7. LINEE GUIDA PER IL SERVICE PROVIDER

7.1 INTRODUZIONE

Questa sezione dell'elaborato, che si basa sulle "Linee Guida del Service Provider" del sito ufficiale dell'OAI [S16] e di quello dell'OAForum [S17], tratta le *linee guida* utili per i gestori e gli implementatori degli harvester dei service provider.

Il protocollo OAI-PMH pone le basi della sua progettazione sugli harvester per semplificare l'implementazione dei repository dei data provider. Un esempio può essere il fatto che un repository in generale può adottare una granularità per il datestamp di tipo YYYY-MM-DD oppure YYYY-MM-DDThh:mm:ssZ o entrambi, di conseguenza l'harvester dovrà supportarle tutte per poter far fronte alle varie situazioni.

Gli argomenti che verranno trattati nel capitolo sono i seguenti:

- Requisiti
- Componenti ed architettura del service provider
- Software di raccolta automatizzati
- Datestamp e granularity
- Set
- Controllo di flusso, Load Balancing e Reindirizzamento
- Liste incomplete di risposta e resumptionToken

- Compressione della risposta
- Come raccogliere tutti i metadati da un repository
- Testing
- Registrazione

In conclusione, questa parte dell'elaborato rappresenta una sezione di utilità per chiunque voglia implementare il proprio service provider e allo stesso tempo un compendio per le parti del protocollo OAI-PMH che riguardano la raccolta dei metadati.

7.2 REQUISITI

I requisiti tecnici necessari all'implementazione di un service provider per la raccolta dei metadati da data provider, attraverso il protocollo OAI-PMH e per la conseguente fornitura di servizi a valore aggiunto, sono i seguenti:

- *Server connesso ad Internet* – l'harvester del service provider, che costituisce un portale di servizi a valore aggiunto per l'utente finale.
- *Sistema di database* – un database relazionale o XML opportunamente progettato per contenere i metadati raccolti dai vari repository.
- *Meccanismo di Resumption Token* – una componente che permette di gestire liste incomplete di risposta inviate dai data provider in replica a richieste che richiedono degli elenchi di record.
- *Ambiente di programmazione* – un ambiente di programmazione capace di effettuare le seguenti operazioni:
    - Richieste HTTP ai server web

- Richieste a database server
- Parser XML

7.3 COMPONENTI ED ARCHITETTURA DEL SERVICE PROVIDER

I componenti di un service provider sono i seguenti:

- *Gestore dei data provider* – software che si occupa della gestione dell'insieme di data provider da cui effettuare le raccolte di metadati. Le voci della lista contenente tali repository possono essere aggiunte o modificate manualmente oppure automaticamente attingendo direttamente dal registro ufficiale dell'OAI.
- *Elaboratore di richieste* – software che crea richieste HTTP e le invia ai repository conformi all'OAI. Per effettuare tali richieste di metadati esso utilizza i verbi specificati dal protocollo OAI-PMH. Può anche effettuare raccolte selettive utilizzando il parametro *set* o i parametri *from* e *until*.
- *Scheduler* – software che permette la raccolta a intervalli regolari di tempo dagli archivi preventivamente specificati dall'utente. Esso permette di limitare l'intervento manuale dell'utente automatizzando le operazioni di raccolta. Un esempio può essere l'uso di un demone a tempo, tipo l'applicazione "cron" di Unix.
- *Controllore di flusso* – software che permette di formulare delle richieste per delle liste incomplete di risposta a seguito del partizionamento della stessa da parte del data provider. Esso utilizzerà i resumption token via via ricevuti per ottenere tutte le liste incomplete. Nel caso in cui si verifichi un errore HTTP 503 (servizio non disponibile), esso analizza la risposta per

estrarre il periodo di tempo (retry-after) da attendere prima di effettuare nuovamente la richiesta.

