Computing for the Mathematical Sciences with XML, Web Services, and P2P

R. **Alexander Milowski**

2005-09-16

Abstract

While computing the Mathematical Sciences is similar to other scientific areas, often the researcher lacks the resources to carry out those computations. Grid computing and web services provide some possibilities for solutions but they do not address the increasing demand for computing resources and ad hoc computation networks. This paper describes a solution to this that uses peer-to-peer technologies to build ad hoc networks of computational agents that all speak XML to carry out computations.

Table of Contents

1. The Problem Statement

In many areas of science, XML and web services along with some form of grid computing are becoming the technology of choice for implementing large scale computational systems. These projects face a coming crisis in that they face the "Programming Challenge" [prog-challenge]. That is, they require a large team of software engineers to develop and deploy these applications.

Failing to face this challenge, the Mathematical Sciences has yet to utilize large-scale computation. Typical mathematical computations have large scaling problems that require specialized solutions. In contrast to the physical sciences where software engineering may be common practice, while the nature of Mathematical research may be computation, the nature of the researcher is not that of a skilled software developer. As a result, there are large barriers-to-entry for the researcher in Mathematics.

Even further, some mathematical calculations are vast in scale. Many computations go through stages where there are millions of polynomials that make up one computation. As a result, a single machine is limited in what it can do.

Finally, funding for computational research in pure mathematics isn't quite the same as that for other scientific areas in the physical or biological sciences. The typical Mathematician doesn't have access to large clusters on which to compute results. As such, researchers are limited to what they can compute on one machine given enough time or access.

Web services would allow mathematical researchers to use computational tools without having to become experts in software development and configuration management as well as enable them to utilize large computer resources. This is very similar to the use of web services in business to expose core competencies as services without the complexity of how those services are implemented. In many ways, the technology components of these web services in science or business are similar but the payload vocabularies are very different.

In fact, XML is a critical component in this mix because mathematical objects have structure. This structure is inherently what each service acts upon or enhances. As such, recording this in a portable and interoperable way is essential.

The twist is that web services aren't enough as computing grids aren't available. Computations must be accomplished on existing resources when time allows. That requires the ability to discover resources as they become available and knowledge of what that resource can compute.

The solution to this rests in peer-to-peer (P2P) computing for distributed computations. The use of P2P computing allows computing resources to check-in and check-out of computing environments and allowing the "cluster" to be distributed. This allows researchers in the Mathematical Sciences to take advantage of unused computer resources that exist in multitudes in computer labs, desktops at universities, and research centers.

This paper examines the architectural uses of XML pipelining, web services, and peer-to-peer technologies to build a computation grid for the Mathematical Sciences. The use of XML vocabularies to encode the description of services, orchestration of computations, and application payloads in XML will be discussed. Finally, the results of deploying an initial version of this P2P grid using the Xeerkat project [xeerkat] in the fall of 2005 will be discussed.

2. Grid Computing in Mathematics

It may be hard for a non-mathematician to imagine in which situations pure mathematics need grid computing. Certainly, at the center of many scientific computations is some kind of mathematical computation. But there are many mathematical computations whose result is just a mathematical object of interest and these objects take time to compute.

For example, at the center of computational algebra are objects called Gröbner bases [groebner]. They are central to solving algebraic, statistical, or optimization problems. They are also central to studying Mathematical ideals. The algorithm for computing these objects is double-exponential in computation time but is rather simple to distribute.

Due to the expensive computation needed to produce a Gröbner basis, their use has remained rather limited to theory. While more numerical methods have been successful, algebraic methods that use Gröbner bases have now become more popular and somewhat more feasible. Yet, computing these bases really feasible relies on the ability to distribute their computation.

Further, even though Gröbner bases have been used in theory, again their use has been limited by the size of the object that can be computed. Theoretical research would benefit greatly by the ability to compute large bases easily. Many conjectures and new properties would be discovered just by their existence.

3. The Worker Analogy and the Grid Economy

The current state of computing at most universities is there are disparate labs run by different departments, desktops and servers in offices, and high-performance clusters tucked away in corners for specific purposes. Most of these computing resources remain unused for large portions of the day. Yet, researchers continually need to locate or purchase expensive equipment for computations while existing resources remain unusable due to their location or their regular use (e.g. daily lab users). What is needed is to match the needs with the abilities.

Let's compare this to the process of hiring and utilizing workers in an organization. When a business needs a human resource in their company they use a multitude of avenues to locate and hire a worker. Usually, a worker is first located by their ability and availability.

Once a worker is found, an offer is extended to that individual. That individual has the right to refuse based on any number of reasons. Ultimately, for the business to succeed, that individual as a skilled resource must be located and hired.

Given this concept, we make analogous constructs in our computational grid. First, we'll place a computation in the role of the business. Here the computation is an active participant in deciding how its ultimate goal is achieved. This allows for strategy-based computations where the algorithms chosen are based on the resources available.

