3. LINEE GUIDA PER IL DATA PROVIDER

3.1 INTRODUZIONE

Benché il protocollo OAI-PMH introduca una soglia minima d'implementazione per rendere possibile l'interoperabilità tra repository, nella sua definizione esistono diversi costrutti, tra i quali i set e il resumption token, che sono opzionali e per i quali il protocollo stesso non dà alcuna limitazione o raccomandazione da seguire.

In questa parte dell'elaborato, basata sulle "Linee Guida del Data Provider" presenti al sito dell'OAI [S4] e sulle considerazioni presenti al sito dell'OAForum [S3], vengono esposte le *linee guida* per l'implementazione del repository da parte del data provider che consistono nella specificazione di:

- Requisiti del data provider
- Componenti ed architettura del data provider
- Rappresentazione dei dati
- Funzionalità opzionali
- Resumption token
- Controllo di flusso e load-balancing
- Gestione degli errori
- Testing

- Registrazione

E' bene considerare come il protocollo presenti molte scelte opzionali e lo scopo principale della sua implementazione sia realizzare una comunicazione solida tra repository e harvester; ciò non ostante tali opzioni possono essere tralasciate dai data provider che non le ritengano necessarie in un dato momento o le ritengano di difficile implementazione.

In conclusione, questa parte dell'elaborato vuole essere una sezione di utilità per chiunque voglia implementare il proprio data provider e allo stesso tempo un compendio per le parti del protocollo precedentemente trattate (Cap 2).

3.2 REQUISITI

Allo stato attuale del protocollo OAI-PMH i repository, che intendono aderire all'iniziativa, assumendo la funzionalità di data provider, debbono possedere i seguenti requisiti:

- *metadati* per la descrizione delle risorse;
- *datestamp* per i record di metadati;
- *base URL* come identificatore unico d'archivio;
- *identificatore unico* per ogni item del repository;
- *Dubline Core* come formato minimo di metadati.

I concetti di metadati, base URL e identificatore unico sono già stati trattati nella parte relativa al protocollo OAI-PMH dell'elaborato al Cap. 2.

In questo paragrafo si focalizzerà brevemente l'attenzione sui datestamp e sul Dubline Core.

3.2.1 Datestamp

L'OAI-PMH indica due tipi di granularità per i datestamp:

- Giorni: YYYY-MM-DD
- Secondi: YYYY-MM-DDThh:mm:ssZ

La seconda granularità, a meno di particolari questioni implementative, è da preferirsi nei vari repository.
Indipendentemente dalla granularità usata, ogni qual volta un item viene aggiunto o modificato, i metadati devono essere resi disponibili effettuando un aggiornamento del datestamp. Se ciò non avviene, l'aggiornamento potrà essere perduto a seguito di una raccolta incrementale. Ad esempio, se un item con datestamp "2002-01-03" viene aggiunto ad un repository e il suo datestamp non viene aggiornato con la data di raccolta, ad esempio "2002-01-04", una raccolta incrementale a partire dal "2002-01-04" non considererà l'aggiornamento dal giorno precedente a tale data.

Per assicurare che un item non venga perso nelle varie fasi di raccolta, è necessario aggiornare il suo datestamp interno ogni qual volta esso venga modificato. Per far ciò si potrebbe seguire un algoritmo come il seguente:

```
if (DatestampInternoItem > DatestampDiDisseminazione)
{
   datestamp = DatestampInternoItem
}
else
{
   datestamp = DatestampDiDisseminazione
}
```

Come data di disseminazione, per l'esempio precedente, gli harvester possono utilizzare il *responseDate* della risposta XML come mezzo per evitare disallineamento col clock del repository.

3.2.2 Dubline Core e altri formati

Come già accennato nel paragrafo 2.2 del Protocollo OAI-PMH, il formato DC (Dubline Core) per i metadati rappresenta una sorta di "linguaggio" che tutti i data provider debbono conoscere per esporre i propri metadati all'esterno e garantire l'interoperabilità tra archivi.

È bene considerare come il protocollo si riferisca al DC quale formato minimo richiesto per la "disseminazione" dei metadati; da ciò si evince come i repository non siano tenuti ad immagazzinare i propri metadati in DC, ma devono piuttosto creare una mappatura tra il formato utilizzato per la loro conservazione e quello richiesto. Molti repository infatti immagazzinano i loro metadati in qualche altro formato e quando l'harvester invia una richiesta, specificando il DC come formato desiderato per la risposta, effettuano la conversione dinamicamente.
Ovviamente tutti i repository che adottano già il DC come formato per la conservazione, non dovranno effettuare alcuna conversione e potranno esporre direttamente i metadati.

Unitamente al DC, che deve essere sempre disponibile, i repository possono adottare altri formati per la disseminazione, ad esempio il MARC21 che è più ricco e descrittivo del precedente.
L'OAI non dà alcuna limitazione in tal senso ed incoraggia le comunità di data e service provider ad esporre i formati da loro scelti.
Per maggiori approfondimenti sull'argomento Dublin Core si rimanda all'elaborato di Buttà Basilio [B3].

3.3 COMPONENTI ED ARCHITETTURA DEL DATA PROVIDER

I componenti che rappresentano "la dotazione software" che un data provider dovrebbe possedere sono i seguenti:

- *Parser degli argomenti* – software che serve a validare le richieste OAI, cioè verificare la loro correttezza sintattica.
- *Generatore di errori* – software che ha il compito di creare delle risposte XML contenenti al loro interno messaggi di errore codificati.
- *Interrogatore ed estrattore di metadati locali* – software che ha il compito di interrogare e recuperare i metadati nel formato richiesto dal database locale.
- *Generatore della risposta XML* – software che si occupa di codificare nel formato XML le risposte da fornire all'harvester.
- *Controllore di flusso* (opzionale) – software che ha il compito di realizzare sequenze di liste incomplete per richieste a cui corrispondono delle risposte contenenti un numero "troppo" elevato di record (vedi par 2.10.10). Il meccanismo di controllo più comunemente usato per la sua implementazione è il resumption token.

