DNS Based Remote Programming Objects

This is a system for accessing remote programming objects using DNS to manage the locating as well as the connecting to the remote object. The remote object could be on another server within the current network, or anywhere on the internet.

With current technology an extensive infrastructure is required to find a remote object. With this process, existing DNS servers are used to locate objects and information about objects. This process means no additional infrastructure is needed, and it is platform independent. For example a Macintosh programmer using Objective C could access a .Net object residing on a Windows Server 2008 machine. Furthermore, unlike most existing remoting technologies, this process does not require the user of a remote object to have any prior knowledge of the remote objects structure. The remote objects structure will be returned in response to the DNS Query.

DNS or Domain Name Service is used to match domain names to IP addresses. There are already extensions to DNS to handle other activities. For example RFC 2230 (http://www.rfc-archive.org/getrfc.php?rfc=2230) defines key exchange via DNS. When one looks up a domain, if the entry defines it as needing encryption keys, then a key exchange is initiated. RFC 5702 allows one to use hashing with DNS records. RFC 5507 defines methods for expanding DNS with a new record.

In this process a given DNS record would include information regarding a remote object. That information would include everything needed to connect to that object. So by simply looking up a domain name, such as www.mydomain.com/myobject a client would immediately be connected to a remote object. If security is required, it will be defined in the DNS entry.

Background

There are several methods for remotely accessing programming objects. COM, DCOM, OLE, .Net, Enterprise Java Beans, and CORBA have all been used. Also patent number 5,881,230 dealt with this issue. The 5,881,230 patent was really just an extension of the OLE concept. In that process a specialized application is required to serve programming objects to clients. That specialized application functions as a proxy, serving up proxy objects and interfaces from the remote object. As with OLE and COM each object has a unique ID called a class identifier (CLSID) also referred to as a GUID (Globally Unique Identifier).
The 5,881,230 patent expressly requires an OLE channel, and the object references are OLE object references. It also requires a minimum of 3 tiers: client, remote server (for objects), and remote object. It also requires encryption.
All current methods (OLE, COM, DCOM, CORBA) have one or more of the following problems:

1. Complex to setup. CORBA and Enterprise Java Beans are notoriously complicated.
2. Technology specific: COM/DCOM is a Windows only technology. While Microsoft claims it can be used with any platform, the difficulties in using it alternative platforms are tremendous. Enterprise Java Beans are for Java only.

3. Require additional infrastructure. For example.Net objects require the Microsoft .Net
4. framework.

Relevant Technologies

DNS

Normally DNS records are associated with translating domain names to IP addresses, this is commonly called 'names to numbers'. This is in fact the primary use of DNS records. This type of activity includes the following types of DNS records:

A Records: These translate domain names into 32 bit IP version 4 addresses. This type of record is defined by RFC 1035. An example record would be:

```
example.com. IN A 195.9.60.10
```

The IN stands for internet and the A indicates this is an address record.

AAAA Records: These translate domain names into IP versions 6 addresses. This type of record is defined by RFC 3596. An example record would be:

```
somewhere aaaa 3ffe:1900:4525:3:03d0:09ff:fef6:6d3cv
```

CNAME Records: This is an alias for a given domain name. It is also defined by RFC 1035. An example record would be:

```
mail.example.com IN CNAME mail.example.net
```

Again the IN indicates this is an internet record and the CNAME designates it as a CNAME record. Note that the IN is important since DNS is also used in many systems to lookup local network resources/names.

However there are already DNS record types that provide information beyond translating domain names into IP addresses. Current examples include

AFSDB records: These records provide the location for AFS database servers. AFS (Andrews File System) is designed to aid in sharing files over TCP/IP. These types of records are defined by RFC 1183.

