# Open Archives Initiative

**OAI Project**
The Open Archives Initiative (OAI) organization is a project funded by several organizations and driven by several committees - it currently has a steering commitee, a technical committee, and another committee. each of these committess is comprised of people from around the globe who represent a wide variety of companies and backgrounds. it was created in XXXX, and it is funded by three organizations - the Digital Library Federation, Coalition for Networked Information and National Science Foundation.
Initially for a e-Prints community.

The phrase for the OAI initiaive is this:

"The Open Archives Initiative develops and promotes interoperability standards that aim to facilitate the efficient dissemination of content."

The goal of the project was to release a lightweight protocol which could be used to find out information about resources which were traditionally 'locked' away in websites, meaning that they were hard to get at or discover. The initial field was for e-print repositories, but the goals have been expanded to any type of information.

The first release of the protocol was released on January, 2001. Since then the protocol has gone through several revisions - version 1.1 was released in July 2001 and the last one, version 2.0, was released on June 2002. The idea was to create an open; lightweight protocol.

**OAI Protocol**
The OAI protocol defines a simple, XML-based structure for the data messages which are passed between server and client and specifies the use of HTTP for the message transport mechanism.

These were good choices for the protocol designed to have a low barrier for entry.

First, HTTP is the same transport method web browsers use to get web pages. HTTP has been available since the early days of the modern Internet and most networks, by default, have been configured to allow this particular protocol travel freely. This is also a 'trick' used by most new Internet services.

Second, XML itself is not a markup format, but rather defines rules for markup formats. Any resulting format is based on text instead of a more complex binary format. This means that any computer ever created can read it as a data format and also that it is human readable as well using no special tools other than a text editor. Additionally, there are free XML-processing libraries available for every major language, so it's relatively easy to extract the information from the XML format.

**Data exchange**
Shortly we will show an example of an OAI protocol request and result. But for the instant we will talk about the next level of information. what exactly is this data passed from server to client?
XML can represent structured data in a simple manner. When an OAI client requests information about a server's resources, the server will respond with the data markedup in the OAI format. It will contain *metadata* concerning its resources, and not the resources themselves. Metadata is in fact information about data (or resources) and could include who created the resource, its publishing rights, the date it was created, and even how a user might finally obtain it.

there are potentially hundreds or thousands of different types of metadata, each concerned with its own aspect of resources. some were discussed earlier in this paper (ELM, SCORM, LOM). These are mentioned because they were created to describe educational resources and thus are more aligned for our uses.

However the OAI specification and protocol are not limited to any particular type of metadata. But seeing that there are so many available, the committee decided can diseminate any type of metadata. However, the Dublin Core metadata format is *required* for all resources. This means, at a minimum, you can gather (and must publish) the Dublin Core metadata for a resource.

The Dublin Core is perhaps the most basic of metadata and can be applied to *any* type of resource (film, pdf, documents, recordings, etc) . This format has 15 basic fields:

| | | |
|---|---|---|
| title | contributor | source |
| creator | date | language |
| subject | type | relation |
| descripton | format | coverage |
| publisher | identifier | rights |

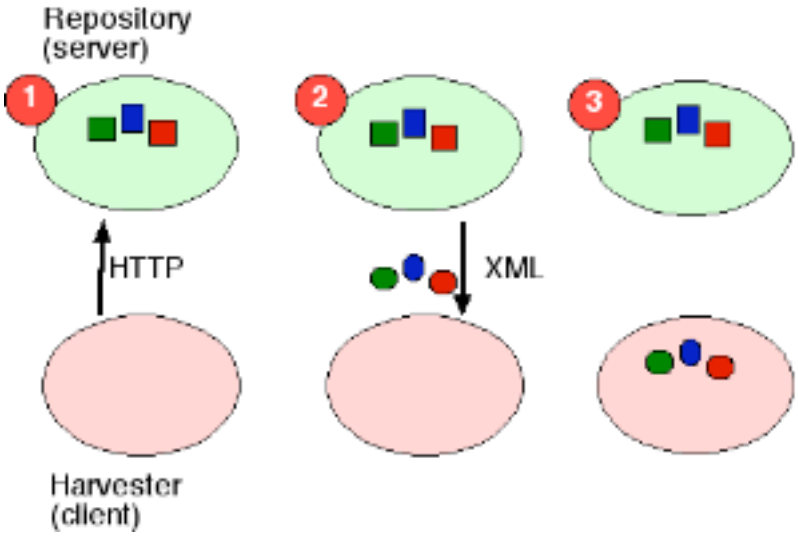
The DC can be extended to suit other needs, but more generally it is incorporated or used along side other metadata formats.