A computation will solicit workers--other computing resources--based on their abilities (e.g. software) and their availability (i.e. scheduled time, current status, etc.). A worker will also take the form of an active agent. Workers can interact with the requester and decide whether they will accept a job. Further, they can refuse and notify others of their status with regards to availability, ability, and current activities.

Finally, we want to have an "active economy" where the computations-as-business have services to sell. Here we will introduce a abstract concept of a gateway in which "goods" (i.e. computation requests) flow in from and back out towards. Essentially, those gateways service the consumer who participates "remotely" in the economy.

Extending this further, nothing prevents a worker from delegating to other workers (i.e. subcontractors) or for a worker to work for more than one requester (e.g. a consultant). Also, nothing prevents workers from taking roles as brokers of workers (i.e. a recruiter). In fact, these may be essential roles. For example, recruiters help load balance a set of resources to ensure they work on the right kind of jobs, are consistently busy, and also to offload recruiting work from a computation. All these relationships are show in Figure 1, “Worker Analogy Relationships”.

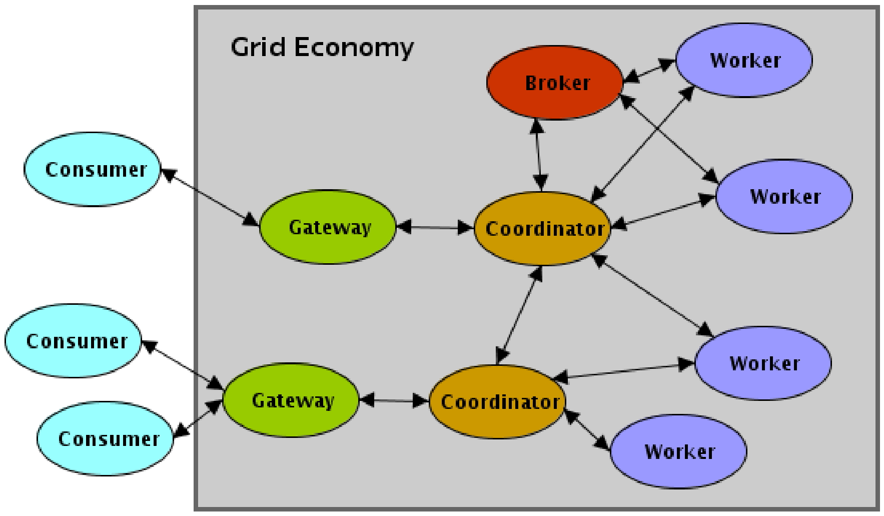

Figure 1. Worker Analogy Relationships

This analogy can be formalized by defining the messages between the consumer and the grid economy as well as the messages between the peers in the grid economy. First, a consumer interacts with the grid via a gateway peer. These peers expose some well know protocol (i.e. HTTP + XML or SOAP) at a fixed address. The consumer interacts with this peer by sending an XML message over that protocol.

The consumer's message is specific to the kind of operation and not generalized. That is, the message is specific to the computation. This way the gateway peer can map between a message and a specific agent role. This is shown in Figure 2, “Abstract Consumer Connection to the Grid Economy via the Gateway”.

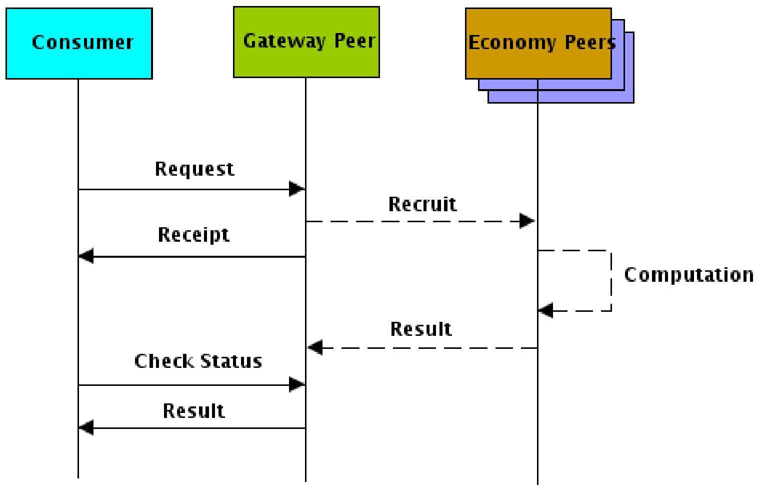

Figure 2. Abstract Consumer Connection to the Grid Economy via the Gateway

In general, the consumer connects to the gateway and sends a message. If that message is accepted, the consumer is given a receipt which can be used to check the status or retrieve the result of a computation. The gateway then proceeds to recruit a peer for the computation.