SRV Records: These records allow one to locate the particular host that provides a particular service (such as FTP or HTTP) for a given domain. An example record would be:

```
srvce.prot.name ttl class rr pri weight port target
_http._tcp.example.com. IN SRV 0 5 80 www.example.com.

Or

sip._tcp.example.com. 86400 IN SRV 0 5 5060 sipserver.example.com
```

The srvce segment shows what service will be implemented such as _http - web service
_ftp - file transfer service , _ldap - LDAP service. The prot section gives the protocol such as _tcp - TCP protocol , _udp - UDP protocol. Port is used to define to the symbolic service but does this is not a requirement.

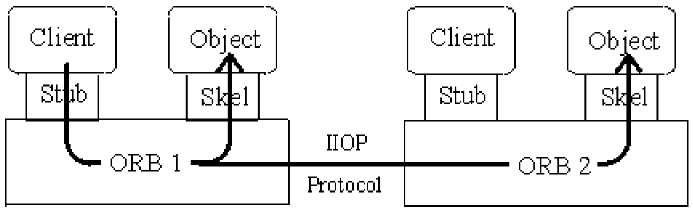
DNSKEY and DNSSEC records are used to handle secure DNS lookups.

What these records show is that there are precedents for using DNS records to do more than simply return the IP address of a given domain name.
A Records, AAAA Records, and AFSDB Records are all considered types of resource records. The first aspect if this process is a new type of resource record. It is conceivable that some embodiments of this process might simply re-purpose existing record types, however the ideal situation would be a new type of resource record.

Current Remote Programming methods

COM, DCOM, CORBA, and .Net Web Services are the most widely used/known methods. So lets begin with a brief overview of how they work and the problems associated.

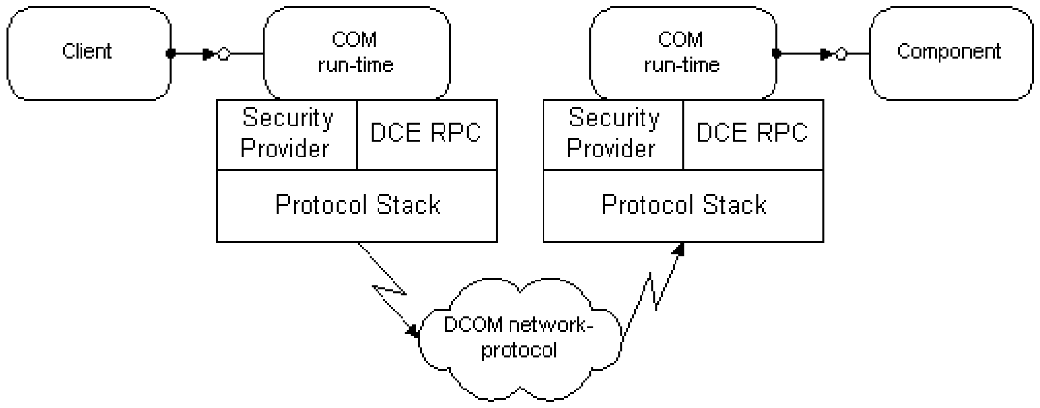
CORBA: Component Object Request Broker Architecture. As the name suggests it depends on a request broker to handle requests from client applications for the server programming object they wish to access. Here is an overview of the process



Note that a stub is used on each end, and the Object Request Broker manages the communication between the object stub and the client stub. Also notice the requirement for a special network protocol IIOP.

COM: Component Object Model is a complex system that is Windows specific. It allows components to register their interfaces, then other components on the same computer can query and utilize them. DCOM or Distributed Component Object

Model takes that process to another level allowing objects to be used across a network. The following diagram, from http://msdn.microsoft.com/en-us/library/ms809340.aspx , illustrates the DCOM Process



Notice that first the COM runtime must exist on both machines. Then a specialized DCOM network protocol is used. Due to the Remote Procedure Call (RPC) requirements, DCOM requires that the operating system on both ends support specific RPC configurations. However, unlike CORBA, it does not require a separate object broker.