**OAI Further Explained**

Up until now we have talked about the purpose of the OAI organization, the protocol, which is to diseminate metadata about resources, and not resources themselves. The next step is to talk about how the OAI protocol functions and how metadata actually gets shared.

**Harvester and Repository**

First off, in the OAI specification, there are three possible items that can play a part in the OAI structure. The first two are the:
- Harvester
- Repository



In the diagram above outlines the relationship between a harvester and a repository.

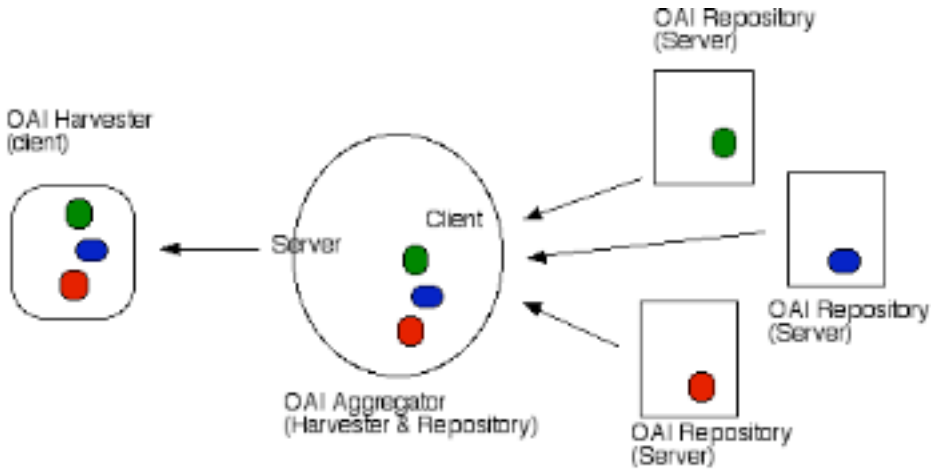
1. in step one the harvester (client) sends a request for information to the server. That is the client is responsible for issuing requests to the server and in general wants to know what resources (represented by colored squares) the server has.

2. in the second step, the server responds to the request by giving the harvester information *about* the resources that match the clients request. the client can ask for information pertaining to all of the resources, or the resources with different types of markup, or the resources that have been edited within a certain time frame among other types of requests.
this metadata is represented by rounded squares. they are a different shape to show that what the client gets is not the resources, but information about the resources.

3. in step three, the client has received and processed the *metadata* records it received from its request. at this point it can do whatever it needs to with them, depending on what type of client it is.

The client gets its name from the verb to harvest, which means to collect or to gather. Something that does this is a harvester. The server fills the other role, of what the client talks to. the server receives requests, but ultimately its job is to give information.

To clarify the roles of the harvester and repository, in tech speak these would be referred to as a client and server, respectively. To take this metaphor a little further, these two things could be viewed like a web browser and web server. The harvester is like the web browser - you put in a request (a URL) and receive web pages and photos. The repository is like the web server - it receives URL requests and gives web pages and photos. The server is a storage for resources and the respository (something that contains; and disperses information). It serves OAI records depending on the request from OAI clients. this could be thought of as a web server.