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

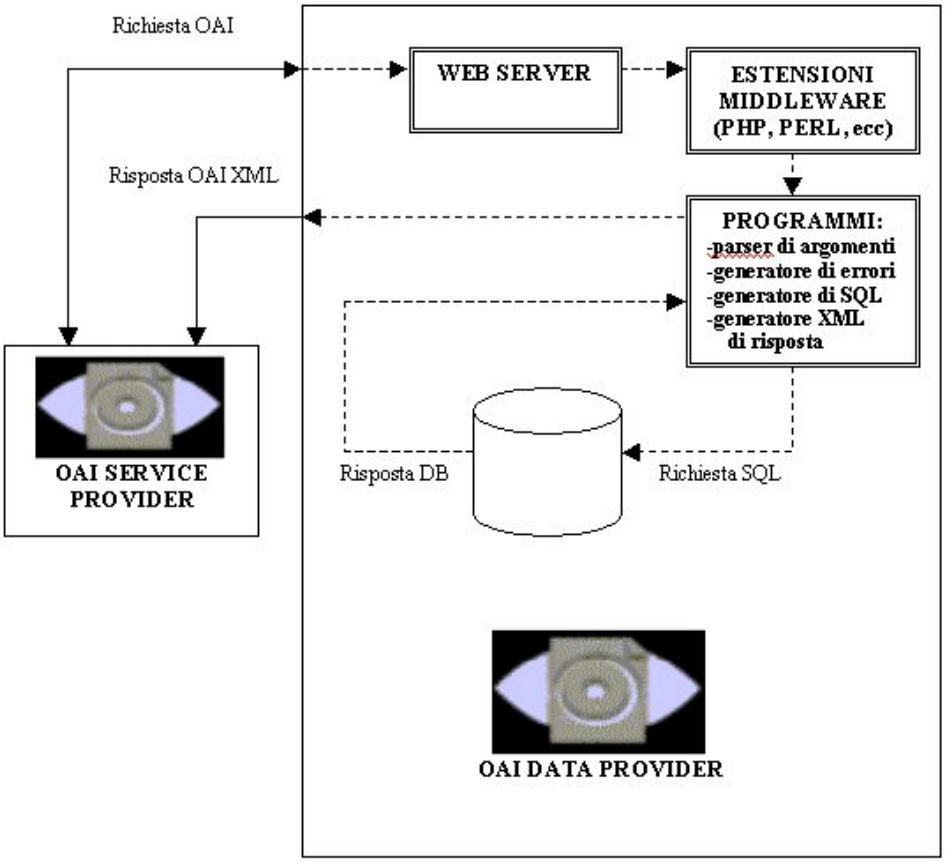

Figura 1 – Architettura di un Data provider

3.4 RAPPRESENTAZIONE DEI DATI

E' buona norma utilizzare le seguenti raccomandazioni per la rappresentazione dei dati da parte di un data provider:

- *Date* – il formato per le date da adottare è il seguente: 2004-05-12, piuttosto che altre forme, quali per esempio: 2004-xx-xx, 2004, 12.05.2004.
- *Codici di lingua* – usare le forme contratte a tre caratteri quali: ita, eng, ger, …, piuttosto che le altre forme, quali: it, en, de, italian, english, german, ….

Si raccomanda inoltre di utilizzare un elemento XML separato per ogni entità costituita da valori multipli come nell'esempio seguente:

```
Autore: <dc:creator>Amelotti, Ercole</dc:creator>
        <dc:creator>Buttà, Basilio</dc:creator>
```

e non utilizzare la forma:

```
Autore: <dc:creator>Amelotti, Ercole; Buttà, Basilio</dc:creator>
```

3.5 FUNZIONALITA' OPZIONALI

Di seguito vengono trattate le funzionalità opzionali del protocollo. Gli amministratori del repository del data provider sono liberi di poterle implementare tutte, alcune o nessuna.

3.5.1 Campo <about>

Come già detto nel par. 2.5 del Protocollo OAI-PMH, un record all'interno del repository del data provider può contenere un campo opzionale *<about>* autoreferenziante, contenente cioè informazioni sul record stesso. Benché esso sia da considerare un utile strumento per indicare ad esempio la provenienza o i diritti di utilizzo dei metadati, il suo uso rimane opzionale.

3.5.2 Set

Come già ampiamente discusso nel paragrafo 2.7 del Protocollo OAI-PMH, i set permettono di raggruppare gli item di un repository consentendo agli harvester dei service provider di poter effettuare raccolte selettive basate sui set. Per tale costrutto non è obbligatoria l'implementazione da parte del data provider, benché esso permetta di accrescere le potenzialità di raccolta da parte degli harvester. L'utilizzo dei set viene generalmente adottato da comunità specifiche di service e data provider che per comodità raggruppano gli item in collezioni da loro predefinite.
Esempi di set, che sono stati definiti all'interno di organizzazioni e comunità, includono quelli basati su journal o titoli di serie, su classificazioni di soggetto o categorie, su collezioni in base alle discipline, sui tipi di risorsa e sugli autori dei lavori.