Web Services: These are the most easily deployable distributed .Net component. However the depend first on the host computer running Microsoft Internet Information Services as a web server, and having it configured to handle .Net web services. Then the client side must be able to connect to and query web services, this is usually only present in Microsoft specific languages. This methodology is inextricably linked to a specific platform and a very specific type of component server (Internet Information Services).

The 5,881,230 patent: This is an extension of OLE, Object Linking and Embedding. That process actually predates COM and is a very cumbersome method for accessing remote objects. That patent clearly defines the need for an object brokering server and is specifically targeted to three (or more) tiered systems. It also requires (in claim 15 and 16) proxies on both ends of the process, much like CORBA.

The problems with all of these systems are:
1. Many require a specialized intermediary such as the Object Request Broker of CORBA, or the IIS server of .Net Web Services.
2. Some are platform dependent, such as COM and DCOM.
3. Several require special protocols such as DCOM and COBRA

So implementing any of these solutions involves some investment in infrastructure and could tie one to that system. For example if you build a distributed system based on CORBA, it is a non trivial task to convert to DCOM or visa versa.

The 6,851,118 B1 Patent: This is an extension of Java Beans. In claim 1 this patent focuses and tying the client version of the remote object to a specific network adapter. The bulk of this patent is about the generation of the client stub, not about finding the remote object or connecting to it. So this patent covers an entirely different aspect of remoting than does the current process.

The New Process

This new process seeks to solve three problems

1. The ability to easily access remote programming objects with no investment in infrastructure.
2. The ability to easily move from one type of remote programming access to another, with minimal difficulty.
3. The flexibility to implement the process in a variety of ways.

Finding Objects

With this process, in one embodiment a new DNS record type would be created. This would be similar two an SRV record, only instead of pointing a host and a service, it would direct to an object.

1. The DNS record would need to include
   The ip address and port to communicate with

2. The type of object connection. This could be connecting via the new method described in this process (DNS Based Object Connection or DBOC, this is the name of the process itself), or in another embodiment of the process, the DNS record could be used as simply a more efficient way to locate CORBA, DCOM, or Web Service servers. Current options for this are

   a. DBOC (this is default, if you leave item 2 blank, it assumes DBOC)
   b. CORBA
   c. DCOM
   d. Web Service (.net)
3. Authentication. Current options are
   a. None
   b. Hashed Username/Password

4. Security. Current options are
   a. None
   b. SSL

An example would be:

```
object.mycompany.com 192.100.100.44, dboc, none, ssl
```

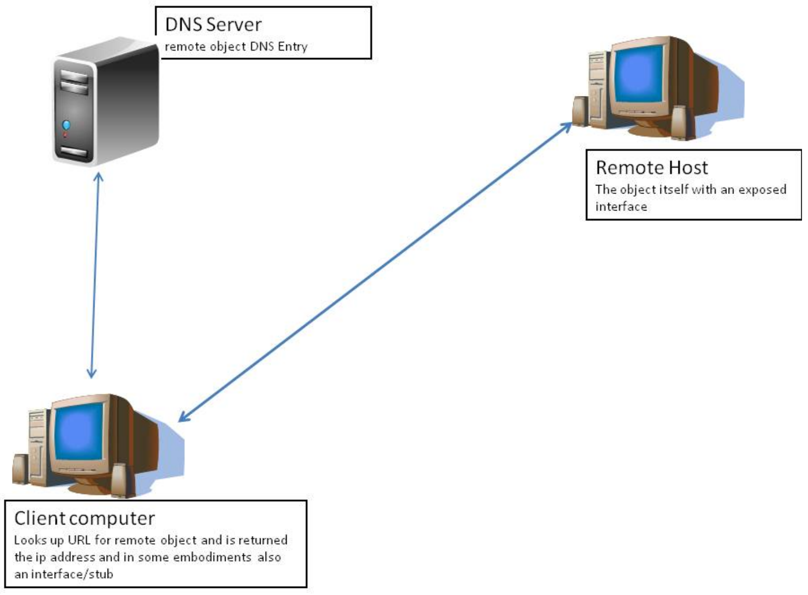
This defines remote objects are available at object.mycompany.com URL/ IP 192.100.100.44, they are to be access via the dns based object connection, no authentication is needed, and ssl is required for transmission.