Once a willing coordinator peer is found, the message is passed to that coordinator peer in the grid economy. That peer may interact with other peers to compute a result. In the end, the result is returned to the gateway peer. That result is sent back to the consumer when it submits a request for the result.

4. The Peer-to-Peer Grid Economy

For this computational grid we'd like the characteristic that computational resources can come and go and as they do so, others are notified. Unfortunately, relying on a centralized repository of state is intractable as the number of resources grows. As a result, a different approach is needed to implement the grid economy.

There are three necessary aspects for this grid economy to work:

1. *Self-describing document encapsulations must exist of peers, resources, and services in the grid.* This is necessary as the grid economy must be decentralized. These documents must be able to be passed around, handed from peer to peer, so that a peer who needs the resources described has the information necessary to act.

2. *Propagation of these documents must be possible throughout the network*. Each of the messages and descriptions needs to be able to be propagated through the network via the peers who are in contact with each other.

3. *Dynamic discovery of these documents must be feasible.* When a peer is interested in a particular kind of service or peer, a query facility must be available so that the peer can ask a question and get an answer.

Fortunately, this grid economy is a perfect fit to the technology that exists in the Peer-to-Peer (P2P) JXTA Project [jxta]. At the core of JXTA is the concept of discovery of advertisements--which are just XML documents. Advertisements describe everything from peers and network protocols to the applications running within the P2P network.

Through the use and discovery of advertisement documents, JXTA makes both the network and applications self-describing. Application agents can query the JXTA network to discover other resources as well as find network protocol routes with which to communicate. In addition, the core protocols are all based on XML.

In their simplest form, ads are just particular kinds of XML documents. An application can make up its own types and publish the ad into the JXTA network. The JXTA network takes care of propagating this ad through a set of rendezvous nodes (much like a DNS server). Every peer is connected to some rendezvous node and so an ad can eventually be migrated through the network to every peer.

Once an ad is published into the JXTA network, it can be discovered dynamically by any other peer. Peers that know how to decode particular kinds of XML ad documents can take advantage of those ads. Others just ignore ads they don't understand.

Finally, much like the web, everything in the JXTA network are identified by IRI values [iri]. Peers have an IRI (a kind of URI) associated with them and can self-organize themselves into groups which are also identified by IRI values. This allows ads to be published that describe and resource by pointing to an IRI value.

This JXTA-style architecture is used to build the grid economy. It fits quite nicely along side the analogy of hiring workers based on abilities and availabilities. Peers query for advertisements that fit the resource(s) for which they are searching. As they become available, they are found.

4.1. Transport Independence

One of the essential features of JXTA is the idea of transport independence. When a peer needs to send a message to another peer, it need not know at the surface what protocol and end address is needed to contact that peer. It just sends that message over a pipe.

The discovery of transport routes happens in the same way as requesting all other resources. A discovery query is sent to about the specified peer as how to open a pipe and an advertisement is found. That advertisement describes the routes and protocols necessary to contact that peer. Internally, this information is used to resolve the pipe to the peer and to send the message.

This is very unlike web services in that there is no direct binding listed in a WSDL instance somewhere. There is no requirement for knowledge of the protocol at all for the application. Instead, this is done behind the scenes by the JXTA infrastructure. All the application needs to do is formulate the XML document it wishes to send.

The essential bit about this is that JXTA can choose the right protocol to contact a peer depending on its situation. The coding of that information is not managed by the XML messaging application. As a result, the network and the infrastructure managers are free to do what they consider to be the correct way to provide network-level connectivity and security. Also, protocols and routes may change depending on the peers available but the application is insulated by the use of JXTA from these changes.

4.2. The Agent Virtual Machine

An agent is a grouping of roles that can accomplish a certain task or computation. Each role is identified by an IRI and defines a particular computation or coordination of a computation. In essense, a set of agent roles IRI values defines the agent as a set of opaque services to which messages can be sent.

Each peer acts as an agent host and uses the definition of the agent to enable a certain role to be played by that peer. Associated with this peer is a set of agent implementations tied certain agent roles. When asked to do so, the peer enables a certain agent implementation and plays host to a specific role. To enable, disable, or transact with this role, the peer acts as a "Virtual Machine" that processes XML messages as its instruction set.

At the core level, a negotiation messaging process must take place for a peer to enable a role and process role-specific messages. In the analogy given earlier, this is the process of hiring a worker or coordinator. A hire request message is sent to the peer and intercepted by the virtual machine. That agent host decides whether the peer is busy, if the role is available, and whether the peer will accept the message.

If the agent role is rejected, a response message indicates that to the requesting peer. Otherwise, a acceptance message is sent to the requesting peer. Either way, the peer should respond with a yes or no message. This messaging is shown in Figure 3, “The Hire Message”.