- *Meccanismo di aggiornamento* – software che si incarica di effettuare l'aggiornamento dei metadati presenti nell'archivio dell'harvester. Esso permette due possibili alternative:
    - Cancellare tutti i metadati marcati come "vecchi" prima di raccoglierli nuovamente.
    - Effettuare un aggiornamento incrementale attraverso il parametro from. Successivamente, utizzando lo unique identifier di ogni record per verificare se esso è già presente o meno, si aggiungono i "nuovi" metadati e si sovrascrivono quelli marcati come "modificati" o "cancellati".
- *Parser XML* – software che analizza le risposte ricevute dai repository validandole tramite un XML schema e che trasforma la parte della risposta relativa ai metadati, codificati anch'essi in XML, nella rappresentazione interna del sistema di memorizzazione.
- *Normalizzatore* – software che trasforma i dati rappresentati nei diversi formati in una struttura omogenea, rappresentando internamente le diverse informazioni, quali ad esempio date, autori, codici di lingua, ecc., in tale struttura. In tal modo esso può mappare o tradurre tra differenti formati di rappresentazione.
- *DBMS (database management system)* – software che si occupa della gestione del database, ricevendo i risultati di output dal normalizzatore e mappando la struttura XML del metadato nella struttura record del database relazionale che gestirà valori multipli di elementi. Nel caso in cui si utilizzi un database XML non sarà necessario effettuare tale mappatura.
- *Controllore di duplicazioni* – software che si occupa di raggruppare record

identici provenienti da data provider differenti. Ciò potrebbe essere realizzato basando l'implementazione sull'identificatore unico degli item relativi ai record raccolti. Tale soluzione tuttavia può non essere totalmente esente da rischi o errori.

- *Modulo dei servizi* – software che fornisce i servizi all'utenza finale. Tali servizi vengono forniti in base ai record di metadati precedentemente raccolti dai vari repository e dunque localmente accessibili, evitando di consultare i data provider durante tali operazioni (il che sarebbe totalmente contrario all'utilità della raccolta centralizzata, soluzione che ha dato luogo alla nascita degli open archive).

L'architettura di un service provider è rappresentata nella figura 1:

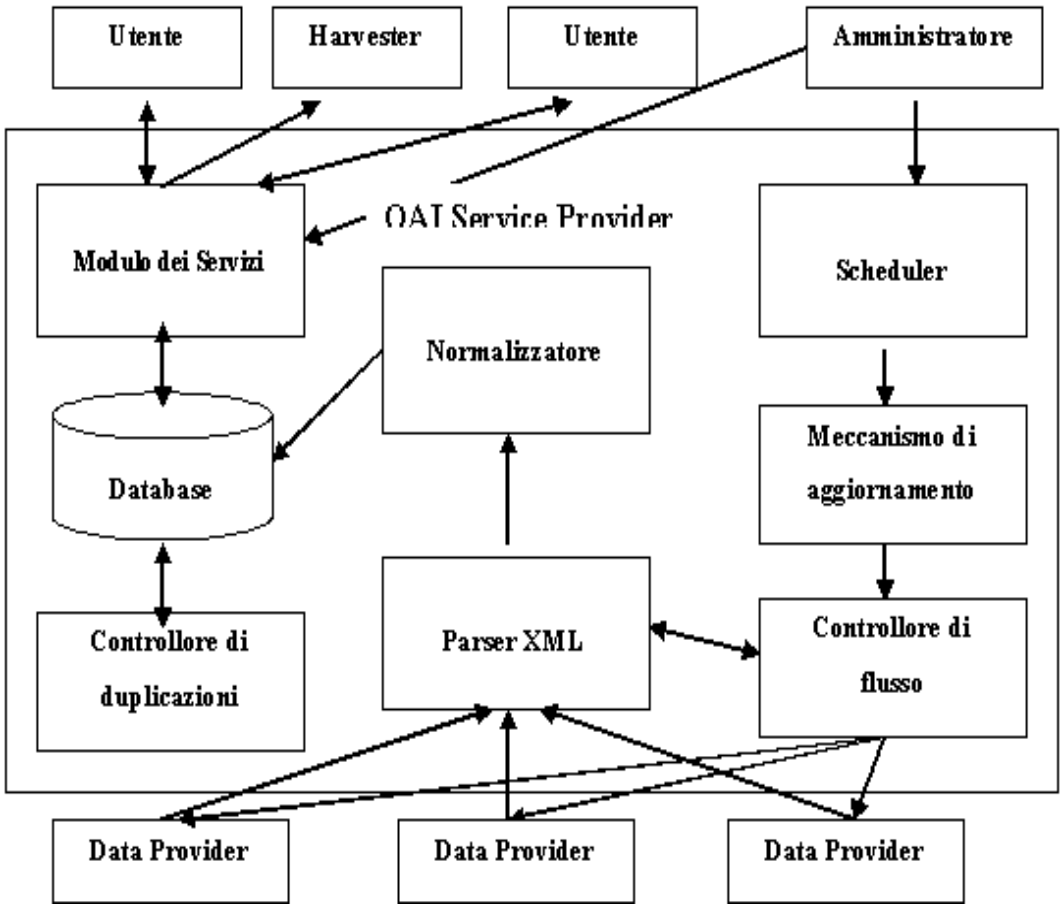

Figura 1 – Architettura di un Service Provider

7.4 SOFTWARE DI RACCOLTA AUTOMATIZZATI

Gli harvester possono essere visti come programmi di tipo client che eseguono

delle richieste periodiche in maniera automatizzata nei confronti dei diversi repository, similmente ai "web robot" (per approfondimenti sui web robot consultare il sito http://www.robotstxt.org/wc/robots.html).
Per tale motivo, nella progettazione di tali programmi dovrebbe essere posta particolare cura per fare in modo che il software si comporti correttamente in funzione delle risposte ottenute dai data provider, onde evitare di rallentare le performance di questi ultimi.
Si raccomanda infatti che i test effettuati su nuovi software di raccolta o su nuove installazioni includano dei controlli per assicurare che risposte o condizioni di errore inaspettate non portino a tentativi di richiesta reiterati in rapida successione. Quindi, questi software di raccolta dovrebbero "autoterminarsi" nel caso in cui ricevano un codice di stato HTTP 403 o altre risposte inaspettate.
Poiché le interfacce OAI-PMH dei repository sono create specificamente per essere accedute dai software di raccolta automatizzati, non è diffuso, a differenza di quanto avviene per i web robot, l'uso del file /robots.txt standard per consentire o vietare la raccolta. Di conseguenza non è previsto che gli harvester consultino tale file.

Gli harvester OAI-PMH dovrebbero aderire alle regole standard HTTP 1.1 fissate per i campi header di richiesta [S17]. Tali campi permettono al client di fornire informazioni aggiuntive al server, circa la richiesta e il client stesso che la ha inoltrata.
In particolare, i campi header di richiesta che devono essere specificati sono i seguenti:

- *HTTP User-Agent*
- *HTTP From*

Entrambi verranno trattati nei paragrafi seguenti.

7.4.1 Header HTTP User-Agent

Un'applicazione di tipo client che permette l'invio di una richiesta viene comunemente definita *user agent*. Esempi di tali applicazioni sono i browser web, editor, spider (robot che attraversano il web) e gli altri strumenti utilizzati dall'utente finale.
Il campo header di richiesta User-Agent contiene informazioni riguardanti l'applicativo user agent che ha originato la richiesta. Solitamente tali informazioni vengono utilizzate per scopi statistici, tracciamenti di eventuali violazioni del protocollo, riconoscimento automatico dei vari user agent che si connettono.
Le applicazioni user agent dovrebbero includere tale campo in tutte le richieste da loro inoltrate.
Il campo header può contenere uno o più elementi "product token" e commenti che identificano l'applicazione user agent, quali la sua versione, e qualsiasi suo sottoprodotto.
La sintassi dell'header di richiesta è la seguente:

User-Agent = "User-Agent" ":" 1*(product | comment)

dove "1*" significa "uno o più", mentre il simbolo "|" indica "e/o"

Un esempio può essere il seguente:

```
User-Agent: CERN-LineMode/2.15 libwww/2.17b3
```

7.4.2 Header HTTP From

Il campo header di richiesta *From* dovrebbe contenere un indirizzo e-mail per l'utente, in maniera tale da potergli consentire il controllo dell'applicazione user agent che fa la richiesta. Tale indirizzo deve essere in un formato cosiddetto "machine-usable", cioè utilizzabile dalla macchina (per maggiori informazioni vedere RFC 1123 all'indirizzo http://www.ietf.org/rfc/rfc1123.txt).

