RabbitMQ OOI SOW

OBJECTIVES

The purpose of this document is to summarise the Statement of Work (SoW) for Rabbit Technologies (Rabbit) on behalf of the University of California at San Diego (UCSD) for the Ocean Observatory Initiative (OOI) Cyber Infrastructure (CI).

The OOI CI Vision: A Pubsub network in the cloud

The CI¹ is a platform for use by a global network of scientists who study the ocean. OOI are part of a network of hundreds of institutions around the world, connected via the internet and private government-funded data networks. OOI represents the NSF’s contribution to this network - consisting of 55 institutions located around the US.

Each institution generates terabytes of data each day, from instruments with sampling frequencies from 10 to 200Hz, and from other sources. The data is in heterogeneous formats and could include data-sets, documents and video. The CI platform will provide a common mechanism for the institutions to collaborate by sharing data and metadata.

The CI architecture consists of several major components which are being piloted in 2009. The architecture, components, dependencies and use cases are on the OOI wiki².

An important component of CI is a Messaging Service³ based on the Advanced Message Queuing Protocol (AMQP). Using AMQP will provide CI users with four capabilities:

1. A means to systematically package oceanographic data they are producing, and metadata for example update notifications, as AMQP messages
2. A way to associate an address with each message in order that it may be processed for delivery to the correct consumers
3. A way for those consumers to express interest in those data streams that they are interested in tracking, via the AMQP’s notions of subscription.
4. A system for data to be routed and delivered to the subscriber via AMQP. In addition, for very large data sets out of band transmission may be used.

This AMQP messaging system will consist of “federations” of AMQP brokers acting together to form a data delivery network (DDN). There will be two types of AMQP broker in the DDN: broker clusters installed on site at participating institutions, and a virtualized broker infrastructure running on cloud computing environments including Amazon EC2.

In summary, the heart of DDN is a CI Pubsub Application optimised for scientific collaboration. The applications that connect to this will be software agents that are created by the OOI team, and deployed on any of the sites participating in the collaboration.

Below, the terms Messaging Service, Pubsub Messaging Service and Pubsub Application may be interchanged.

The role of Rabbit: creating a prototype Pubsub Application

¹ Main page: http://www.oceanobservatories.org/spaces/display/CIDev/Home
² See http://www.oceanobservatories.org/spaces/display/CIDev/Iteration2+Integration
³ See http://www.oceanobservatories.org/spaces/display/CIDev/Messaging+Service
This SoW describes a project to implement a prototype cloud Messaging Service for the CI Pubsub Application by deploying RabbitMQ on Amazon EC2. This service will have an API that interfaces between the cloud messaging service that Rabbit provides, and the OOI agent based and end user applications.

The delivery deadline for this work is four months after work commences. The work will start on August 15th (target date) and the plan is shown in the Implementation section below.

**Rabbit Deliverables**

Rabbit will deliver:

1. Open source software
   1. Extensions to the RabbitMQ broker and client code, to improve RabbitMQ and to provide interfaces suitable to the CI Pubsub client applications
2. Cloud-based Pubsub Messaging Service
   1. Implementation of a cloud messaging service that is specific to enablement on the Amazon platform, e.g. integration with Amazon web services. Performance will be baseline, which is deemed to mean: consistent with the experiences of reasonable users who are in production with RabbitMQ on AWS today.

There will be further projects post December 15th 2009 if Rabbit’s prototype is successful, towards a production quality system for multiple users launching in March 2010.

In summary Rabbit will implement its broker in the cloud, as a managed service, with CI mandated extensions. This can play the role of the Messaging Service for the CI Pubsub Application. An open source version of the broker and extensions will also be delivered.

**Ownership of software deliverables**

The open source software listed in categories (1.1.c.ii) and (1.1.e) below will be jointly owned by Rabbit and OOI and made available under a BSD license, non-exclusively in accordance with terms and condition specified in the JOI subaward JSA 7-11. Both parties will have full independent ownership rights to exploit such software in any manner and are not required to obtain the consent of or permission from the other party as long as such exploitation is consistent with the JOI subaward JSA 7-11.

All other open source software will be owned by Rabbit.

