa.

A simple example would be an SMS, which when received is converted into a bit of JSON that looks something like this:

```
{
    "message_id": "message1",
    "to_addr": "1234",
    "from_addr": "27761234567",
    "content": "This is an incoming SMS!",
    "transport_name": "smpp_transport",
    "transport_type": "sms",
    "transport_metadata": {
        // this is a dictionary containing
        // transport specific data
    }
}
```

Applications

Applications are tasked with either generating messages to be sent to or acting on the messages received from end users via the transports.

As a general rule the Applications should not care about which transport the message was received from, it merely acts on the message contents and provides a suitable reply.

A reply message looks something like this:

```
{
    "message_id": "message2",
    "in_reply_to": "message1",
    "to_addr": "27761234567",
    "from_addr": "1234",
    "content": "Thanks! We've received your SMS!",
    "transport_name": "smpp_transport",
    "transport_type": "sms",
    "helper_metadata": {
        // this is a dictionary containing
        // application specific data
    }
}
```

Dispatchers

Dispatchers are an optional means of connecting Transports and Applications. They allow for more complicated routing between the two.

A simple scenario is an application that receives from a USSD transport but requires the option of also replying via an SMS transport. A dispatcher would allow one to contruct this.

Dispatchers do this by inspecting the messages exchanged between the Transport and the Application and then deciding where it needs to go.

```
+----------------+
| SMS Transport  |<----+   +------------+    +-------------+
+----------------+     +-->|            |    |             |
                           | Dispatcher |<-->| Application |
+----------------+     +-->|            |    |             |
| USSD Transport |<----+   +------------+    +-------------+
+----------------+
```

How does it work?

All of these different components are built using the Python programming language using Twisted, an event driven networking library.

The messages between the different components are exchanged and routed using RabbitMQ a high performance AMQP message broker.

For data storage Redis is used for data that are generally temporary but and may potentially be lost. Riak is used for things that need strong availability guarantees.

A sample use case of Redis would be to store session state whereas Riak would be used to store all messages sent and received indefinitely.

Supervisord is used to manage all the different processes and provide an easy commandline tool to start and stop them.

Roles & Responsibilities

Each of the various components that make up the Vumi stack plays a specific role has certain characteristics that make it a good fit for a particular set of responsiblities.

Twisted

Twisted is an event-driven network programming framework written in Python and licensed under the MIT License.

Twisted projects variously support TCP, UDP, SSL/TLS, IP Multicast, Unix domain sockets, a large number of protocols (including HTTP, XMPP, NNTP, IMAP, SSH, IRC, FTP, and others), and much more. Twisted is based on the event-driven programming paradigm, which means that users of Twisted write short callbacks which are called by the framework. [1]

In Vumi, Twisted is the framework that provides the underlying bases classes for Applications, Dispatchers and Transports. It provides access to RabbitMQ and exposes great tools for connecting to or building HTTP APIs.

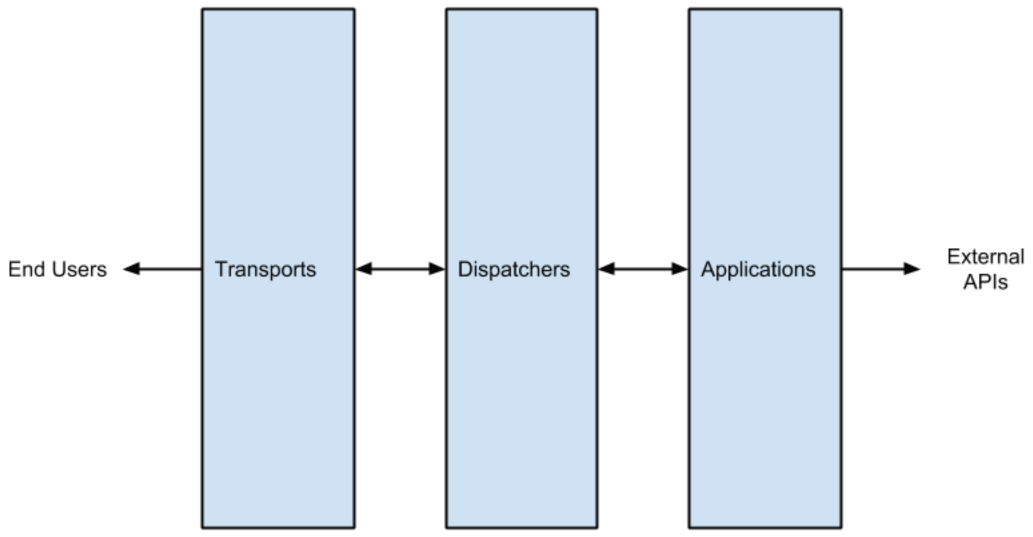
Central to Twisted Applications is the concept of a Deferred. A object that is a promise of a value that is not available yet. Deferreds allow Vumi to handle many tasks concurrently, a call to an external network (like a SOAP API for example), does not block it from processing other requests in the mean time. Nor does waiting for a slow API to respond incur any type of performance penalty.