La sintassi per indicare l'header è la seguente:

From = "From" ":" mailbox

Un esempio:

```
From: webmaster@w3.org
```

Tale campo può essere usato per scopi di log e per identificare sorgenti di richieste non valide o non desiderate. From non viene utilizzato come mezzo di protezione degli accessi, nel senso che un determinato valore ad esso assegnato non preclude, nè dà particolari agevolazioni d'accesso, piuttosto viene adoperato per risalire alla persona che ha acceduto al proprio sistema in modo tale da poterla contattare in caso di eventuali problemi. Le applicazioni automatizzate di tipo *robot agent* (come user agent, ma automatizzate) possono essere un esempio di ciò, in quanto esse dovrebbero sempre includere questo header, cosicché la persona responsabile di tali applicazioni possa essere contattata per ogni possibile problema.
Il client non dovrebbe inviare l'header From senza l'approvazione dell'utente che ne fa uso, in quanto ciò può comportare una violazione del suo diritto alla

privacy.
Di conseguenza si raccomanda che venga fornita la possibilità all'utente di poter disabilitare, abilitare o modificare il valore di questo campo ogni volta che deve effettuare una richiesta.

7.5 DATESTAMP E GRANULARITY

I record ottenuti in risposta ai verbi *GetRecord*, *ListIdentifiers* e *ListRecords* [B1] includono nella sezione header un campo *datestamp*. Tale campo può essere rappresentato tramite le seguenti forme:

- YYYY-MM-DD detto formato "giorno"
- YYYY-MM-DDThh:mm:ssZ detto formato "secondo"

Dato un determinato item, i record in esso contenuti non devono necessariamente contenere tutti il campo datestamp nel medesimo formato.

I repository possono esprimere i vari datestamp nelle due forme precedentemente esposte, ma devono dichiarare la più fine granularità supportata nell'elemento <granularity> della risposta relativa al verbo *Identify*.

Essi devono anche supportare necessariamente i parametri *from* e *until* nelle seguenti forme:

- YYYY-MM-DD
- YYYY-MM
- YYYY

poiché gli harvester devono poter assegnare a tali parametri granularità meno precise di quelle supportate dal repository a cui la richiesta viene inoltrata. Indipendentemente dalla granularità adottata per gli argomenti from e until della richiesta, i valori di datestamp dei vari record dati in risposta saranno comunque espressi nella granularità più fine supportata dal repository.

Se l'harvester specifica dei valori assegnati a from e until con una granularità più fine rispetto a quella supportata dal repository, quest'ultimo ritornerà un messaggio di errore *badGranularity*.

Quando viene effettuata una raccolta di record o successivamente ad essa, uno o più item di un repository possono essere aggiunti o modificati. Dipendentemente dalla granularità supportata dal repository si possono avere le seguenti due situazioni:

- *Granularità YYYY-MM-DD*: in tal caso, se la modifica o l'aggiunta di item avviene durante la raccolta o successivamente ad essa, ma entro lo stesso giorno, è consigliabile che l'harvester effettui la prossima raccolta specificando un valore per il campo from a partire dal responseDate dell'ultima risposta ottenuta. Eventuali sovrapposizioni di record con lo stesso datestamp dovranno essere opportunamente gestite (i record già presenti e non modificati dovranno essere scartati).

- *Granularità YYYY-MM-DDThh:mm:ssZ*: in tal caso, se la modifica o l'aggiunta di un item avviene durante la raccolta, quindi tra il tempo di ricezione della richiesta e quello del responseDate restituito nella prima lista parziale di risposta di una sequenza, è consigliabile che l'harvester esegua la prossima richiesta di raccolta specificando un valore per il campo from che corrisponda al tempo di elaborazione della stessa, dato

approssimativamente dal valore del responseDate meno un certo quantitativo di tempo.

7.6 SET

Gli harvester possono scegliere di ignorare i *set* [B1] eventualmente implementati dai repository a cui si rivolgono, non specificando il parametro set nelle richieste e non prendendo in considerazione gli elementi <setSpec> nei vari record raccolti.
Un harvester può venire a conoscenza se un repository implementa i set ed eventualmente quali sono, attraverso l'invio di una richiesta ListSets. In tal caso si potrà avere in risposta la lista completa dei set oppure, se non supportati, un messaggio di errore *noSetHierarchy*.