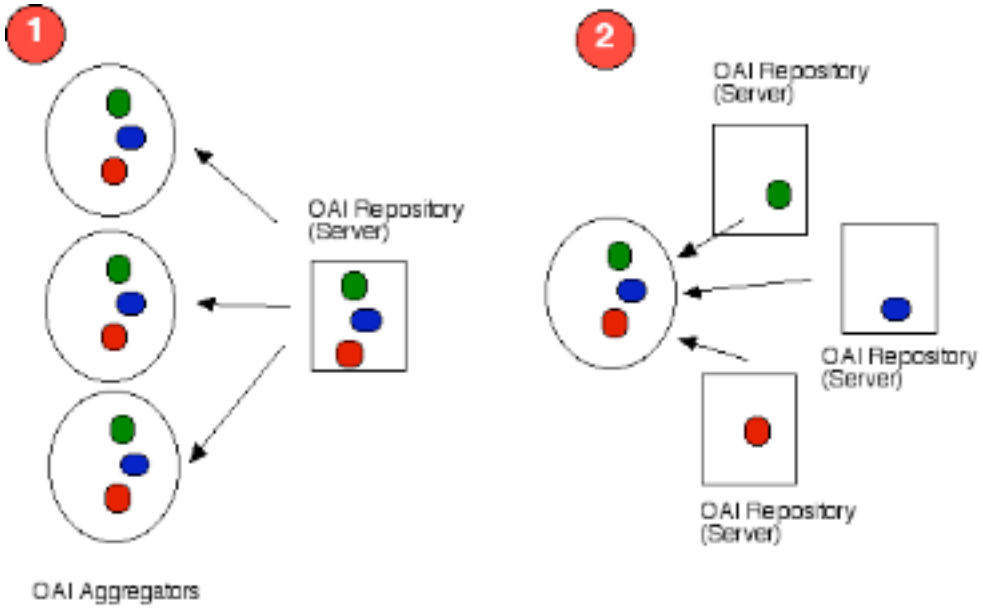

**Aggregator**

The third and final possible component of the OAI structure, the aggregator, is actually a special case because it is comprised of the first two components we discussed. that is to say that performs the tasks of a harvester *and* repository and the same time.
however it differs from a pure respository in that the aggregator doesn't contain any resources itself, but rather it stores information that it has collected from *other* OAI Servers.

this information is the metadata about the resources, but not the resources themselves. We know

where the original resources are located and how to retrieve them from the information stored in the metadata records. it's up to the oai client to do something intelligent with this information.

**aggregator uses**




the aggregator was designed to to two things - collect information from many places and then distribute it.

two ways of using an aggregator are shown above:
1. in number one, the first usage is used as a way to help a single OAI repository from becoming overloaded with requests. In this scenario we have several aggregators in front of a single repository. each aggregator harvests ans stores the metadata records from the repository. next, it in turn serves out requests which come from OAI harvesters or clients. in this particular diagram the aggregators are only replicating one single site.

This is beneficial because the company with the respository doesn't have to buy a single, large computer to handle the load, because at some point it's expensive to buy a computer with these capabilities. So instead they can put up many aggregators using inexpensive servers. these aggregators can respond to requests while they periodically update themselves with the main server. As this site becomes more popular, it's very easy to add another aggregator to the chain. this is a classic strategy used by big websites to handle the massive loads coming from their customers.

2. however aggregators are not limited to replicating just one site. as shown in the second setup they can replicate many sites. in this diagram we show a single aggregator which is harvesting information from several repositories. because the aggregator only knows the repositories that it has collected metadata records from, this can serve as an aggregator which contains highly focused content. that is, the repositories which are harvested by the aggregator are carefully chosen for their content - they contain content which is very pertinent to the people who will use the aggregator.

you can have a mix of the different setups if necessary.

this use more along the lines of how we use the OAI system. we have a single aggregator for a

campus. we add oai repositories from people who publish, or areas within our website. we don't expect a lot of accesses so a setup our use is more along these lines. so the sites that we incorporate are those which we know have pertinent information for our professors. some examples might be several publishers who create educational materials, libraries of interest, or other teacher community websites who publish their matierals using OAI protocol.
this means that our classeur searches over a single OAI aggregator. we want our classeur to look at a single aggregator for its searches, but we want to haec information from various sites. if there is another site which is of interest, we can easiliy add it to the aggregator. likewise, if there is a site contained within, we only have to delete the site. we only have to change the information in one location.
in short the university faculty can make searches over many different sources of teaching resources, but in one step.