RabbitMQ

RabbitMQ is open source message broker software (sometimes called message-oriented middleware) that implements the Advanced Message Queuing Protocol (AMQP). The RabbitMQ server is written in the Erlang programming language and is built on the Open Telecom Platform framework for clustering and failover. Client libraries to interface with the broker are available for all major programming languages. [2]

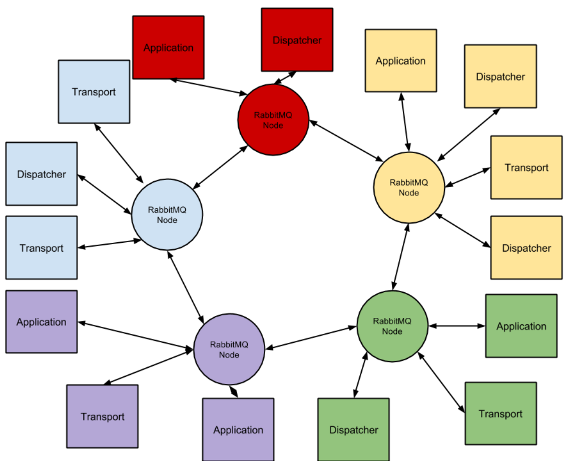
RabbitMQ is one of the main tools that enables Vumi to achieve scale. All internal message communications between Applications, Dispatchers and Transports happen through RabbitMQ following the Publish-subscribe pattern.

While conceptually Vumi is generally illustrated as looking like the following:



However, RabbitMQ operates in a fully clustered fashion and allows for application topologies that are completely distributed without a single point a failure.

Because of RabbitMQ realistically Vumi looks a lot more like:



Redis

Redis is an open-source, networked, in-memory, key-value data store with optional durability. [3]

Because of the distributed nature of Vumi and the need to eliminate single points of failure, there will generally be multiple instances of the same application, dispatcher or transport.

Redis provides Vumi with the ability to share bits of information between various actors in the system. It allows for a request that arrives via a transport on node 1 to be answered by another instance of the same transport on node 2.

Riak

Riak is an open-source, fault-tolerant key-value NoSQL database implementing the principles from Amazon's Dynamo paper with heavy influence from Dr. Eric Brewer's CAP Theorem. Written in Erlang, Riak is known for its ability to distribute data across nodes using consistent hashing in a simple key/value scheme in namespaces called buckets. [4]

Riak is Vumi's main storage database. All messages sent and received by Vumi is stored in Riak. Like RabbitMQ, Riak operates in a cluster of nodes. Riak is an eventually distributed database. This particular trait allows Riak to operate in a fashion where every node in the cluster is a master node and the database cluster can continue to operate without dataloss if a number of nodes fail.

Riak gives Vumi fast database write speeds, high availability and the ability to scale horizontally on demand.

Node.js

Node.js is a software platform for scalable server-side and networking applications. Node.js applications are designed to maximize throughput and efficiency, using non-blocking I/O and asynchronous events. Node.js applications run single-threaded, although Node.js uses multiple threads for file and network events. [5]

Vumi provides a wide range of applications. One of those is a specialized Application that provides a sandboxed environment within which third-party application code can be hosted. This sandbox provides a Node.js environment.

Applications within this environment have access to a standard set of re-usable resources such as access to the message store, key-value store, contacts database and the ability of interacting with external HTTP based APIs.

Vumi provides documentation as well as some example applications online.

1	http://en.wikipedia.org/wiki/Twisted_(software)
2	http://en.wikipedia.org/wiki/RabbitMQ
3	http://en.wikipedia.org/wiki/Redis
4	http://en.wikipedia.org/wiki/Riak
5	http://en.wikipedia.org/wiki/Node.js