The service implementation for the “Cloud-based Pubsub Messaging Service” (2, above), will be owned by Rabbit and will not be open source. In the future, Rabbit will provide a commercial service for cloud messaging, by developing this prototype to production quality. This commercial service may be used by any paying customer and will not be limited to use in CI. OOI have stated that it is in their best interest for Rabbit’s cloud messaging business to be viable.
SPECIFICATIONS

This section describes the deliverables in more detail.

1. Open source software
   1. RabbitMQ broker and client extensions
   2. CI Pubsub Messaging Service interfaces
   2. Cloud-based Pubsub Messaging Service
      1. Implementation of a prototype on Amazon cloud

The following references are required reading for the specification below:

- OOI document Architecture for CI
  - http://www.oceanobservatories.org/spaces/display/CIDev/MS+Iteration2
- OOI document Use Cases and Actors for CI:
  - http://www.oceanobservatories.org/spaces/display/CIDev/Iteration2+Integration

1.1 RabbitMQ Broker and Client Extensions

The current latest release of RabbitMQ is version 1.5.4 which implements AMQP 0-8. This summer the Rabbit team will release a version supporting AMQP 0-9-1.

For the prototype messaging service, the following RabbitMQ broker and client extensions must be delivered:

a. AMQP 0-9-1 compliant RabbitMQ broker
b. AMQP 0-9-1 compliant Java, C. Note: OOI may also test using txAMQP (Python)
c. Federation of independent RabbitMQ brokers
   i. Please see details on AMQP Broker Federation below.
   ii. OOI requires an explicit Inter-Process Communication (IPC) API to an independent Communication Service be used to establish and transport the communications between the group of federated brokers (i.e. between broker clusters and not in a cluster). This IPC API will be defined by Rabbit. The IPC API is to be implemented as the Strategy Pattern\(^4\). Rabbit will provide an in-house developed strategy of this API. OOI will develop in parallel a separate strategy based on John Day's Distributed IPC Facility Network architecture which supports the replacement of the Rabbit strategy with the OOI strategy without requiring changes to the extended RabbitMQ Broker software.
d. Addressing
   i. OOI want a definitive addressing scheme in the prototype and this is to be agreed as early as possible during the work. See Federation below.
   ii. It is strongly preferred that this scheme coincide with AMQP 1.0. Therefore, this work is seen as a contribution to the 1.0 process (mediated by AR and MA).
e. Exchange Space and Exchange Point
   i. Please see details on this below. It is a novel notion meriting serious exegesis.

Note that Exchange Spaces (1.1.e) are a CI extension to AMQP. The remaining features are in AMQP 0-9-1 or will be in AMQP 1.0.

1.1.1 Exchange Space and Exchange Point

An exchange point is a virtual exchange not bound to any one broker but reified as a set of physical named exchanges on multiple brokers. Each exchange point belongs to an exchange space, and an exchange space collects and controls all the exchange points in its scope, so that Users may send messages to those exchange points.

The concept of an “exchange point” is that an AMQP exchange is not simply a named exchange, but that this named exchange is also a physical entry point into a “virtual exchange” consisting of a collection of exchanges across multiple brokers possibly in many domains. Hence an exchange point can be said to be reified across multiple brokers because it is a set of many entry points, each one being a physical named exchange. The dual nature of such an exchange is shown below.

This diagram should be compared with the use cases in the OOI wiki “Exchange Space” diagram which shows communication between clients C1-1, C1-2 and C2-1, as if through a ‘cloud’ of brokers B1 and B2. The same diagram lists two “Message Transfer Routing Examples” which should be compared with the paths in the diagram to the left on this page. Let B1 be represented by the broker shown as a grey box. Then the wiki diagram’s “Example One” (the red dotted message path) corresponds to delivery from P to C via the broker-local queue shown above in red. While, in wiki “Example Two”, the message must be forwarded onto a queue on a remote broker (following the blue dotted path). This would correspond to using the queue plus relay mechanism shown left in light blue.

1.1.1.1 Physical reification as multiple exchanges underdetermines delivery

For more complex scenarios, note that several designs may be investigated to implement eventual delivery of a message, e.g. to trade off between latency and bandwidth. Note that naive or rich DHT approaches may be used, but multicast may be more immediate. The examples below are merely illustrative and not a commitment to a design.
1.1.1.2 Exchange Spaces, Addressing and Identity

The above notion of an exchange point would suffice in a world where we only wanted to make one fixed set of physical exchanges look like one virtual exchange across some set of brokers. This is insufficient for supporting multiple communities, sub-communities, and use cases as anticipated by OOI. We use the OOI notion of an Exchange Space to collect Exchange Points into groups, bound together into a single administrative scope. We can think of Exchange Points taking care of message delivery, but User applications have a primary relationship with Exchange Spaces.