In another embodiment of the process, in which the DNS record is simply used to locate an CORBA, DCOM, Web service, or other type of remote object server then the entry would look like this:

```
object.mycompany.com 192.100.100.44, CORBA
```

As you can see in this embodiment of the process the client is returned the IP address, and an identifier telling the client that the IP address is for a CORBA object server.

The actual communication would work like this:


DNS Server
remote object DNS Entry
Remote Host
The object itself with an exposed interface
Client computer
Looks up URL for remote object and is returned the ip address and in some embodiments also an interface/stub

General overview of the process

The DNS lookup is much like a standard DNS lookup only additional information is returned to the client. As we have already seen, expanding DNS records to return additional data has precedent.

Getting an interface to the remote object.

Other methodologies such as DCOM and CORBA require the programmer using the remote object, to know details about that object, and to be able to create a stub for that object. The current process also corrects this short coming. In this process the stub object is returned in one of the following formats:

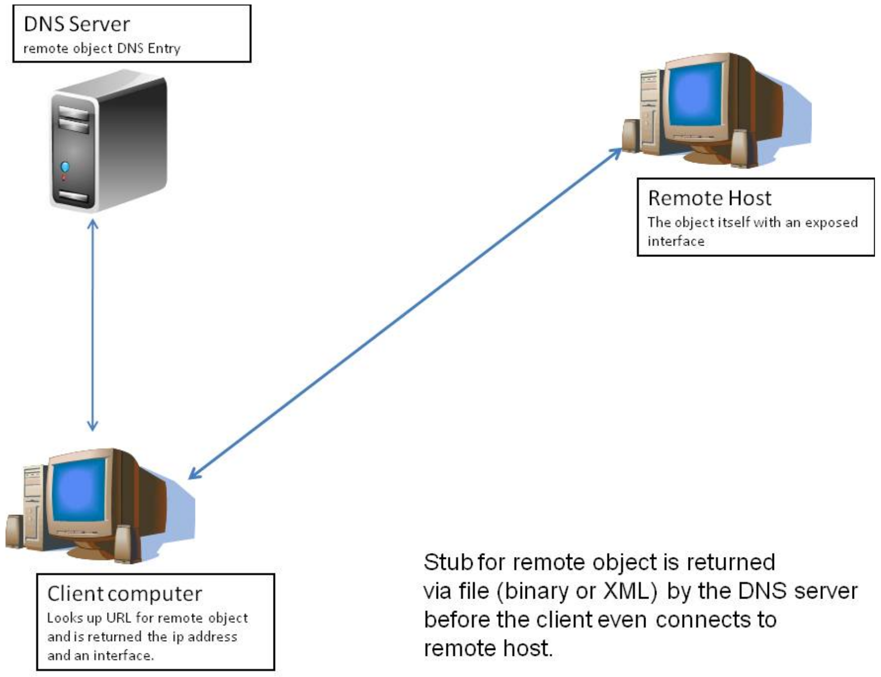
1. Binary
2. XML

This allows the client to use its preferred communication method to receive the stub structure and then utilize that structure. Other formats are certainly possible with this process, but these are the two first being considered.