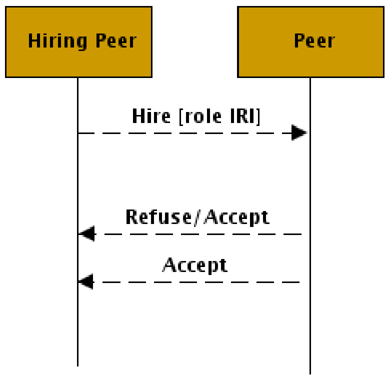

Figure 3. The Hire Message

Example 1. Definitions for namespace http://www.xeerkat.org/Vocabulary/Agent/2005/1/0

```
<xs:element name='hire'
               xmlns:xs='http://www.w3.org/2001/XMLSchema'>
  <xs:complexType>
    <xs:attribute type='xs:anyURI' use='required' name='from'/>

    <xs:attribute type='xs:anyURI' use='required' name='group'/>

    <xs:attribute type='xs:anyURI' use='required' name='role'/>
  </xs:complexType>
</xs:element>

<xs:element name='refuse'
               xmlns:xs='http://www.w3.org/2001/XMLSchema'>
  <xs:complexType>
    <xs:attribute type='xs:anyURI' use='required' name='from'/>

    <xs:attribute type='xs:anyURI' use='required' name='role'/>
  </xs:complexType>
</xs:element>

<xs:element name='accept'
               xmlns:xs='http://www.w3.org/2001/XMLSchema'>
  <xs:complexType>
    <xs:attribute type='xs:anyURI' use='required' name='from'/>

    <xs:attribute type='xs:anyURI' use='required' name='role'/>
  </xs:complexType>
</xs:element>
```

Here it is important to note that in the P2P world, all messages are really asynchronous. A peer may disappear from the network or become unavailable. Sending a message involves using transports where the response is not intended to come over the same synchronous protocols used to send the request. As such, the response shown above happens asynchronously as indicated by the dashed arrows.

At any point after being hired a peer can resign. This is a simple one-way message shown in Figure 4, “The Dismiss and Resign Messages”. This allows a peer to resume other duties (e.g. acting as a desktop or lab computer) quickly.

If the hiring peer is finished with the services of a peer, it can send a dismissal message. This is a two-way exchange where there dismiss message is sent and a resignation is received. While the resignation isn't necessary it does provide the means to account for determining whether the resigning peer was still available. This is also shown in Figure 4, “The Dismiss and Resign Messages”.

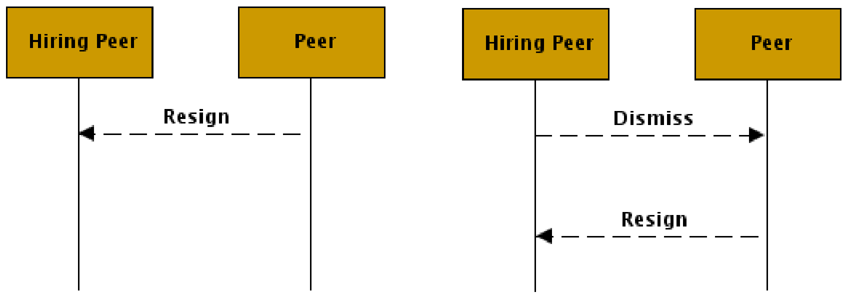

Figure 4. The Dismiss and Resign Messages

Example 2. Definitions for namespace http://www.xeerkat.org/Vocabulary/Agent/2005/1/0

```
          <xs:element name='resign'
                        xmlns:xs='http://www.w3.org/2001/XMLSchema'>
            <xs:complexType>
              <xs:attribute type='xs:anyURI' use='required' name='from'/>

              <xs:attribute type='xs:anyURI' use='required' name='group'/>

              <xs:attribute type='xs:anyURI' use='required' name='role'/>
            </xs:complexType>
          </xs:element>

          <xs:element name='dismiss'
                        xmlns:xs='http://www.w3.org/2001/XMLSchema'>
            <xs:complexType>
              <xs:attribute type='xs:anyURI' use='required' name='from'/>

              <xs:attribute type='xs:anyURI' use='required' name='role'/>
            </xs:complexType>
          </xs:element>
```

Once the peer has accepted an agent role, the role's implementation is ready to accept messages. These messages are simply those messages that are not defined above as agent VM messages. When encountered, they can be passed directly to the role's implementation. This is a simple operation of checking the namespace on the document element to discover whether it is a peer message or agent VM message.

While only a few messages have been described here, the agent VM message set can be extended to include other common operations. These operations might include such things as status of the agent host, shutdown control, upgrade of agent implementations, etc. Essentially, the agent VM message set is inherently extensible.

4.3. Discovering Resources

A peer is discovered by finding a peer agent advertisement (an XML document). This ad document describes the agent and the roles that it supports. By checking this document, another peer knows what agent roles it supports and the IRI that identifies the peer.