For example, an Exchange Space might provide a set of FX trading services to banks.

A sponsor, such as Reuters or a bank consortium, might create such a Space using infrastructure similar to CI, and then invite other parties, such as banks, to join the FX community represented by this Space. Each bank wishing to trade FX would join. Then, each bank would enrol brokers into the Space. Other brokers might be provided by the sponsor and by service providers, in the Cloud. These would also enrol in the Space.

All brokers enrolled in the space would present exchanges for membership in the Exchange Space. Each AMQP exchange would be a member of some Exchange Point, and each Exchange Point would provide some service such as “trade GBP-USD” or “trade USD-JPY”. Each service would use Pubsub to route messages such as “I buy 100 GBP” between participating brokers using the “virtual exchange” delivery model shown above.

Individual Users might access the system through a screen in which FX services and named counterparties (people, but represented by AMQP addressing and routing keys) would be visible through this screen. FX trading could then take place. The same screen might be used for banks to provide access to other Exchange Spaces, for example there might be a Space for a US Equity Trading community, and another Space for secure Instant Messaging (“chat”), and another Space for sharing PDFs of trade documentation. The same model could be used to federate across social applications eg. Flickr, Facebook.

The above example illustrates the following points:

a. Exchange Spaces administer Exchange Points so that Users may make use of Pubsub services accessible to some community
b. Exchange Spaces are a way to implement peered communities based on Pubsub infrastructure for data delivery across multiple domains
c. The concepts are those of Pubsub: Identity, Name, Enrolment, Discovery, Authentication, Addressing and these can be instantiated using any transport

Exchange Spaces combine Exchange Points with social application concepts: identity and membership of a community. Users in a community wish to make use of applications that require Pubsub to function. Whereas the Exchange Point is concerned with publication and delivery, allowing implementation of a single service, the Exchange Space provides a social context for whole applications servicing the community.

1.1.2 Federation and Addressing

For this work Federation is deemed to apply only to “a messaging system that is realised as a set of message broker domains sharing the same IPC syntax, semantics and naming”.
The following diagram illustrates a general pattern for implementing Federation across clouds (or “data centers”). This can be broker neutral and will suffice for some (but not all) cases prior to AMQP 1.0. It can be seen as prototyping the AMQP 1.0 concept of a Transfer Service\(^5\). In this design Exchanges are global and queues are local so it is consistent with CI notions in particular Exchange Space and Exchange Point.

\(^5\) See Rob Godfrey’s presentation at the 2009 AMQP Face to Face public event

1.1.2.1 Notes on Addressing

The above Federation design tolerates most of the plausible global addressing schemes including those tabled for AMQP 1.0 which are partly like SMTP and partly like UUCP.

Note that addressing for the Pubsub Messaging Service will need to accommodate notions of identity, enrolment, and discovery suitable for the Exchange Space.

1.2 CI Pubsub Messaging Service Interfaces

OOI will provide a specification document, detailing the Interface, during the project. The following represents a high level understanding of the Interface use cases and commands.

1.2.1 Use Cases: Actors

The CI Interface will provide for two classes of use cases by which Actors - agents and end user application services - can interact with the Pubsub Messaging Service:
a. Administrative Actors are software agents that e.g., take responsibility for a broker and create AMQP exchanges, queues; manage ES and EP lifecycle and related services eg ‘broker exchange’ and special ES/EP command queue for the ES admin.
b. User Actors will be end user application services that publish messages and perform other common “pubsub” actions such as subscribing to a data source. These actions will be mediated by Magnet Messaging Service Client (see below) and by normal AMQP discovery of correct exchanges. The Users are members of the ocean science community.

A diagram “Structuring the Messaging Service: Interfaces, layers, entities” is recommended, from the OOI wiki. It shows the “Service Layer” where User Actors initiate actions, and the “Contract and Control Layer” of agents in the role of Administrative Actors.

Further diagrams here depict Magnet, an agent/adaptor framework built by OOI, which manages the agents and mediates between the agents and the Messaging Service. Magnet will run on Amazon AMIs as well as on CI servers.