In one embodiment of the process the stub structure is returned by the DNS server in response to the DNS query. This would require the DNS server to store a file (for example an XML file) that contains that structure. Then the DNS entry would require an additional field denoting that file. The DNS entry is shown here:

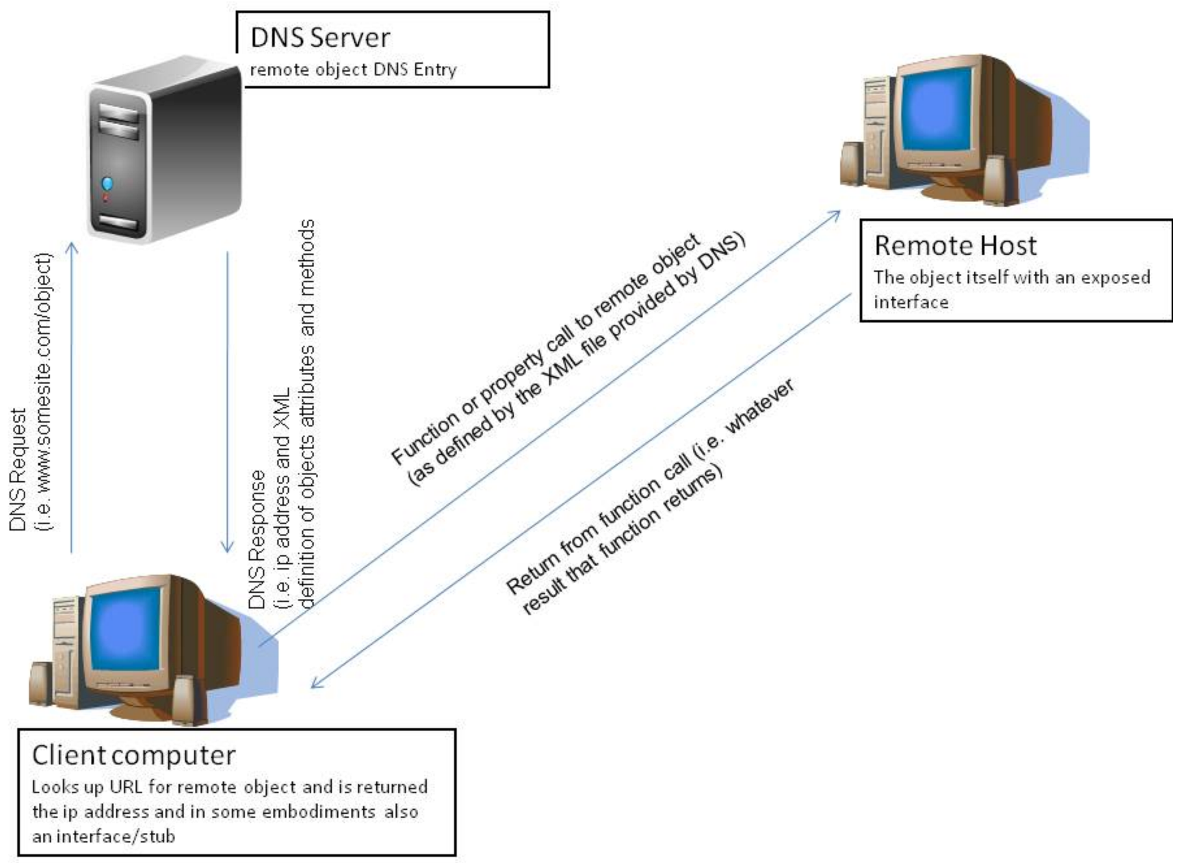
```
object.mycompany.com 192.100.100.44, dboc, none, ssl, interface.XML
```

The actual communication overview is shown here:


DNS Server
remote object DNS Entry
Remote Host
The object itself with an exposed interface
Client computer
Looks up URL for remote object and is returned the ip address and an interface.
Stub for remote object is returned via file (binary or XML) by the DNS server before the client even connects to remote host.