In the JXTA network a peer publishes this agent advertisement. Other peers can then query for agent advertisements. As the advertisement propagates through the JXTA network, peers are able to discover new peer abilities when they query.

This agent document is described below. It consists of an advertisement ID, the peer id, the peer input pipe for communication, and some number of agent role IRI values.

Example 3. Definitions for namespace http://www.xeerkat.org/Vocabulary/Agent/Ad/2005/1/0

```
<xs:element type='my:AgentAdvertisement' name='agent'
               xmlns:xs='http://www.w3.org/2001/XMLSchema'/>

<xs:complexType name='AgentAdvertisement'
                   xmlns:xs='http://www.w3.org/2001/XMLSchema'>
  <xs:sequence>
    <xs:element name='types'>
      <xs:complexType>
        <xs:sequence>
          <xs:element type='xs:anyURI' minOccurs='0' name='type'
                         maxOccurs='unbounded'/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>

  <xs:attribute type='xs:anyURI' use='required' name='id'/>

  <xs:attribute type='xs:anyURI' use='required' name='peer'/>

  <xs:attribute type='xs:anyURI' use='required' name='pipe'/>
</xs:complexType>
```

4.4. Agent Role Messages and XML Pipelines

Agents can communicate with each other by sending XML messages to each other. As a peer in the JXTA network is identified by an IRI, the JXTA network facilitates sending messages directly to a peer. As such, an agent just needs to know an IRI value and it can send a message to another peer and JXTA takes care of the transport.

For the purposes of simplicity, a message is just an XML document. Any message that is not an Agent VM message is considered to be an Agent Role message. These messages can be any document or use any kind of layered vocabulary to create a multi-part document or enable the ability to transport binary data. All an agent implementation needs to know how to construct XML messages.

As the environment is P2P, messages are asynchronous by nature. Peers respond to messages they receive and then can send responses if they wish. There is no synchronous messaging within an agent without the agent accomplishing this themselves with another layer.

That said, a request/response message pattern is quite simple as the sender is identified when the message is received and a response can be generated as soon as the message is processed. This is much like messaging operations in a MOM-architected application (i.e. message-oriented middleware architecture).

When peers respond to a message, some bit of code receives an XML document as a streaming XML Infoset [xml-infoset]. That is, a set of XML Infoset items (much like StAX [stax]) that describe the XML document. This allows a peer to keep its overhead to a minimum when processing messages.

A perfect match for processing messages is an XML pipeline [smallx-pipeline]. In its simplest form, an XML pipeline is a sequence of manipulation steps. It might be a chain of XSLT [xslt] transforms or a pair of XML Schema [xml-schema] validation steps that wrap a bit of custom code. Either way, the input and output is an XML Infoset and the output can be sent as the response to the sender of the message.

XML Pipelines are the easiest way to define agent roles. Rather than writing computation code against the agent API, the computation is written as a XML filter step and consumes and produces an XML Infoset. That XML filter is easily integrated as a step in an XML pipeline and the result is a separable process--the XML Pipeline--that can be tested or reused external to the P2P agent environment.

4.5. Gateways as Web Services

Consumers interact with the grid economy through special peers called gateways. These gateways broker the introduction of the message into the P2P grid and the routing of the message back to the consumer. Typically, this process is asynchronous as the whole computation might take longer than a consumer would want to wait.

One way for a consumer to interact with the gateway peers is via a web service. The consumer interacts with an asynchronous web service to send messages into the grid economy and then uses the service to inquire about the result. In turn, these services are handled by gateway peers.

A typical enterprise environment uses a queuing system to feed messages from the service to an application like the gateway peers. The incoming service makes a brief check of the incoming message and places the request into an "inbox" queue. A gateway peer is attached to that queue and is then notified of an incoming message.

The gateway peer can then go through the process of recruiting a coordinator peer for the computation and route the request to that peer. The routing of the message is recorded and the gateway peer can continue. Eventually, the response is sent to the gateway peer and an outbound message is created in an "outbox" queue.

This whole process is shown in Figure 5, “Gateway Service”. The labels in the figure correspond to the steps below:

1. The consumer interacts with the web service by sending an XML message. If that message is acceptable, it is placed in an "inbox" queue and a receipt is returned to the consumer.
2. The message waits in the inbox queue for a gateway peer to accept the message. In theory, there can be multiple gateway peers interacting with this queue to provide scaleability.
3. The gateway peer picks up a message from the queue and recruits a coordinator.
4. If a coordinator is found, the message is passed to the coordinator. Otherwise, a response is generated and the gateway peer proceeds to step 7.
5. The coordinator processes the message and proceeds with its computation--possibly interacting with other peers.
6. A response is generated by the recruited coordinator and that message is sent back to gateway peer.
7. The response is placed back in the "outbox" queue by the gateway peer with the proper correlation to the receipt given in step 1.
8. The response waits in the outbox queue for some allowed amount of time.