**Technical OAI stuff**

So, what does an OAI client request look like? here is a simple one:

*http://oaiserver.com/oai?verb=ListRecords&metaPrefix=oai_dc*

Notice that this looks oddly like a URL you would pass in a browser, as it was designed to be. before we mentioned that the requests use the HTTP protocol, the same as a web browser. actually, you can test OAI servers by putting the requests in a browser and get a response back from a server.

here is an incomplete server response which shows a partial structure of the XML which could be returned.

```
<?xml encoding="UTF-8" ?>
<OAI-PMH>
<header>
...
</header>
<records>
  <record>
blah blah
  </record>
</records>
</OAI-PMH>
```

that the markup is relatively straightforward. it can be read easily by a human ( important during testing ).

**Creating our own**

Why did we decide to create our own OAI implementation when there were already several harvesters, repositories, and aggregators to choose from? The problem was that there weren't any components available which were written using Python, the core language of Zope. Nevertheless we downloaded several servers to test and had many discussions debating whether to go with something available or something in house. There were several things to consider:

1. We had to weigh the difficulty of us using and installing yet another piece of server software alongside those we already had in place. Every new piece of software in a system means time used to learn and know something else. it takes time away from something else. and introduces

another point of failure. it's harder to fix something if you know nothing about the system.

An even larger problem is that none of the components we found included searching capabilities. Of course this is to be expected because searching is not part of the OAI specification. However, it's *was* necessary for our client tool, the Classeur de Reference. With an entirely new system, this meant that we would have to figure out how to bolt on searching within a framework we didn't know. with tools we weren't aware were available (meaning we'd have to search for something that works). Since all of the others are written in different language, and neither were a strong point for any member of our team, we would have had to learn and then adapt anything to our needs.

Searching using Zope is painless to implement, plus we are a Python/Zope house so nothing would be new for us to implement. we have implemented a simple searching facility built with Zope for our version 0.1. In the future we hope to incorporate a more robust searching system which will allow users to fine tune their searches. taking advantage of the metadata contained within the servers.

There were several needs, the most important being:
- make it easy to publish resources
- make the database searchable

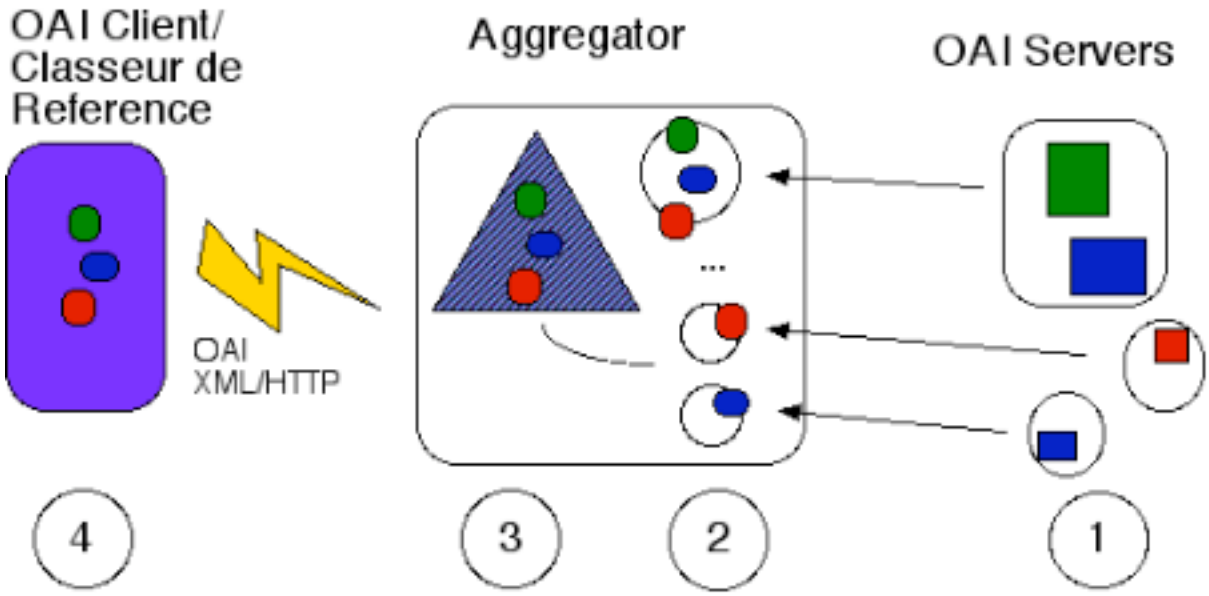
Already there were compelling reasons for us to go our own way. We know the system. we have the servers. we have the expertise. we know using zope can be painless for the users. we can make the database searchable.
The overall architecture of our system makes use of several features of Zope and without them, putting together the system would have been much more difficult.