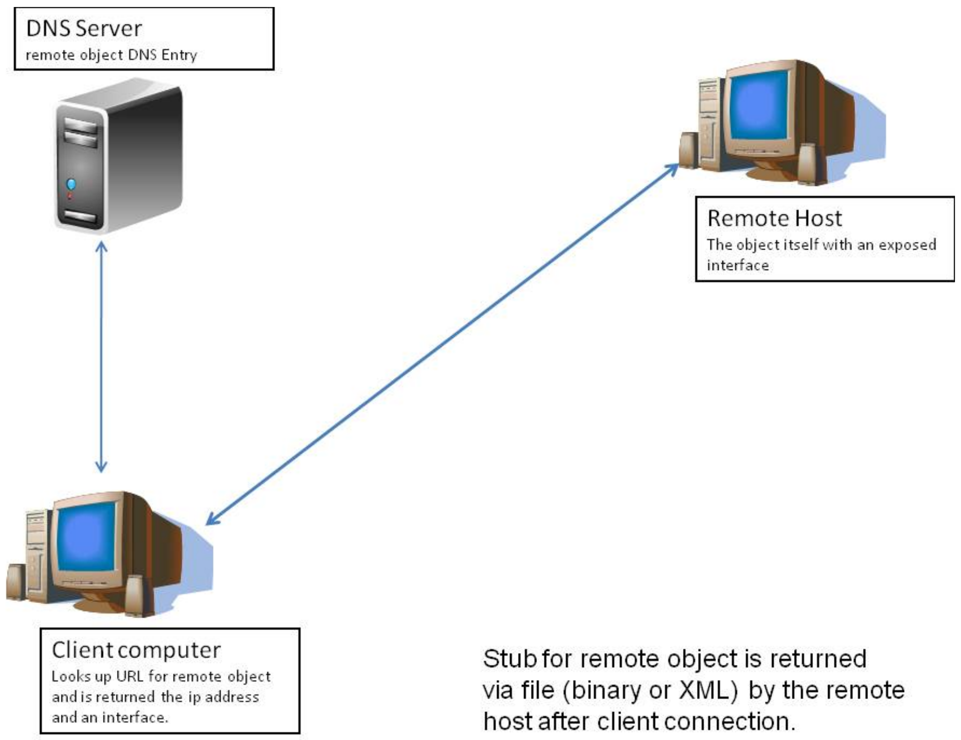
Embodiment 1: DNS Server returns stub

The following image gives more detail on the process.


DNS Server
remote object DNS Entry
Remote Host
The object itself with an exposed interface
DNS Request (i.e. www.somesite.com/object)
DNS Response (i.e. ip address and XML definition of objects attributes and methods
Function or property call to remote object (as defined by the XML file provided by DNS)
Return from function call (i.e. whatever result that function returns)
Client computer
Looks up URL for remote object and is returned the ip address and in some embodiments also an interface/stub