3.5.3 Compressione della risposta

Il repository del data provider può utilizzare un meccanismo di compressione per

la risposta da inviare all'harvester, a seguito di una precedente richiesta OAI-PMH, per migliorare le prestazioni del sistema di comunicazione.
Tale tipo di comunicazione quindi si rende possibile sotto le seguenti condizioni:

- l'harvester deve essere in grado di accettare risposte compresse
- le risposte inviate devono essere abbastanza lunghe da garantire un certo margine di tempo per permettere di effettuare rispettivamente la compressione da parte del repository e la decompressione da parte dell'harvester

Dalle due precedenti condizioni si capisce che i repository dovrebbero porsi il problema di implementare un meccanismo di compressione della risposta, qualora le loro collezioni di record abbiano un numero superiore alle migliaia e, soprattutto, qualora gli harvester, che più frequentemente si rivolgono ad essi per la raccolta, abbiano adottato tale meccanismo.

Un repository può adottare nessuno, uno o più tipi di codifica per la compressione della risposta e può indicare ciò utilizzando uno o più elementi *compression* nella risposta al verbo *Identify* di cui si è parlato nel par. 2.9.4.2 del protocollo OAI_PMH. I valori raccomandati che è possibile assegnare a compression sono quelli definiti per l'header Content-Encoding indicati nell'RFC 2616 [S5]che descrive l'HTTP 1.1. Essi sono:

- *gzip* – un formato di codifica prodotto dal software di compressione dei file "gzip". Tale formato impiega la codifica Lempel-Ziv con un CRC a 32 bit.
- *compress* – un formato di codifica prodotto dal programma di compressione dei file Unix "compress". Tale formato è adattabile alla codifica LZW (Lempel-Ziv-Welch).
- *deflate* – fomato corrispondente allo "zlib" combinato col meccanismo di

compressione “deflate”.

- *identity* – è la codifica di default che non adotta alcuna compressione.

Tutti i repository devono supportare la codifica *identity*, cioè senza compressione, e devono rispondere alle richieste con tale codifica a meno che l’harvester non utilizzi l’Accept-Encoding dell’header della richiesta specificandone un altro tipo.

Di seguito i passi del flusso tra data e service provider:

1. il repository renderà noto all’harvester, che gli ha inoltrato una richiesta di tipo Identify, le codifiche di compressione adottate presenti negli elementi compression:

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

2. successivamente l’harvester invierà la richiesta di raccolta vera e propria facendola seguire dal campo header aggiuntivo Accept-encoding:

```
GET http://xXx.org/oai-script?verb=ListRecords&
                              set=libri&
                              from=2004-04-21 HTTP/1.1

Accept-encoding: gzip;q=1.0, identity;q=0.5
```

3. la risposta, opportunamente codificata, specificherà la codifica scelta dal repository attraverso il Content-Encoding header:

```
Content-Encoding: gzip
```

3.5.4 Liste incomplete e resumption token

Come già discusso nel paragrafo 2.10.10 del Protocollo OAI_PMH, il repository del data provider può decidere di implementare il partizionamento della risposta per migliorare le proprie prestazioni. Tale meccanismo fa uso delle *liste incomplete di record* e del parametro aggiuntivo *resumptionToken*.

Dal momento che il protocollo OAI-PMH non dice nulla circa la dimensione raccomandata per una lista incompleta, potrebbe essere ragionevole che i repository con meno di un migliaio di item rispondano sempre con un'unica lista di record di risposta e quindi non implementino il resumption token.

L'utilizzo secondo l'OAI-PMH del resumption token è definito attraverso le seguenti indicazioni:

- un repository deve includere un elemento resumptionToken come parte di ogni risposta che include una lista incompleta
- per ricevere la prossima lista incompleta, la richiesta successiva deve usare il valore dell'elemento resumption token ricevuto nella risposta per assegnarlo all'argomento resumptionToken
- l'ultima lista incompleta di risposta deve includere un elemento resumption token vuoto

Se tali indicazioni non vengono rispettate o altri utilizzi vengono fatti, dovrebbe essere restituito un errore all'harvester.
Anche se non specificato dall'OAI-PMH, è bene seguire le seguenti regole:

1. ogni lista incompleta deve essere composta da un numero intero di record, cioè un record non deve essere presente in parte in una lista e nella restante in un'altra.
2. proprio perché il formato del resumption token non è specificato dal protocollo, esso dovrebbe essere sconosciuto all'harvester del service provider, in quanto quest'ultimo dovrebbe limitarsi al solo utilizzo specificato dall'OAI_PMH.
3. prima di includere il resumption token come argomento della successiva richiesta, l'harvester deve codificare eventuali caratteri speciali di cui esso si compone (vale quanto detto nel paragrafo 2.6.1.3.1 del protocollo OAI).

Questa parte del protocollo può essere la più complessa da implementare poiché deve tener conto dei carichi di lavoro del repository del data provider.

3.6 RESUMPTION TOKEN

Il protocollo OAI-PMH non specifica un modo ben preciso per implementare il resumption token e quindi il meccanismo di partizionamento della risposta. Questo però lascia ai gestori dei vari repository la massima libertà e flessibilità di adattamento del meccanismo alle proprie esigenze.

Gli harvester devono trattare i valori dei vari elementi resumption token restituitigli, come oggetti trasparenti; cioè oggetti di cui non conoscono l'implementazione.
Un repository può scegliere se rendere la sintassi del resumption token ovvia, come ad esempio:

```
<resumptionToken>
   resultSet=157&nextRange=1001-2000
</resumptionToken>
```

o può scegliere di codificarlo opportunamente in modo che la sintassi non sia visibile, come:

```
<resumptionToken>9023A210CD007</resumptionToken>
```

Il protocollo è stato progettato per fornire un meccanismo tramite il quale qualsiasi harvester possa effettuare raccolte incrementali da un repository e non perdere i cambiamenti che possono avvenire in esso: aggiunta, modifica e cancellazione di record. Anche l'implementazione di liste incomplete e del resumption token per il partizionamento dovrebbe permettere di non perdere i cambiamenti nel repository.