L'elemento *setSpec* di un record può contenere un unico valore oppure una serie di valori separati dal simbolo due punti ":". Nel primo caso il valore rappresenta un set unico, nel secondo invece una gerarchia di set.
Effettuare una raccolta specificando un particolare set, determinerà la restituzione dei record appartenenti a tutti gli item ad esso associati e di quelli appartenenti agli item degli eventuali sottoset del set specificato.
Per esempio, considerando un set matematica contenente i sottoset: geometria, algebra, ecc., specificando nella richiesta il valore geometria, nel seguente modo: matematica:geometria, i record della risposta saranno relativi a tutti gli item contenuti in tale set, diversamente se si specifica solo il set matematica, i record restituiti saranno relativi al set matematica e a tutti i suoi sottoset.

7.7 CONTROLLO DI FLUSSO, LOAD BALANCING E

REINDIRIZZAMENTO

I software di raccolta dovrebbero rispettare il *controllo di flusso* delle risposte ritornate dai vari repository, dove per controllo di flusso si intende un eventuale partizionamento della risposta complessiva in più liste incomplete di risposta. Non rispettare ciò può comportare eventuali cali nelle performance del repository o in casi estremi dei danni.
Se un harvester riceve come risposta da parte del repository un codice HTTP 503 "Service Unavailable" (servizio non disponibile), come mezzo per effettuare il controllo di flusso, dovrebbe attendere, prima di inviare nuovamente la richiesta, un tempo almeno pari a quello specificato nell'header HTTP Retry-After, tale valore può essere o una data o un numero intero di secondi [S17]. Qualora l'HTTP Retry-After non sia presente, l'harvester non dovrebbe riprovare immediatamente, ma attendere per un tempo prefissato di alcuni minuti o preferibilmente sospendersi in attesa di un intervento manuale. In ogni caso, il software di raccolta dovrebbe essere scritto in maniera tale da non ritentare indefinitamente a seguito di eventuali problemi nella risposta.

E' possibile anche che un repository invii un codice di stato HTTP 302 "Found", per reindirizzare l'harvester ad un altro URL specificato nell'header HTTP Location. Ciò può essere fatto o come strategia di load balancing [B1] o per qualunque altra ragione.
Qualora l'header HTTP Location non sia presente in una risposta contenente un codice 302, l'harvester non dovrebbe riprovare automaticamente a inviare la richiesta.

7.8 LISTE INCOMPLETE DI RISPOSTA E RESUMPTION TOKEN

Il software di raccolta deve prevedere la possibilità di ricevere in risposta alle richieste ListIdentifiers, ListRecords e ListSets, liste incomplete.
Difatti, mentre i repository dei data provider possono o meno implementare tale meccanismo, gli harvester devono obbligatoriamente adottarlo per far fronte a tali eventualità.
Essi possono così capire quando si ha a che fare con una lista incompleta verificando la presenza in tale risposta dell'elemento *resumptionToken*.
Per passare alla lista incompleta di risposta successiva l'harvester dovrà assegnare il valore contenuto in tale elemento all'argomento resumptionToken della prossima richiesta.
L'ultima lista incompleta di risposta sarà riconoscibile dal fatto che il resumptionToken non contiene alcun valore.
Una volta ricevuta tale ultima lista è possibile concatenare tutte le risposte relative ai vari record per ottenere quella completa.

L'esempio seguente mostra una sequenza di richieste e risposte di liste incomplete:

Richiesta 1:

```
http://xXx.org/script?verb=ListIdentifiers
                      &from= 2001-01-01
                      &until=2001-01-03
```

Lista incompleta di risposta 1:

```
<ListIdentifiers>
  <header>...</header>
  <header>...</header>
  ...
  <resumptionToken completeListSize="20" cursor="0">
    2001-01-02:2001-01-03:0
  </resumptionToken>
```

```
</ListIdentifiers>
```

Richiesta 2:

```
http://xXx.org/script?verb=ListIdentifiers
                      &resumptionToken= 2001-01-02%3A2001-01-03%3A0
```

Lista incompleta di risposta 2:

```
<ListIdentfiers>
  <header>...</header>
  <header>...</header>
  ...
  <resumptionToken completeListSize="20" cursor="9">
    2001-01-03:2001-01-03:0
  </resumptionToken>
</ListIdentifiers>
```