**ZCatalog**
The ZCatalog is a Zope object which enables web programmers to quickly setup search capabilities. The ZCatalog has indexing functionality indexing and searching within Zope. The ZCatalog creates and stores *references* for each object cataloged. While they are *not* the original object, they can can be used to obtain the original if necessary.
Also this tool facilitates some of the more obscure notions of the oai spec, including resumptionTokens.

**Dublin Core**
All objects added within the Zope Object Database (ZODB) have Dublin Core metadata as part of their basic structure. it's impossible to have an object within zope which doesn't have Dublin Core metadata - ie, it is at the very heart of Zope. this is great because DC is a *required* metadata format for all records within a repository.We take advantage of this fact when doing automatic publishing with the ZOAI Server. we go over this a little deeper later.

Of the two main objects we have in our system, the aggregator and server, i will introduce the OAI Aggregator first, because the ZOAI Server is conceptually the same. Once we explain this, you will see that, using the coolness of object oriented techniques, not a whole lot had to be changed in order to have both types of objects.

# OAI Agregator Architecture




**How It Works**

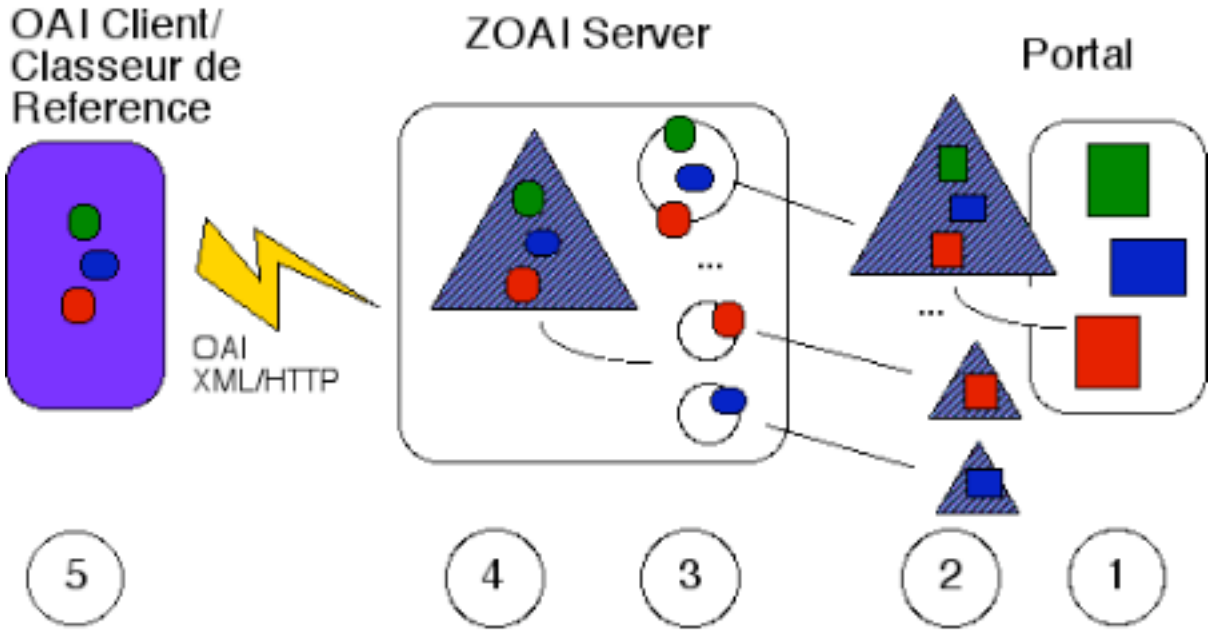
1. In this step, are the right hand side of the diagram we see various OAI repositories or aggregators which are located somewhere on the network or out on the Internet. As noted before, OAI aggregators and repositories serve *records* of metadata of resources, and not the resources themselves.