Ad esempio può accadere che una variazione avvenuta durante una sequenza di raccolta (harvest sequence), non sia particolarmente critica da inficiare il risultato finale; è il caso, ad esempio, di un item modificato durante la sequenza di richieste necessarie per completare una ListRecords.

Il nuovo valore del datestamp del record sarà sicuramente un orario successivo a quello di inoltro della prima richiesta da parte dell'harvester, ma con molta probabilità, precedente all'ultima inoltrata.

In questo caso specifico è necessario fare una scelta tra prendere il record nella sequenza corrente o scartarlo aspettando la prossima occasione di raccolta.

Quando il repository del data provider effettua il partizionamento di una risposta, esso deve in qualche modo tener traccia dello stato di avanzamento della stessa, consentendo così all'harvester di effettuare le richieste successive per i prossimi record in modo iterativo.

Le due principali strategie tra cui scegliere per l'implementazione del resumption token e delle risposte di liste incomplete sono:

- codifica del resumption token
- caching del result set.

Nei prossimi due paragrafi se ne da una descrizione.

3.6.1 Codifica di stato del resumption token

La soluzione qui presentata, consiste nel codificare lo stato di una lista incompleta di risposta nell'elemento resumptionToken permettendo così al repository di non doverlo conservare (come avviene invece per il Caching del result set par.3.6.2), ma di calcolare lo stato stesso ad ogni successiva richiesta dell'harvester, tramite l'argomento resumptionToken contenente il valore consegnato dal repository nella precedente lista incompleta.
Ovviamente la possibilità di non dover salvare informazioni per una risposta incompleta, viene pagata da un sovraccarico computazionale per la codifica e la decodifica del resumption token stesso.

L'esempio seguente mostra la codifica di stato nel resumption token:

data la richiesta OAI-PMH

```
http://xXx.org/oai-script?verb=ListRecords
                          &set=libri
                          &from=2004-04-21
```

l'elemento resumptionToken, consegnato all'harvester unitamente alla prima lista incompleta di risposta, potrà essere il seguente:

```
<resumptionToken>
                set=libri
                &from=2004-04-21
                &until=2004-06-13
                &range=751-1500
                &metadataPrefix=oai_dc
</resumptionToken>
```

Come si può notare l'elemento racchiude gli argomenti **set**, **from**, **until**, **range**, **metadataPrefix**, ognuno dei quali è una singola parte dello stato della risposta.
Gli argomenti set e from indicano rispettivamente che l'harvester ha richiesto i record di metadati appartenenti al set *libri* e a partire dal 21-04-2004, e dunque sono quelli specificati nella richiesta di partenza.
I restanti argomenti non sono specificati nella richiesta originaria.

L'argomento until è presente come parte del resumptionToken in quanto la richiesta specifica che i record partano dal 21-04-2004 (argomento from) fino al più recente datestamp disponibile; tale ultimo valore viene fatto corrispondere in questo caso al 13-06-2004, ossia il datestamp della risposta inviata all'harvester o più precisamente il *responseDate*. Questo tipo di approccio permette di evitare che item nuovi o modificati, con datestamp successivi al valore di until, vengano inclusi nelle successive liste incomplete di risposta, escludendo così i cambiamenti avvenuti in seguito alla prima lista.
L'argomento range viene utilizzato per tener traccia di quanti record sono stati inviati, mentre metadataPrefix ha valore *oai_dc* poiché nessun formato è stato specificato nella richiesta e di conseguenza i record vengono disseminati con il formato minimo di interoperabilità stabilito dal protocollo.

Questo meccanismo di implementazione consiste nel codificare gli argomenti della richiesta originaria e qualche altra informazione di completamento, come

until, range e metadataPrefix visti nell'esempio precedente, nell'elemento resumptionToken di risposta.

Successivamente, quando il repository riceverà il resumptionToken a seguito di una nuova richiesta di lista, potrà riportare lo stato del sistema al punto in cui si era interrotto elaborando tutte le informazioni neccessarie per il ripristino, codificate nel resumptionToken stesso.

Benché, come già detto, questa soluzione sia computazionalmente dispendiosa, il vantaggio sta nel fatto che non è necessario trattenere nel repository informazioni per il ripristino, tale compito viene lasciato all'harvester che dovrà farsi carico di memorizzare il resumptionToken nel caso voglia proseguire il completamento della risposta.

La figura 2 seguente mostra a grandi linee come viene risolto il caso tipico di un'anomalia di aggiornamento:

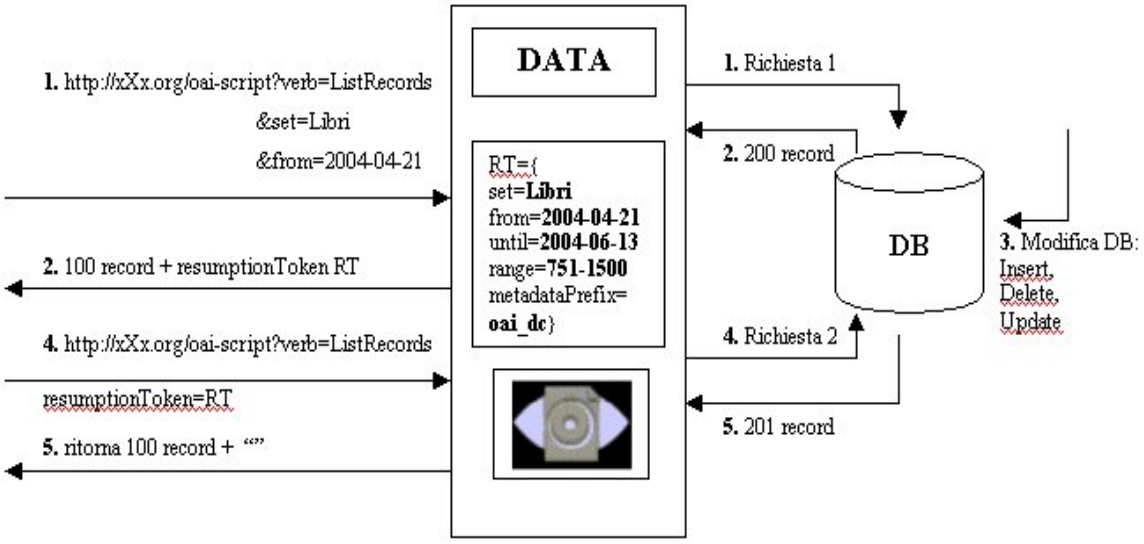

Figura 2 – Anomalia di aggiornamento nel partizionamento della risposta tramite resumption token