Richiesta 3:

```
http://xXx.org/script?verb=ListIdentifiers
                      &resumptionToken=2001-01-03%3A2001-01-03%3A0
```

Lista incompleta di risposta 3 (il resumptionToken è vuoto poichè ultima lista):

```
<ListIdentfiers>
  <header>...</header>
  <header>...</header>
  ...
  <resumptionToken completeListSize="20" cursor="18">
  </resumptionToken>
</ListIdentifiers>
```

7.8.1 Argomento resumptionToken di codifica negli URL

Nei casi in cui, a seguito di una richiesta da parte dell'harvester, la risposta consista di una lista incompleta e un elemento resumptionToken, la successiva richiesta dovrà prevedere la corretta codifica del valore di quest'ultimo. Tale codifica dovrà essere resa disponibile per entrambi i metodi HTTP di richiesta GET e POST. La richiesta, nel caso siano presenti caratteri riservati, dovrà specificare al loro posto opportune sequenze di escape.
Esse sono le seguenti:

Carattere	Sequenze di Escape
/	%2F
?	%3F
#	%23
=	%3D
&	%26
:	%3A
;	%3B
(spazio)	%20
%	%25
+	%2

Per ulteriori approfondimenti sulla codifica, vedere l'elaborato di tesi correlata [B1].

7.8.2 Recupero di errori per le richieste di lista

In linea di principio, se durante una richiesta di lista avviene un errore di rete o una qualsiasi altra condizione che porti alla perdita di una lista incompleta di risposta, l'harvester può inviare nuovamente l'ultima richiesta effettuata contenente il più recente valore di resumptionToken per continuare la sequenza.

Un requisito che viene mantenuto durante le richieste di lista è quello dell'"idempotenza" del resumptionToken. Tale concetto significa che l'insieme delle liste incomplete di risposta corrisponderanno alla corretta lista di risposta completa (come se il partizionamento non fosse avvenuto).
Se durante una sequenza di richieste di lista incompleta, l'harvester riceve un errore di tipo badResumptionToken, esso deve assumere una delle due seguenti possibilità:

- Il resumptionToken è scaduto
- Il resumptionToken non è valido per un qualunque altro motivo

In questi casi particolari, l'harvester non ha alcun modo per recuperare la sequenza di richieste di lista, esso dovrà far ripartire la richiesta di lista originaria daccapo.
Se un harvester riceve invece un qualunque altro errore non conosciuto, la sequenza di richieste di lista, anche in questo caso, non sarà recuperabile ed esso dovrà nuovamente far ripartire la richiesta di lista.

7.9 COMPRESSIONE DELLA RISPOSTA

I vari repository possono supportare diversi tipi di codifica della risposta per quanto riguarda la compressione. Per rendere noti i vari tipi di compressione

supportati essi devono indicarli negli elementi *compression* della risposta al verbo Identify. In tal modo, gli harvester, inviando la richiesta Identify, possono scegliere, tra quelle supportate, le codifiche di preferenza.
Un esempio che mostra una parte della risposta al verbo Identify, che specifica che il repository supporta le codifiche *gzip* e *compress* in aggiunta all'obbligatoria *identity* è la seguente:

```
<Identify ...>
  ...
  <compression>gzip</compression>
  <compression>compress</compression>
  ...
</Identify>
```

Ricevuta una risposta simile a quella sopra indicata, un harvester che supporti ad esempio la compressione gzip potrà inviare le sue richieste con uno dei seguenti header HTTP:

- Accept-Encoding: gzip, identity
- Accept-Encoding: gzip;q=1.0, identity;q=0.5

La prima forma dice semplicemente che entrambi i tipi di risposta (gzip e identity) sono accettabili.
La seconda forma, che è quella raccomandata, dice che la codifica gzip è da preferire ad identity poiché è specificato un valore per "q" più elevato.
Per maggiori approfondimenti vedere [S17].
In tutti i casi il tipo di codifica obbligatoria identity deve essere inclusa nell'elenco.