9. The consumer checks with the web service to see if there is a response. If there is a message in the queue corresponding the receipt given in step 1, that message is returned. Otherwise, the service returns a message signifying that the response is not available yet.

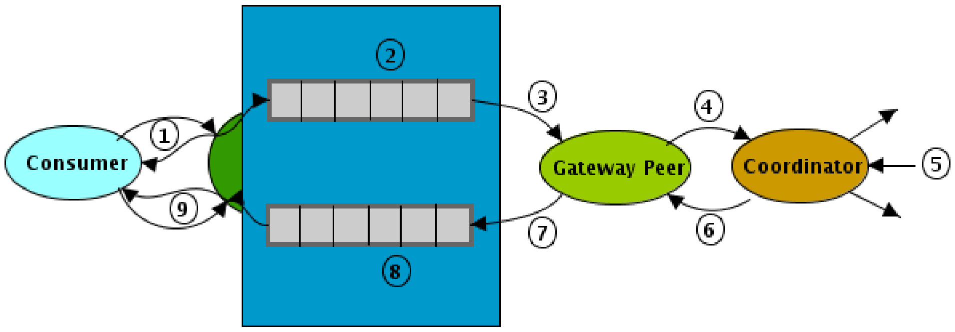

Figure 5. Gateway Service

This description of the interaction between the consumer, service, queues, and gateway peers is somewhat typical for asynchronous services. The difference here is that web service is a simple front end that checks the message for a particular structure, type of consumer, and whatever other special filters a gateway service provider might want to impose. Once the message passes those checks, it is blindly placed in a queue.

Independently, that queue is being serviced by a peer in the grid economy. Typically, this gateway peer must be provided by the same service provider for security reasons. Nevertheless, that separation of concerns allows the deployment of front-end services to be specialized to particular users while the gateway peer might be much more general.

Further, the front-end web service can be implemented by an XML pipeline. The following pipeline is an example XML pipeline in the Smallx XML Pipeline format [smallx]. This pipeline validates an incoming "ping" message with XML Schema and then inserts the response into a JMS queue.

Example 4. Submit Pipeline

```
<p:pipe name="submit"
   xmlns:p="http://www.smallx.com/Vocabulary/Pipeline/2005/1/0"
   xmlns:c="http://www.smallx.com/Vocabulary/Pipeline/Component/2005/1/0"
   xmlns:f="http://www.smallx.com/Vocabulary/Pipeline/Forms/2005/1/0"
   xmlns:jmss="http://www.smallx.com/Vocabulary/Pipeline/JMS/Steps/2005/1/0"
   xmlns:jms="http://www.smallx.com/Vocabulary/Pipeline/JMS/2005/1/0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:mg="http://www.mathgrid.org/Vocabulary/Services/2005/1/0"
>

<p:trap>
  <p:validate assert="true">
    <p:map href="ping.xsd"/>
  </p:validate>
  <p:on-error>
    <p:template>
       <mg:error>
          <xsl:text>Your document was not valid due to: </xsl:text>
          <xsl:value-of select="/c:error-context/c:error"/>
       </mg:error>
    </p:template>
  </p:on-error>
</p:trap>

<p:route>
  <p:when test="/mg:error">
    <p:identity/>
  </p:when>
  <p:otherwise>
    <jmss:send queue="mathgrid-net-in"/>
    <p:template>
      <mg:message ref="{/jms:message/@ref}">
      <xsl:text>Your message has been successfully queued.</xsl:text>
      </mg:message>
    </p:template>
  </p:otherwise>
</p:route>

</p:pipe>
```

4.6. The Ping Example

The simplest and most fundamental example is ping--bouncing messages between peers. In the case of the full computational grid, there is a web service that accepts a ping request, a gateway that processes those messages, and some number of peers that respond. In the P2P context, ping needs to be a bit different.

In the grid economy we want a ping request to distribute the ping messages between a coordinator peer and some number of worker peers. In this way, the ping request is between multiple peers running the ping agent roles rather than just two. This tests the infrastructure as well as serves as an example for how similar computations might work.

In this definition of ping, we'll need just two roles:

1. Ping Coordinator The coordinator that accepts the initial ping request and recruits workers which it will ping. This role is identified with the IRI "http://www.xeerkat.org/Agents/Roles/Examples/Ping/Coordinator".

2. Ping Worker This worker receives ping messages and echos them back to the sender (the coordinator). This role is identified with the IRI "http://www.xeerkat.org/Agents/Roles/Examples/Ping/Worker".

When the gateway receives a ping request, it recruits the Ping Coordinator to handle the message. This coordinator recruits Ping Worker peers. As it discovers these peers, it hires them and sends a ping message. While discovery is happening, the Ping Coordinator can also send ping messages to Ping Worker that have already been hired. In the end, the goal is to send some number of ping messages, receive their responses, and do so in a minimal amount of time.