1.2.2 Use Cases: Interface Commands

The Interface refers to the “Messaging Service Interface” as shown in the “Structuring the Messaging Service” wiki diagram. This is the main abstraction layer in the whole Pubsub Application between the two main ‘tectonic plates’ of Pubsub service and Control agents.

The CI Interface will support two classes of command, by which the above Actors may interact with named resources in the Messaging Service. These are:

a. AMQP Commands
   i. The full AMQP 0-9-1 command set will be supported.
   ii. Resources in the system will have names which the Interface will expose.
   iii. Resources include AMQP resources e.g. exchanges
   iv. AMQP Messages will move between those named resources.

b. CI Pubsub Application Commands
   i. These are extensions to the AMQP command set, providing control capability for the broker enhancements in section 1.1 above, and other new constructs t.b.d.
   ii. Note: in the original AMQP 0-8 these sorts of extensions would have been called “Content Classes” or “Applications”
   iii. Resources in the system will have names which the Interface will expose.
   iv. Resources include CI Pubsub Application resources e.g. exchange spaces
   v. CI Pubsub Application Messages will move between those named resources

In other words the TOTAL set of Pubsub commands is a superset of AMQP.

The following list of use cases, from the OOI wiki, illustrates (b) above:

- Create an Exchange Space within the system

---

6 See [http://www.oceanobservatories.org/spaces/display/CIDev/MS+Iteration2](http://www.oceanobservatories.org/spaces/display/CIDev/MS+Iteration2)
7 [http://www.oceanobservatories.org/spaces/display/CIDev/Iteration2+Integration](http://www.oceanobservatories.org/spaces/display/CIDev/Iteration2+Integration)
8 See [http://www.oceanobservatories.org/spaces/display/CIDev/Magnet+Framework](http://www.oceanobservatories.org/spaces/display/CIDev/Magnet+Framework)
9 We have discussed the notion of including ‘social’ application constructs like “follow” and “relay” in the API but this is not in the formal scope of this project at the present time
- Create an Exchange Point in an Exchange Space (implies creating a representation of the Exchange Point in each Message Broker)
- Use an Exchange Point (use = put something in the queue or reading )
- Find out local Broker
- Add a Message Broker to the message broker federation

Further use cases are described throughout this wiki page: [http://www.oceanobservatories.org/spaces/display/CIDev/MS+Iteration2](http://www.oceanobservatories.org/spaces/display/CIDev/MS+Iteration2)

More use cases may emerge from the initial collaboration phase between TGJ and MM. See e.g.: [http://www.oceanobservatories.org/spaces/display/CIDev/Iteration2+Integration](http://www.oceanobservatories.org/spaces/display/CIDev/Iteration2+Integration)

### 1.2.3 Clients and language support

The Pubsub Messaging Service Interface features described above will be accessible using Java, C and Python clients. The Java and C clients will be delivered as part of the project by the Rabbit team, The Python txAMQP clients may be obtained from Fluidinfo. Note: Magnet is written in Python and txAMQP is its main Rabbit client.

### 1.2.4 Additional Notes

The messaging service should be transport independent at the architecture level.

In the future, the broker and service MAY support other protocols such as HTTP but we are still discussing this and HTTP messaging is not in scope for the prototype.

### 2. The Rabbit messaging service on Amazon Cloud

The CI will include a Rabbit Messaging Service that supports the Interface 1.2 above and includes the broker enhancements detailed in section 1.1. This service will run on Amazon EC2 and persist data to Amazon EBS. We call this service “Rabbit Cloud”, as distinct from the RabbitMQ broker software which users install and manage themselves.

For the avoidance of doubt:

*RabbitMQ will play two complementary roles in the CI: it will be available through the Rabbit Cloud service, and it will be available as an open source RabbitMQ broker for any user to run anywhere. Moreover the CI may use other AMQP brokers, and the DDN may use cloud hosted services from other providers e.g. SQLstream, that integrate with Rabbit.*

### 2.1 Specification of Rabbit Cloud

Please refer to the diagrams here: [http://www.oceanobservatories.org/spaces/display/CIDev/Iteration2+Integration](http://www.oceanobservatories.org/spaces/display/CIDev/Iteration2+Integration) which show the Messaging Service in relation to other roles.