7.10 COME RACCOGLIERE TUTTI I METADATI DA UN

REPOSITORY

Un harvester può voler ottenere una copia completa di un repository comprendendo la struttura dei set e tutti i formati di metadati. Per fare questo bisogna seguire una strategia come la seguente:

- Inviare una richiesta Identify per ricavare la granularità di datestamp più fine supportata dal repository.
- Inviare una richiesta ListMetadataFormats per ottenere una lista di tutti i formati (da assegnare a metadataPrefix) supportati.
- Raccogliere usando le richieste ListRecords per ogni formato (metadataPrefix) supportato. Essere a conoscenza del fatto che il repository supporti la granularità di tipo YYYY-MM-DDThh:mm:ssZ assicura minor sovrapposizione di record dovuta a cambiamenti durante le raccolte.
- Utilizzando gli elementi setSpec contenuti negli header dei vari record raccolti, è possibile dedurre la struttura dei set implementati dal repository.
- Si possono ricostruire i vari item del repository verificando tutti i record aventi lo stesso valore per il campo unique identifier e considerando tale valore come identificatore dell'item stesso. Ai vari item raccolti bisogna assegnare come valori di datestamp, data e ora locali al sistema di raccolta.
- Le informazioni di provenienza, i diritti di utilizzo, ecc., contenute nel campo <about> possono essere ricostruite al livello di item se tutti i record appartenenti ad un item specifico hanno il medesimo contenuto per tale campo. Nel caso in cui tale informazione è differente per ogni record appartenente allo stesso item, può essere necessario immagazzinare tale informazione separatamente per ogni formato di metadato.

La figura 2 seguente mostra in linea generale la comunicazione tra Data e Service provider.

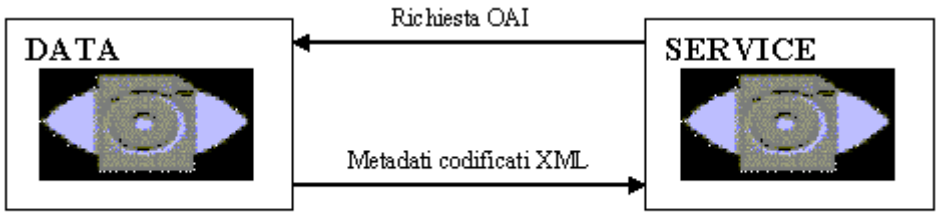

Figura 2 – Comunicazione tra Data e Service Provider

7.11 TESTING

Una volta realizzato il proprio service provider, conforme al protocollo OAI-PMH, bisognerà testare l'applicazione effettuando delle raccolte di metadati dai data provider registrati, conformi all'OAI.
Un elenco di tali data provider è presente sul sito web ufficiale dell'OAI, all'indirizzo:

http://www.openarchives.org/community/index.html

Ovviamente, in funzione dei servizi a valore aggiunto che si desidera offrire, è possibile raccogliere i metadati da data provider non registrati in tale elenco.
Comunque, effettuare il testing qui menzionato, con i data provider registrati, rappresenta una verifica ottimale per assicurarsi di avere un'implementazione funzionante e conforme al protocollo OAI-PMH.

7.12 REGISTRAZIONE

Una volta accertato il corretto funzionamento del service provider realizzato e testato, è possibile registrarlo al sito ufficiale per i service provider:

http://www.openarchives.org/service/registerasprovider.html

In tal caso verranno richieste le seguenti informazioni:

- Il nome del service provider (es. CDSware)
- Una descrizione dei tipi di servizi offerti
- L'ambito di interesse (es. matematica, medicina, …)
- L'URL della pagina web da associare all'applicazione
- L'indirizzo e-mail del responsabile del servizio da contattare
- Una lista dei data provider da cui si raccoglie

L'OAI-PMH in ogni caso non dice nulla riguardo ai servizi che un service provider deve fornire all'utente finale, da ciò si evince come la natura dei servizi dipenderà dall'interesse degli implementatori e dai tipi di metadati raccolti.

BIBLIOGRAFIA

[B1] Amelotti Ercole, *Protocollo OAI-PMH negli Open Archive e applicazione CDSware per la rappresentazione dei relativi dati bibliografici*, tesi di laurea in informatica, Università degli Studi di Messina, A.A. 2003-2004 (relatore Puccio L., correlatore De Robbio A.).

SITOGRAFIA

[S16] http://www.oaforum.org/tutorial/

[S17] http://www.ietf.org/rfc/rfc2616.txt