2. There are two parts of the system to create a Zope OAI Repository, whether or not is it a regular repository or one in an aggregator.
in both cases you have data coming into the system, and then data going out. in the case of the aggregator the data coming into the system is from other OAI repositories. the information going out is the information gather, but it is going out through the left hand side of the diagram. After we have added an aggregator, we need to add Harvesters. There will be one harvester for each server we want to get data from. each of these harvesters will create their own OAI records and store them. each harvester is responsible for communication with an oai server - this includes registering errors, putting together partial responses.

3.When each record is created, the harvester tells the aggregator that it exists and the aggregator then catalogs it for faster searching. after this, the aggregator is concerned with responding to oai requests iteself. it uses the catalog to find quick results quickly, then puts them all together into the XML response.
A special thing about our servers is that they also can do more refined searches that a standard OAI respository. this is necessary from a user point of view to be able to find interesting information. there are no ‘regular’ search facilities defined in the OAI specification.

4. We have our classeur de reference. it communicates with the oai server via the search interface initially. this is described elsewhere.

# ZOAI Server Architecture




**How It Works**

The operation of this system is exactly like the aggregator. it differs slightly in how we get information into the system.

1. In the first step, resources (various documents) are created in the portal software by users. As stated above, when objects are created within the ZODB they automatically are marked with Dublin Core metadata such as the Creator, Date of Creation, Title, ID, etc.

Optionally, using another component of this system, other types of metadata can be added to each document. These could be formatted LOM, SCORM, ELM, etc.

2. When objects are added to a Zope server, they can be cataloged in one (or more) catalogs we use for searching, etc. This is the best case because instead of searching all over the portal for objects, we just look in the catalog and it can tell us which objects exist. In our case, our Portal software uses several catalogs, however only one of them is of interest for the ZOAI Server. It is the main catalog which holds all of the important user objects.

3. When a ZOAI server is added it is empty of any data. At this point it can respond to OAI requests, however most of them will return notification that the database is empty. a ZOIA server is added, ZCatalog harvesters are added inside of it. One for each catalog to be searched. These can either be set to take all objects which are indexed (automatic publishing), or look for those who have been specifically marked for sharing (manual publishing).
Records are created for each object which is close to the actual OAI format.
These records also serve to enable ‘deleted record’ support in the OAI server. This isn’t possible using the regular catalogs because once an item has been deleted, all information about it in the catalog is removed.

4. The ZOAI server itself has its own catalog. This allows searching. It takes the OAI format from

# Links

**Open Archives Initiative**
http://www.openarchives.org/

**Dublin Core Metadata Initiative**
http://purl.org/dc/

**Zope web-application server**
http://www.zope.org/

**Zope ZCatalog**
http://www.zope.org/zcatalog/