Come si può vedere dai seguenti passi:

1. il data provider riceve la prima richiesta ListRecods
2. il provider ha 200 record corrispondenti alla richiesta, ma ne invia solo 100 ed invia anche un resumption token RT nella risposta
3. avviene una modifica all'interno del repository del data provider, ad

esempio viene aggiunto un nuovo record che corrisponde ai requisiti della prima richiesta

4. il data provider riceve la seconda richiesta corredata del resumption token RT
5. il provider adesso ha ancora 100 record corrispondenti alla richiesta più un record appena inserito, ma, dato che il resumption token ha come valore del parametro from la data di invio della prima richiesta, esso ritornerà quei 100 più un resumption token vuoto, ad indicare che i record da raccogliere sono finiti.

In tal modo il record appena inserito potrà essere acquisito attraverso raccolte successive da parte del service provider.

A meno che i repository non abbiano una buona ragione per gestire una "cache" delle informazioni di ripristino, essi dovrebbero utilizzare il meccanismo di codifica che è stato esposto in questo paragrafo.

3.6.2 Caching del result set

Questa soluzione prevede che il repository trattenga i risultati di una richiesta di lista (list request) inoltrata dall'harvester e li ritorni in un secondo momento come un certo numero di risposte di lista incompleta. Tali risposte saranno rese accessibili all'harvester dopo l'invio di ulteriori richieste contenenti gli appropriati valori nel resumptionToken.

Una maniera per implementare questo meccanismo potrebbe essere quella di codificare un *identificatore di sessione* ed un *cursore* nel resumptionToken per tener traccia rispettivamente, della richiesta che deve essere ancora completata e

dei record che devono essere ancora inviati.
Un identificatore di sessione non può essere usato come resumptionToken senza informazioni aggiuntive sul proprio stato perché deve essere possibile riutilizzarlo per inviare una nuova richiesta di lista incompleta. È permesso ai repository di far scadere i valori di resumptionToken trasmessi agli harvester: il momento in cui avviene la scadenza può essere specificato nell'attributo *expirationDate* dell'elemento.

L'esempio seguente mostra la codifica nel resumption token:

data la richiesta OAI-PMH

```
http://xXx.org/oai-script?verb=ListRecords
                          &set=227
                          &from=1999-02-03
```

l'elemento resumptionToken, consegnato all'harvester unitamente alla prima lista incompleta di risposta, potrà essere il seguente:

```
<resumptionToken>searchId=4234&range=750-1500</resumptionToken>
```

le successive richieste provenienti dall'harvester, comporteranno l'invio da parte del repository dei prossimi 750 record appartenenti al set di risultati (result set) memorizzato.

Un repository potrebbe anche controllare che in un determinato momento due harvester differenti non inviino la stessa richiesta; in caso contrario sarebbe possibile servirli entrambi tramite lo stesso set di risultati. Benché possa sembrare poco probabile, una situazione del genere è molto comune per comunità ben definite di data e service provider, nelle quali ad esempio i tipi di richiesta

inoltrati si basano frequentemente sui "set di item" messi a disposizione. E' evidente come in casi simili a questo, il caching del set dei risultati porti a prestazioni migliori per il repository.
Lo svantaggio introdotto da questo tipo di approccio è dovuto alla responsabilità aggiuntiva da parte del repository di dover gestire il riempimento e lo svuotamento della cache dei risultati.
Dato che la conoscenza da parte dell'OAI riguardo ai profili di raccolta da parte degli harvester di diverse comunità è alquanto limitata, non è ancora chiaro quale possa essere l'algoritmo di gestione del meccanismo di caching migliore per un dato repository. Con ciò è impossibile al momento dare una raccomandazione generale sulla gestione di tale problema.

Si può comunque dire come i repository che vogliano implementare questo tipo di soluzione, dovrebbero utilizzare l'attributo *expirationDate* (par. 3.6.5) dell'elemento resumptionToken per indicare agli harvester per quanto tempo il valore in esso contenuto sarà valido.

3.6.3 Idempotenza delle richieste di lista

I repository devono implementare il meccanismo che coinvolge il resumptionToken in modo da pemettere agli harvester di recuperare una sequenza di richieste per le liste incomplete, inviando nuovamente una richiesta di lista con il resumptionToken più recente. Lo scopo di questo è permettere agli harvester di accorgersi degli errori di rete o di aggiornamento in modo da ripetere la richiesta da capo.
Il protocollo OAI-PMH richiede per le richieste di lista una "forma debole" di idempotenza usando l'argomento resumptionToken:

- se non ci sono cambiamenti nel repository, due richieste di lista identiche devono ritornare il medesimo risultato.
- se non ci sono cambiamenti nel repository, inviando di nuovo la richiesta di lista incompleta più recente, usando lo stesso resumption token non scaduto, esso deve ritornare il medesimo risultato. Questa caratteristica permette agli harvester di ripetere l'ultima richiesta (ad esempio perché errori di rete sono avvenuti) evitando di ripartire da quella iniziale.
- se ci sono cambiamenti nel repository tra due richieste di liste complete o incomplete, entrambe "devono" includere tutti gli elementi che non hanno avuto cambiamento nel datestamp e "possono", relativamente alla politica adottata, includere record con datestamp fuori dal range della richiesta iniziale. La conseguenza di ciò è che le singole richieste di lista incomplete devono avere lo stesso grado di idempotenza della richiesta di lista come se fosse completa.