Detailed Overview 1

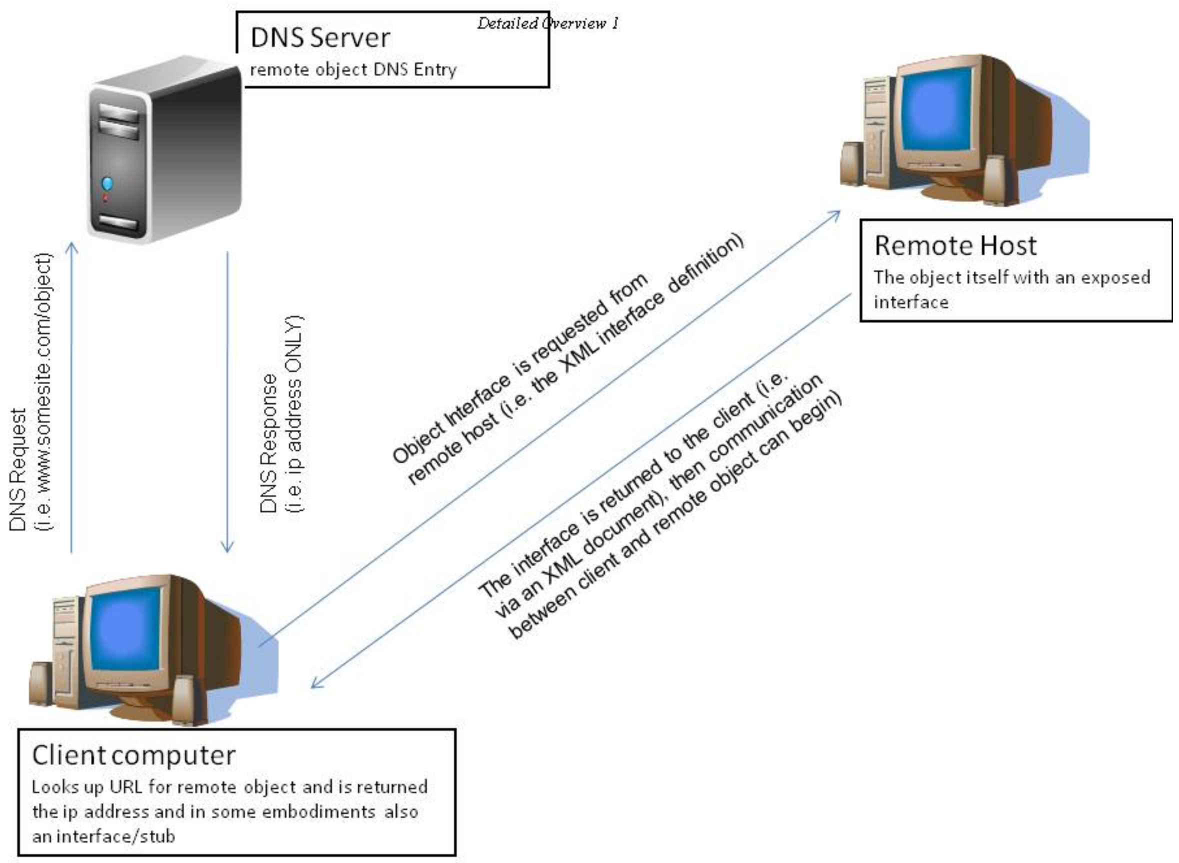
However in another embodiment of the process the DNS server simply returns the IP address, when the client connects to that address the stub structure is returned (again as a file, such as an XML file or in binary format).


DNS Server
remote object DNS Entry
Remote Host
The object itself with an exposed interface
Client computer
Looks up URL for remote object and is returned the ip address and an interface.
Stub for remote object is returned via file (binary or XML) by the remote host after client connection.

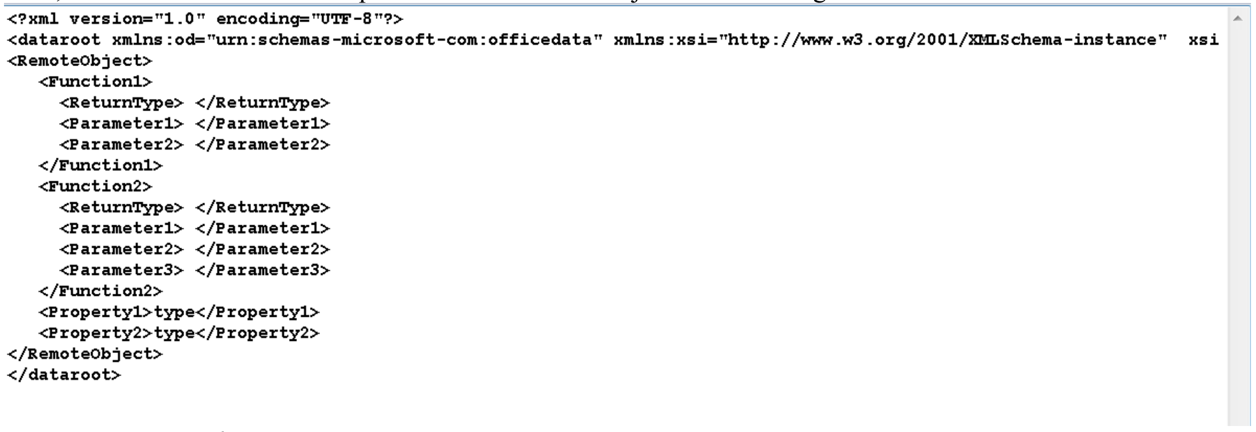
Embodiment 2: Remote host returns stub

The following image shows this in more detail:

Detailed Overview 2


Detailed Overview 1
DNS Server
remote object DNS Entry
Remote Host
The object itself with an exposed interface
DNS Request (i.e. www.somesite.com/object)
DNS Response (i.e. ip address ONLY)
Object Interface is requested from remote host (i.e. the XML interface definition)
The interface is returned to the client (i.e. via an XML document), then communication between client and remote object can begin)
Client computer
Looks up URL for remote object and is returned the ip address and in some embodiments also an interface/stub

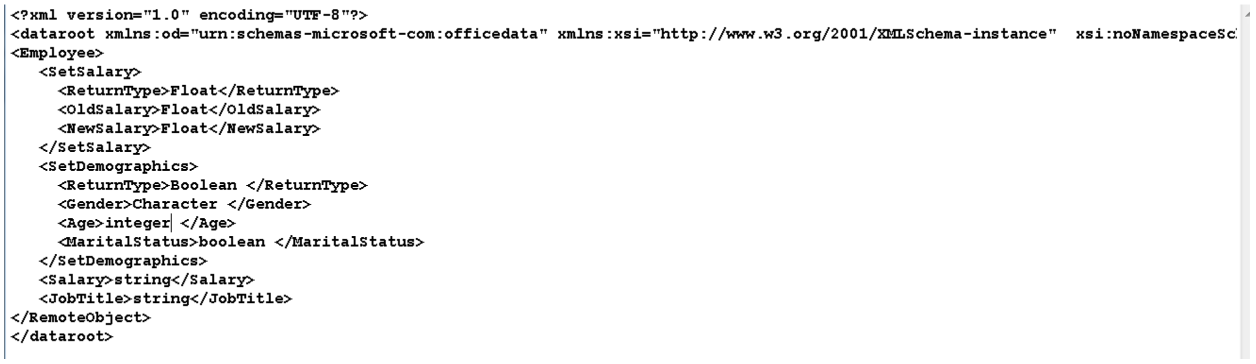
In either case the details of the remote object interface could be returned to the client computer in many different forms. XML is one form, the one described in this document. However it could also be returned in a pure binary form, or in other forms. An example of an XML remote object definition is given here:



```
<?xml version="1.0" encoding="UTF-8"?>
<dataroot xmlns:od="urn:schemas-microsoft-com:officedata" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi
<RemoteObject>
   <Function1>
     <ReturnType> </ReturnType>
     <Parameter1> </Parameter1>
     <Parameter2> </Parameter2>
   </Function1>
   <Function2>
     <ReturnType> </ReturnType>
     <Parameter1> </Parameter1>
     <Parameter2> </Parameter2>
     <Parameter3> </Parameter3>
   </Function2>
   <Property1>type</Property1>
   <Property2>type</Property2>
</RemoteObject>
</dataroot>
```

Generic XML Template

To give a more concrete example, assume the remote object is an employee object, the interface returned might look like this:


```xml
<?xml version="1.0" encoding="UTF-8"?>
<dataroot xmlns:od="urn:schemas-microsoft-com:officedata" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:noNamespaceSc
<Employee>
   <SetSalary>
     <ReturnType>Float</ReturnType>
     <OldSalary>Float</OldSalary>
     <NewSalary>Float</NewSalary>
   </SetSalary>
   <SetDemographics>
     <ReturnType>Boolean </ReturnType>
     <Gender>Character </Gender>
     <Age>integer </Age>
     <MaritalStatus>boolean </MaritalStatus>
   </SetDemographics>
   <Salary>string</Salary>
   <JobTitle>string</JobTitle>
</RemoteObject>
</dataroot>
```

Specific XML Example