Rabbit will deliver the messaging service in two formats:

a. Endpoint service
   i. This will be a broker that the Rabbit team hosts and manages itself, on Amazon EC2, which users may use in a similar way to Amazon Simple Queueing Service but through an AMQP API instead of the SQS API
ii. This is NOT a stated requirement for OOI during this project but it simplifies testing and fault isolation, and may be useful for eg. SQLstream.

b. Managed AMI service
   i. Rabbit provides dedicated, identified, preconfigured AMIs running a RabbitMQ broker and any other CI Pubsub Application functionality as detailed above.
   ii. Future:
       1. Not in scope for the prototype: support for additional OOI software not detailed in this document on the same AMIs
       2. In the future, Rabbit would provide a commercial managed support service covering e.g. problem triage, patches and AMI upgrade management

2.2 Simplifying assumptions for the prototype

The following assumptions are made; they simplify delivery of the prototype.

a. No Clustering
   i. The Rabbit Cloud prototype will not offer RabbitMQ in clustered mode. Eventually OOI would like improvements to the RabbitMQ cluster and how it is used and viewed by the client. These are NOT in scope for this piece of work.

b. No persistence for brokers on the “edge” of the system
   i. In the future (i.e. not in scope for this prototype) some edge brokers may have pluggable persistors to a remote EBS on AWS.
   ii. In the future it may be possible to connect on-premise RabbitMQ brokers to on-cloud storage such as EBS or S3 but this is not in scope.

c. Use RabbitMQ .deb files only
   i. OOI will take .debs from www.rabbitmq.com repos not from Ubuntu repos.

d. Only CI Pubsub Application software will be on AMIs
   i. Not in scope for the prototype: support for additional OOI software not detailed in this document on the same AMIs

e. Explicit routing
   i. Due to the potentially changing and fragile nature of AMQP federation, deployments will be explicitly routed. Dynamic routing is out of scope today.

f. Limited elasticity
   i. Modulo the ACLs requirement in section 2.3 below, Rabbit Cloud will limit multiple users to a single team during the prototype. That team is OOI.

g. No Performance Guarantees
   i. The prototype will not make any performance guarantees. Note that many Amazon services are still improving performance\(^\text{10}\).

h. No additional Cloud Management and No Billing
   i. We would ultimately like to offer e.g. a GUI eg to track usage, and billing tools. These are out of scope for the prototype.

2.3 Main Requirements for the Prototype

a. Ubuntu guests
   i. Rabbit Cloud AMIs will use Ubuntu 9.04 as the guest operating system

b. RabbitMQ broker
   i. Rabbit Cloud AMIs will provide a complete and preconfigured RabbitMQ broker

c. Single broker node persistence on-cloud
   i. Persist Rabbit Cloud AMIs on EBS for RabbitMQ for a single broker node

\(^{10}\) See report here [http://tech.mangot.com/roller/dave/entry/ec2_variability_the_numbers_revealed](http://tech.mangot.com/roller/dave/entry/ec2_variability_the_numbers_revealed)
d. Use Amazon static IP addresses
   i. Resolve local and global IP addresses on EC2.
e. Use RabbitMQ ACLs because a broker may have multiple users
   i. An exchange space is like a VLAN: we need ACLs to limit access to that on a
given broker; and a given broker may participate in multiple exchange spaces.

2.4 Business Model for Rabbit Cloud

The future business model for Rabbit Cloud will be an “on demand pay as you go” model
but the full details are to be determined. It is not required that this be finalised during the
project, but we welcome discussion with OOI on this point.

IMPLEMENTATION PLAN

3.1 Work items

3.1.1. AMQP 0-9-1

Support in the broker and in Java and C clients.

3.1.2. RabbitMQ CI AMIs on Amazon

Bundle RabbitMQ plus CI code on EC2 and EBS for testing and for prototype-production.

3.1.3. Design the API

Design of a prototype API for enhancements to AMQP to cover exchange space
management, virtual-host management, user and permissions management, statistics
gathering - per the above SoW.

   - Work with Michael Meisinger of OOI to architect the API
   - Collaborative design effort between OOI and RabbitMQ

4. Implement the API

Prototype functional implementation of the API from (3).

5. Implementation of the full prototype

Implement the remaining SoW CI code, including non-functional requirements of the
prototype on AWS. This includes Exchange Spaces, Exchange Points, Administration, as
stated in SoW. It also includes exploration of approaches to tools for broker federation,
message relaying and topic synchronisation.