This whole process is diagrammed in Figure 6, “Ping Example” and the steps aredescribed below:

1. A hire message is received by the agent peer. This message identifies the agent role requested. In this case, the role is that of a Ping Coordinator. If the agent is available, it will respond with an accept message. It should be noted that the hiring peer is not identified. In most instances it is a gateway peer but nothing prevents another coordinator peer from hiring this agent as a Ping Coordinator.

2. A ping request XML message is sent to the Ping Coordinator. This message has the following definition:

Example 5. Definitions for namespace (none)

```
<xs:element name='ping-request'
               xmlns:xs='http://www.w3.org/2001/XMLSchema'>
  <xs:complexType>
   <xs:attribute type='xs:int' use='required' name='count'/>


    <xs:attribute type='xs:int' name='idle-max'/>

    <xs:attribute type='xs:int' name='hire-limit'/>
  </xs:complexType>
</xs:element>
```

3. The Ping Coordinator discovers agent ads by interacting with its rendezvous peer.

4. An agent is picked from the available agent ads and a Ping Worker is hired.

5. Any number of ping messages can now be sent to the hired Ping Worker. These messages conform to:

Example 6. Definitions for namespace (none)

```
<xs:element name='ping'
               xmlns:xs='http://www.w3.org/2001/XMLSchema'>
  <xs:complexType>
   <xs:attribute type='xs:int' use='required' name='count'/>

  </xs:complexType>
</xs:element>
```

6. If available, another agent is picked, hired, and ping messages are sent.
7. Once the requested number of ping messages have been sent, the Ping Worker agents are dismissed.
8. The Ping Coordinator now sends the results of the ping to the hiring peer. This message conforms to:

Example 7. Definitions for namespace (none)

```
<xs:element name='ping-stats'
               xmlns:xs='http://www.w3.org/2001/XMLSchema'>
  <xs:complexType>
    <xs:attribute type='xs:int' use='required'
                 name='count'/>

    <xs:attribute type='xs:int' use='required'
                 name='failures'/>

    <xs:attribute type='xs:int' use='required'
                 name='elapsed'/>
  </xs:complexType>
</xs:element>
```

9. The Ping Coordinator is dismissed by the peer it was hired by and sends its resignation to that peer. In theory, another ping request could be sent to this coordinator before it was dismissed and the whole process would start again at step 3.

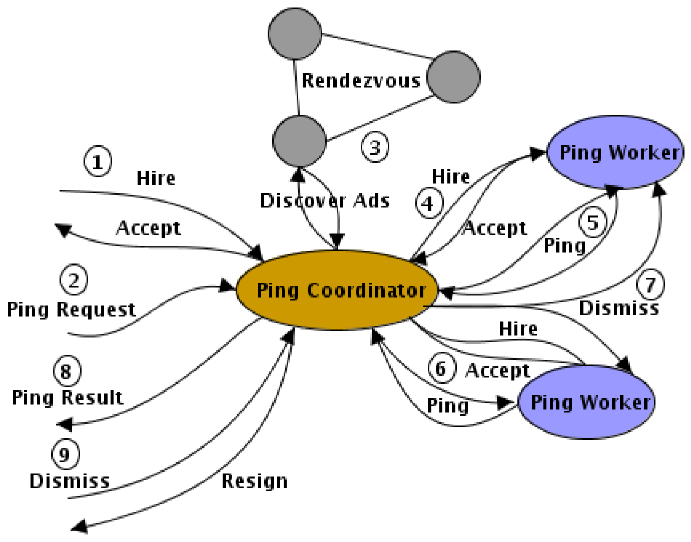

Figure 6. Ping Example

In Figure 6, "Ping Example", the Ping Worker is just an XML Pipeline that echos ping messages. This pipeline is defined as follows:

Example 8. Ping Worker Pipeline

```
<p:pipe xmlns:p="http://www.smallx.com/Vocabulary/Pipeline/2005/1/0"
        name="echo">
  <p:identity/>
</p:pipe>
```

5. Atom and the P2P Storage Problem

Between all of these peers are XML messages with potentially useful information--some of which may be critical for analysis after the computation finishes. In a P2P environment, there is no centralized repository. However, a consumer can define a repository for a requested computation.

If the correspondences were archived between all the peers that participated in computation they would make up a temporal mesh of messages. These messages can be traced between two peers, some of which are responses to other messages, and all of which ultimately lead up to the final result. These messages define a trace of the computation and are important to the analysis of the result.

Many current computation engines only provide a small subset of the intermediary results. In fact, most just provide the input and the output. The advantage in this P2P grid environment is that the messages are composed in XML and, as such, are easy to store. The problem that many in the P2P world are trying to solve is how to create decentralized storage for a decentralized set of peers.