  Nei casi in cui ci siano cambiamenti sostanziali nel repository, "potrebbe" essere appropriato ritornare un errore *badResumptionToken*, segnalando all'harvester che dovrebbe iniziare da capo la sequenza di richieste.
- solo il resumptionToken più recentemente ricevuto da un harvester, può essere inviato di nuovo. La conseguenza di ciò è che i repository debbono supportare il requisito di idempotenza per la più recente richiesta di lista incompleta che è stata risposta. Essi devono accettare sia il più recente resumptionToken inviato che il suo predecessore.

Considerando le due strategie di implementazione viste ai due precedenti paragrafi, se in un repository avvengono dei cambiamenti rapidi in confronto alla velocità per la quale un harvester invia susseguenti richieste con argomenti resumptionToken, esse possono produrre differenti risultati.

Possiamo considerare un repository che aggiunga un certo numero N di nuovi record nel mezzo della sessione di un harvester, e per la quale questi record corrispondano al criterio di selezione specificato nella richiesta iniziale (set, datestamp, ecc.):

- le implementazioni che fanno affidamento sull'immagazzinamento dello stato della risposta nel repository (Es: Il caching del result set) sono con molta probabilità propense nel dare risultati relativi al momento iniziale della raccolta; dunque preferiscono non tener conto dei cambiamenti avvenuti.
- le implementazioni che trasferiscono lo stato della risposta nel valore attribuito al parametro resumptionToken (Es: Codifica di stato del resumptionToken) sono con molta probabilità propensi nel dare i risultati relativi al momento dell'ultima richiesta; dunque preferiscono tener conto dei cambiamenti avvenuti nel repository.

Benché entrambe le soluzioni possano essere implementate con schemi più sofisticati per limitare il differente comportamento di risposta, sarebbe meglio che gli implementatori considerino il tempo medio di accesso previsto al repository e impostino l'attributo opzionale expirationDate dell'elemento resumptionToken opportunamente. Giocando infatti su queste due grandezze sarebbe possibile determinare se il comportamento deve essere prossimo ai sistemi che fanno uso del caching della risposta o a quelli che codificano lo stato della stessa nel resumption token.

Le regole precedentemente esposte relative all'idempotenza del resumption token, rappresentano dunque un sistema ottimale per aiutare gli harvester nelle raccolte da grandi repository dati gli errori che possono avvenire (es: risposte perdute,

errori di aggiornamento, ecc.).

3.6.4 Attributi del resumption token

L'elemento resumptionToken nella rappresentazione XML della risposta può includere degli attributi opzionali al suo interno. Essi sono:

- *expirationDate* – è una data nel formato UTC che indica la data di scadenza della validità del resumption token inviato dal repository del data provider. È raccomandato che i repository usino per esso dei valori che rimangano validi per almeno alcune decine di minuti.

  Se, per qualsiasi motivo, un repository non può continuare il completamento di una risposta, esso può inoltrare un errore *badResumptionToken*, eventualmente corredato di una stringa esplicativa, per fermare la sequenza di richieste dell'harvester. L'harvester dovrà poi ripartire dalla richiesta iniziale.

  Qualora l'attributo non venga adottato, i valori assegnati al resumption token devono essere validi e riconoscibili nel tempo.

- *completeListSize* – indica il numero totale degli elementi di cui si compone la risposta completa (composta da più liste incomplete) ad una richiesta di lista. Il valore dell'attributo può essere utilizzato per risolvere problemi relativi ai cambiamenti nel repository, visto che durante più richieste di liste incomplete uno o più record possono essere inseriti o cancellati.

  Il valore dell'attributo in questione può essere accurato solo nel caso di sistemi in cui il result set della risposta venga memorizzato (caching del result set). In altri casi bisognerà consentire ai repository di rivedere e indicare all'harvester eventuali variazioni nel valore del completeListSize.

- *cursor* – indica il numero di elementi di cui si compone l'ultima lista

incompleta ricevuta; il suo valore è sempre “0” nella prima lista incompleta.

3.7 CONTROLLO DI FLUSSO E LOAD-BALANCING

Due concetti importanti, che interessano i repository nel trattamento delle richieste provenienti da vari harvester, sono quelli di *controllo di flusso* e di *load-balancing*.

Il controllo di flusso può essere visto come una tecnica di trattamento *intra-repository* resa disponibile attraverso le liste incomplete e gli elementi di resumption token visti in precedenza; la sua semantica è quindi specificata dal protocollo OAI-PMH e vale tutto quello che è stato detto nel paragrafo 3.6.

Il load-balancing (bilanciamento del carico di lavoro) è, al contrario del controllo di flusso, una tecnica *inter-repository* per reindirizzare la raccolta da repository, che in un determinato momento gestiscono grossi carichi di richieste, a repository con gli stessi contenuti, che in quello stesso momento non sono pesantemente caricati. La semantica di tale tecnica non è esplicitata dal protocollo e dovrebbe essere gestita a livello transport.

Entrambe le tecniche sono opzionali e possono essere usate indipendentemente l’una dall’altra o in combinazione. Un esempio tipico, infatti, è quello in cui una richiesta viene reindirizzata ad un repository *mirror* che abbia un carico più leggero di richieste da gestire e, successivamente, il repository, attraverso l’utilizzo di elementi resumptionToken, passa ad un flusso di richieste e risposte scambiate con l’harvester fino al completamento.