3.2 Dependencies

The work splits into three independent streams:

(1) is an independent piece of RabbitMQ work that is not OOI specific;
(2) is an independent piece of work with both general and OOI-specific applicability;
(3) and (4) and (5) are more exploratory prototyping activities that are tightly interrelated.
3.3 Personnel

The development lead from RabbitMQ will be Tony Garnock-Jones for the full duration of the project. Tony will in addition be responsible for the design and implementation of items (3), (4) and (5) above. The remaining development work will be delivered by other team members, including Matthias Radestock and Matthew Sackman, who will be responsible for items (1) and (2).

Project oversight:

1. Alexis Richardson (OOI liaison)
2. Matthias Radestock (Technical and delivery oversight)

3.4 Schedule

Work on (5) can start at project kick-off - the target date is August 15th 2009. Exploration of the space ought to be complete enough to use by early September.

Work on (1) is part of the RabbitMQ AMQP development track. We are targeting AMQP 0-9-1 syntactic interoperability currently, with more complete semantic interoperability by late September.

Work on (3) should ramp up at project kick-off; we expect full engagement with Michael by early September, and completion of the design work by late September. Implementation work, (4), should begin early September and end late October.

Initial work toward automated EC2 deployment of RabbitMQ + OOI prototype code, (2), should be done at project kick-off, so that continuous integration can be done during development and refinement work. Initial EC2 configuration should be available for the start of (4), that is in September. Once work on (1) is ramping down in early September, the remainder of the EC2 configuration and integration work (2) can be done.

The time between ramping down activities (1) through (5) and project termination on the 15th of December will be used for review and refinement, documentation, and integration.

3.5 Assumptions and Project Risks

If work on item (3) is not started on time, or continues for longer than planned, it will affect item (4). Key risks here include the availability of Michael Meisinger throughout the period proposed.

The assumption of continuous customer acceptance testing may not hold. If testing by OOI is scheduled for near the end of the project, rather than as an ongoing activity during the project, it may affect either the project’s scope or its deadline.
3.6 Work plan

<table>
<thead>
<tr>
<th>Date</th>
<th>Dev 1</th>
<th>Dev 2</th>
<th>Dev 3</th>
<th>Mgmt</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 Aug 2009</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.3</td>
</tr>
<tr>
<td>22 Aug 2009</td>
<td>0.2</td>
<td>1</td>
<td>0</td>
<td>0.2</td>
</tr>
<tr>
<td>29 Aug 2009</td>
<td>0.4</td>
<td>0</td>
<td>0.4</td>
<td>0.6</td>
</tr>
<tr>
<td>5 Oct 2009</td>
<td>0.6</td>
<td>0</td>
<td>0.6</td>
<td>0.4</td>
</tr>
<tr>
<td>12 Oct 2009</td>
<td>0.8</td>
<td>0</td>
<td>0.8</td>
<td>0.2</td>
</tr>
<tr>
<td>19 Dec 2009</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>26 Dec 2009</td>
<td>0.8</td>
<td>0</td>
<td>0.75</td>
<td>0.25</td>
</tr>
<tr>
<td>2 Jan 2010</td>
<td>0.6</td>
<td>0</td>
<td>0.6</td>
<td>0.4</td>
</tr>
<tr>
<td>15 Jan 2010</td>
<td>0.4</td>
<td>0</td>
<td>0.25</td>
<td>0.75</td>
</tr>
<tr>
<td>17 Jan 2010</td>
<td>0.2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>24 Jan 2010</td>
<td>0</td>
<td>0</td>
<td>0.6</td>
<td>0.2</td>
</tr>
<tr>
<td>31 Jan 2010</td>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>14 Feb 2010</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
<td>0.8</td>
</tr>
<tr>
<td>21 Feb 2010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>28 Feb 2010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
</tr>
<tr>
<td>5 Mar 2010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.4</td>
</tr>
</tbody>
</table>

TOTAL Dev  TOTAL Mgmt
3rd 6d       3rd 6d
Contingency  Contingency
15%          15%
3rd 6 d      2nd 6 d

3.7 Pricing

The fee for this work will be USD XXX payable in two installments of USD XXX, of which the first payment is immediate on commencement and the second is on acceptance (Dec 15th). Expenses for use of AWS will be borne by Rabbit.