Further, as the messages are all XML documents, they can easily be stored along with metadata about the peer that created them and in response to what stimulus. In some ways they can be viewed as a threaded news feed of computation information--which is a perfect application of a vocabulary like Atom [atom]. The Atom news feed represents the whole computation with metadata about which peers computed what. Which then makes inspection of the result much easier as it would be possible to use new feed readers to inspect results.

6. The Xeerkat Project and Mathgrid.org

An agent API and reference implementation of this P2P grid is available from the Xeerkat Project [xeerkat]. This project uses JXTA to implement a layer on top of the JXTA network. This layer implements the Agent VM which allows hosting of multiple agent roles.

Several agents have been developed for testing this grid. First, the Monos Algebra software [monos] has been integrated to provide the ability to compute algebraic objects relating to monomial and binomial ideals. This software already provides these facilities via XML Pipelines and web services. Previously, the web applications associated with Monos only fed local web services but now they can also feed the grid just by switching the web service to which it sends the request.

A second agent uses the Monos agents to help find counter examples to the Cutting Stock Problem [cutting-stock]. In short, given a set of patterns to cut from some amount stock, you want to arrange the patterns to minimize waste. While this is a hard problem to solve directly, there is a way to estimate the solution for a particular problem. There is a conjecture that the maximum difference between the unobtainable solution and this estimate is less than two. Others believe this is not true.

This cutting stock grid agent runs through a large set of possible cases searching for particular mathematical objects that demonstrate a possible counterexample. While the number of cases is too large to compute all of them, running through a large number of examples will produce data upon which the search can be focused.

At the current time, the Xeerkat project has been deployed at Mathgrid.org with web service gateways at Mathgrid.net. These services provide the ping agent along with the Monos and Cutting stock agents. If peers join the grid with their own agents, those computations would be facilitated as well. They only need to provide gateway services to introduce messages into the network.

The amazing part is that, as a consequence of discovery, scaling the grid system requires only that others install and operate agent host peers. When the agents are installed into this peer, their ad is automatically updated and propagated through the grid economy. In the end, they are eventually discovered by other peers and the gateway. This yields a zero-configuration environment for the host of the grid.

Bibliography

[jxta] *Project JXTA* 2001 http://www.jxta.org

[iri] *IRI* 2005 http://www.faqs.org/rfcs/rfc3987.html

[smallx] *Smallx XML Infoset and Pipeline* 2005 http://smallx.dev.java.net

[xeerkat] *Xeerkat Project* 2005 http://xeerkat.dev.java.net

[monos] *Monos Algebra Software* 2005 http://www.milowski.com/software/monos

[cutting-stock] *Thesis: Computing Irredundant Irreducible Decompositions and Scarf Complexes of Large Scale Monomial Ideals* R. Alexander Milowski San Francisco State University 2004

[stax] *JSP 173: Streaming API for XML* 2004 http://www.jcp.org/en/jsr/detail?id=173

[prog-challenge] *EXPERT OPINION: THE COMING CRISIS IN COMPUTATIONAL SCIENCE* Douglas Post High Performance Computing 13 March, 2004

[groebner] *Groebner Bases: an algorithm method in polynomial ideal theory in Multidimensional Systems Theory* Bruno Buchberger Dordrecht D. Reidel Publishing Company 1985 184-232

[xml-infoset] *XML Information Set* John Cowan, Richard Tobin 2004 http://www.w3.org/TR/xml-infoset

[smallx-pipeline] *Smallx: XML Pipelines Version 1.0* R. Alexander Milowski 2005 https://smallx.dev.java.net/pipeline-spec-2005-06-14.html

[xslt] *XSL Transformations(XSLT) 1.0* James Clark 1999 http://www.w3.org/TR/xslt

[xml-schema] *XML Schema Part 1: Structures Second Edition* Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn 2001 http://www.w3.org/TR/xmlschema-1

[atom] *Atom Syndication Format* M. Nottingham, R. Sayre 2005 http://www.atompub.org/2005/08/17/draft-ietf-atompub-format-11.html

Biography

R. **Alexander Milowski**
School of Information Management and Systems, University of California, Berkeley
Center for Document Engineering [http://cde.berkeley.edu/]
Berkeley
California
United States of America
Smallx Consulting, LLC
San Francisco
California
United States of America

Alex Milowski has been involved in the development of XML and SGML technologies since 1990 — starting with the development of SGML systems for producing EDGAR filings for the SEC. He was involved with the development of XML from the very start and participated on the W3C's committees for XSL and XML Schemas as well as many others. He's the author the of W3C Note on XML Messaging and co-author of W3C Note on SOX (Schema for Object-oriented XML). In 1995 he founded a company called Copernican Solutions--which developed SGML and XML technologies for software systems--and was later acquired, via Veo Systems, by Commerce One. Most recently, he was CTO for Markup Technology, a developer of XML pipelining technology. Currently, he is a member of the advisory board for the Center for Document Engineering at University of California, Berkeley and consults on applications of XML.