3.7.1 Controllo di flusso

Il significato e le possibilità di implementazione dell'elemento resumptionToken sono state precedentemente descritte al par. 3.6. Quello che viene discusso in questo paragrafo sono i metodi del controllo di flusso a livello transport.

Il protocollo HTTP definisce un codice di stato 503 che sta per "Service Unavailable" (servizio non disponibile). Quando un repository, a cui viene inviata una richiesta, supporta già un carico di lavoro abbastanza elevato, invece di processare la richiesta e ritornare il codice 200 che sta per "OK", può inviare una risposta 503 con un'intestazione "Retry-After" (riprovare più tardi). Questa soluzione potrebbe essere particolarmente utile nei casi in cui debbano essere effettuati degli aggiornamenti nel database per ritardare le richieste.
Il repository può specificare un periodo di tempo di attesa prima del prossimo tentativo da parte dell'harvester.

A discrezione dei repository è possibile inviare un codice di stato 403 "Forbidden" (Vietato) al posto del 503, nel caso in cui gli harvester non rispettino i tempi di attesa stabiliti dal codice 503 precedentemente inviato. In effetti questa soluzione è al quanto drastica e dovrebbe essere adottata solamente nei casi in cui le performance sono particolarmente critiche.

Riassumendo, bisognerebbe adottare un controllo di flusso che rispetti nell'ordine le seguenti possibili fasi:

- *Codice di stato HTTP 200* - eventualmente con un elemento resumptionToken.
- *Codice di stato HTTP 503* - con un appropriato valore Retry-After se le

successive richieste sono troppo ravvicinate o se il server è troppo carico.

- *Codice di stato HTTP 403* - se le richieste non aderiscono al tempo di attesa precedentemente specificato.

3.7.2 Load Balancing

E' possibile costruire gerarchie di repository cooperanti anche se non specificato dal protocollo.
Vi sono casi di repository con carichi di lavoro particolarmente elevati che indirizzano ulteriori richieste a dei repository mirror, in tali casi il load balancing può essere realizzato attraverso:

- *Load balancing basato sull'HTTP*
- *Load balancing basato sul DNS*

L'utilizzo del load balancing è raccomandato nei casi di repository che siano spesso sovraccarichi e che aderiscano ai dettagli del protocollo OAI-PMH.

3.7.2.1 Load Balancing basato sull'HTTP

Per quanto riguarda la tecnica di load balancing qui discussa possiamo considerare il seguente esempio:

consideriamo la richiesta:

```
http://xXx.org/?verb=ListRecords&set=yyy&from=2004-05-03
```

se il repository xXx.org è particolarmente indaffarato, può scegliere di inviare un codice 302 per reindirizzare la richiesta ad un repository mirror, another.xXx.org, con la seguente intestazione Location:

```
http://another.xXx.org/?verb=ListRecords&set=yyy&from=2004-05-03
```

A questo punto il mirror potrà gestire la richiesta eventualmente generando una lista incompleta, le successive richieste dovranno però essere riferite al mirror another.xXx.org in questione.

E' bene notare come per la richiesta con verbo Identify inoltrata ad another.xXx.org, la risposta debba contenere come valore di baseURL xXx.org ossia il nome del repository di partenza.

La stessa considerazione può essere fatta per tutte le altre richieste, in particolar modo la parte della risposta racchiusa dal tag request dovrà contenere come baseURL il nome del repository d'origine (xXx.org).

```
<?xml version="1.0" encoding="UTF-8" ?>
<OAI-PMH xmlns="http://www.openarchives.org/OAI/2.0/"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/
         http://www.openarchives.org/OAI/2.0/OAI-PMH.xsd">
<responseDate>2002-05-01T19:20:30Z</responseDate>
<request verb="GetRecord" identifier="oai:xXx.org:
                                      th/9901"
                          metadataPrefix="oai_dc">
         http://xXx.org/OAI-script
</request>
<GetRecord>
   <record>...</record>
</GetRecord>
</OAI-PMH>
```

Da notare che il mirror another.xXx.org può a sua volta reindirizzare, tramite il

codice di stato 302, la richiesta ad un altro repository, nel seguente modo:

```
http://yetanother.xXx.org/?verb=ListRecords&set=yyy
                          &from=2004-05-03
```

E' importante che gli implementatori assicurino che reindirizzamenti circolari, del tipo mostrato in figura 3, non si verifichino in alcun caso:

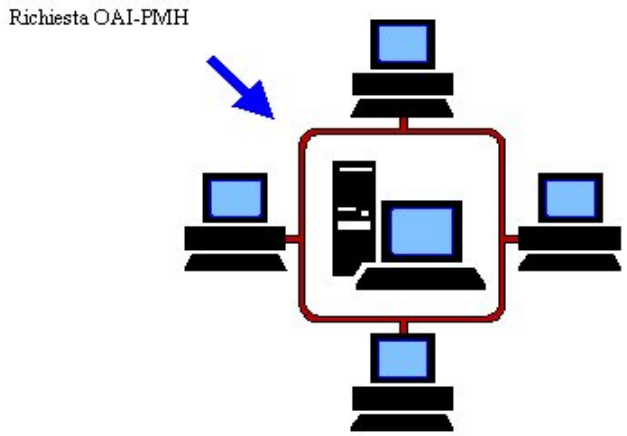

Figura 3 – Caso tipico del reindirizzamento circolare di una richiesta OAI-PMH

3.7.2.2 Load Balancing basato sul DNS

Questa soluzione può essere particolarmente semplice da implementare purchè si abbia a disposizione un distributore di DNS. Tale host potrebbe ad esempio avere

come nome di dominio xXx.org e risolvere la base URL della richiesta in una delle seguenti: axXx.org, bxXx.org, cxXx.org, ecc., che presumibilmente si riferiscono a macchine separate. In tal modo la scelta del repository che processerà la richiesta ricadrà su quello più opportuno, mentre l'harvester accederà a quest'ultimo in maniera random.
I repository che intendono implementare tale strategia devono assicurare una stretta sincronizzazione tra i mirror che risponderanno alle richieste.
Come visto nel paragrafo precedente, anche in questo caso sarà necessario specificare come baseURL, per la richiesta Identify e per le parti delle altre risposte contenenti il tag request, quello del repository di partenza (xXx.org).

Rispetto alla tecnica basata sull'HTTP, la soluzione qui trattata risulta essere meno potente in quanto un reindirizzamento sarà sempre effettuato dal server DNS.
La figura 4 seguente mostra il reindirizzamento di una richiesta OAI-PMH dal server DNS xXx.org al "mirror provider" axXx.org:

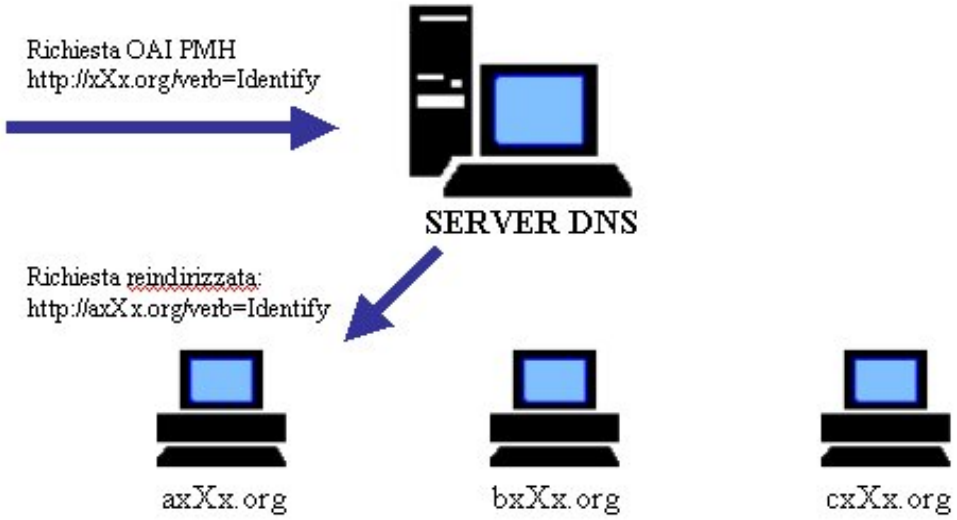

Figura 4 – Reindirizzamento di una richiesta OAI-PMH basata sul DNS

3.8 GESTIONE DEGLI ERRORI

Per quanto riguarda i vari tipi di errori di risposta stabiliti dall'OAI-PMH si faccia riferimento al paragrafo 2.10.9.

Le varie risposte di errore dovranno contenere uno o più elementi *error*. Benché non richiesto dal protocollo, si raccomanda che gli implementatori dei vari repository includano messaggi di errore (contenuti tra i tag error) dettagliati e utili. In più l'elemento error dovrà adottare l'attributo obbligatorio *code*.

I repository in ogni risposta devono includere un solo elemento error anche se

esistono ulteriori condizioni di errore.
Per scopi di debug invece, si raccomanda che i repository riportino tanti errori quanti ne identificano. La risposta in questo caso dovrà prevedere più elementi error ed è preferibile separare anche gli errori multipli con lo stesso codice d'errore. Ad esempio:

```
<error code="badArgument">Illegal argument 'arg1'</error>
<error code="badArgument">Illegal argument 'arg2'</error>
```

ed è da preferire a:

```
<error code="badArgument">Illegal arguments 'arg1', 'arg2'</error>
```

3.9 TESTING

Una volta realizzato il proprio data provider, conforme al protocollo OAI-PMH, bisognerà testare l'applicazione inviando delle richieste all'interfaccia corrispondente e conseguentemente verificarne i risultati.
Ciò può essere fatto utilizzando l'applicazione usata all'università del Vermont "Repository Explorer" raggiungibile all'indirizzo http://oai.dlib.vt.edu/cgi-bin/Explorer/oai2.0/testoai/, che è un tester interattivo e automatico. Esso permette di fornire gli argomenti della richiesta attraverso dei form HTML. Le risposte vengono validate in conformità alle specifiche OAI-PMH.

3.10 REGISTRAZIONE

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

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

In tal caso bisognerà fornire l'URL della propria implementazione in modo da consentire all'OAI di effettuare dei "test di conformità" (che possono includere: condizioni di errore, comportamenti non corretti, ecc.). Se dovessero essere presenti problemi tecnici, questi saranno notificati al sottoponente. In caso dovesse risultare conforme ai requisiti richiesti, esso sarà aggiunto alla lista ufficiale. Successivamente l'OAI si riserva di poter effettuare dei controlli periodici sui data provider registrati.

BIBLIOGRAFIA

[B3] – Buttà Basilio, *Metadati negli Open Archive: tecniche di conversione dei formati dei record nel formato MARC21 utilizzato da CDSware*, tesi di laurea in informatica, Università degli Studi di Messina, A.A. 2003-2004 (relatore Puccio L., correlatore De Robbio A.).

SITOGRAFIA

[S3] - http://www.oaforum.org/tutorial/

(Tutorial on line per coloro che si avvicinano agli open archive a cura dell'Open Archives Forum, un'iniziativa di studio approvata dall'OAI)

[S4] - http://www.openarchives.org/OAI/2.0/guidelines.htm

(Sezione del sito ufficiale dell'OAI riguardante i requisiti per l'implementazione dei data provider)

[S5] - http://www.ietf.org/rfc/rfc2616.txt

(RFC (Request For Comment) che presenta le specifiche tecniche del protocollo HTTP 1.1)