Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica Magistrale

Migrazione verso una architettura REST di un applicativo per l'Inter Library Loan

Tesi di Laurea in Sistemi Distribuiti

Relatore:
Chiar.mo Prof.
Alessandro Amoroso

Presentata da:
Vincenzo Alaia

Correlatore:
Dott. Alessandro Tugnoli

II Sessione
Anno Accademico 2012-2013

Ai miei genitori.

What is life?

It is the flash of a firefly in the night.

It is the breath of a buffalo in the wintertime.

It is the little shadow which runs across the grass and loses itself in the sunset.

(Crowfoot - Indiani d'America)

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

Con la naturale evoluzione delle tecnologie informatiche si sono avuti grandi cambiamenti nell'ambito dello sviluppo software, sopratutto perché il modello di comunicazione tra singoli utenti ed aziende sta migrando verso uno scenario distribuito in cui applicazioni presenti su diversi dispositivi riescono a comunicare tra loro scambiandosi informazioni di qualsiasi genere attraverso la rete internet. Si è passati quindi da un modello *human-centric* dove l'utente effettua la richiesta e l'esecuzione della pagina, a un modello *application-centric* dove è l'applicazione che effettua le richieste e quindi sono le applicazioni che comunicano tra loro. E' nata così la necessità di sviluppare un'architettura a servizi che permetta e garantisca l'integrazione e l'interoperabilità tra sistemi eterogenei. L'industria dell'informatica, ha già affrontato questi problemi sviluppando i cosiddetti Middleware nati agli inizi degli anni '80 come soluzione al problema di connettere le nuove applicazioni ai vecchi sistemi pre-esistenti. Con il termine Middleware si intende "un software di connessione che consiste in un insieme di servizi e/o di ambienti di sviluppo di applicazioni distribuite che permettono a più entità (processi, oggetti, ecc.), residenti su uno o più elaboratori, di interagire attraverso una rete di interconnessione a dispetto di differenze nei protocolli di comunicazione, architetture dei sistemi locali, sistemi operativi, ecc." [1]

I Middleware funzionano bene quando le diverse applicazioni che devono comunicare tra loro sono sviluppate dalla stessa azienda. Nei sistemi informativi aziendali però spesso nasce l'esigenza di far comunicare applicazioni sviluppate da aziende diverse. Inoltre alcuni Middleware per ottenere l'indi-

pendenza dal sistema operativo e hardware impongono una limitazione sul linguaggio di programmazione da utilizzare (es. RMI che è indipendente dal sistema operativo ma è utilizzabile solo da Java). Un'approccio più moderno per la risoluzione di questi problemi sono i Web Services, che possono essere definiti come un'evoluzione logica dei componenti software e dei Middleware. Un Web Service si basa su due concetti fondamentali: il servizio "un componente computazionale autonomo che incapsula una determinata funzionalità ed è in grado di fornirla attraverso la rete" ed un set protocollare (XML, SOAP, WSDL, UDDI) supportato da un gran numero di organizzazioni per permettere ad utenti e applicazioni eterogenee di lavorare e interoperare insieme indipendente dalla piattaforma hardware e software attraverso l'utilizzo di Internet. Nei prossimi capitoli verrà approfondito il concetto di Web Service mostrando i diversi stili di Servizio Web, da RPC a SOAP a REST analizzando due tipologie di architetture: *SOA - Service Oriented Architecture* e *ROA - Resource Oriented Architecture*.

La trattazione si divide in 4 capitoli, in particolare nel primo capitolo verranno introdotti i Servizi Web definendo i protocolli e gli standard che sono alla base di questi sistemi software ponendo le basi per gli argomenti trattati nei capitoli successivi. A tale scopo in questo capitolo vengono analizzate nel dettaglio due tipologie di architettura: Service Oriented e Resourse Oriented mostrando le caratteristiche di ognuno e i possibili benefici derivanti dal loro utilizzo. In particolare verrano trattati nel dettaglio il protocollo SOAP e l'architettura REST in modo da riuscire a comprendere meglio le modalità di comunicazione delle diverse tipologie di servizi web.

I secondo capitolo sarà dedicato alla descrizione delle tecniche e dei protocolli esistenti per l'autenticazione nei Servizi Web. In particolare verrano definiti gli standard di autenticazione basati su HTTP: http Basic, http Digest e WSSE. Mentre verrà introdotto e spiegato nel dettaglio il protocollo OAuth il quale è stato utilizzato e implementato per gestire l'autenticazione e l'autorizzazione quando vengono effettuate chiamate alle API REST di NILDE.

Nel terzo capitolo invece viene descritto l'ambito di sviluppo di questo lavoro di tesi fornendo una panoramica sui principali software di ILL (Inter Library Loan) e i protocolli di comunicazione e standard di riferimento utilizzati da questi software. Nel dettaglio verrà analizzata l'architettura software di NILDE - Network Inter Library Document Exchang. Il lavoro di questa tesi infatti si basa principalmente sull'estendere l'architettura di NILDE sviluppando e fornendo un insieme di API in modo da rendere disponibili i servizi di NILDE accessibili dall'esterno ad esempio da altri programmi di ILL.

Nell'ultimo capito vengono descritti i principali framework per la realizzazione di servizi REST e verrà spiegato l'implementazione di OAuth su NILDE. L'ultima parte di questo capitolo è dedicata alla progettazione e all'implementazione dei servizi REST sviluppati su NILDE indicando quelle che sono state le scelte sulle tecnologie e sui protocolli utilizzati per la fornitura delle API.

Concludendo si terranno alcune considerazioni personali e alcuni spunti per sviluppi futuri.

Capitolo 1

I Servizi Web

I Web Service, forniscono degli standard per l'interoperabilità tra diverse applicazioni software eseguite su diverse piattaforme e/o framework. Secondo il *W3C - World Wide Web Consortium*[2] un Servizio Web è "un sistema software progettato per supportare l'interoperabilità tra diversi elaboratori su di una medesima rete; tale caratteristica si ottiene associando all'applicazione un'interfaccia software (descritta in un formato automaticamente elaborabile quale, ad es., il *Web Services Description Language - WSDL*) utilizzando la quale altri sistemi possono interagire con il Web Service stesso attivando le operazioni descritte nell'interfaccia (servizi o richieste di procedure remote) tramite appositi "messaggi" di richiesta inclusi in una "busta" (la più famosa è SOAP): tali messaggi di richiesta sono formattati secondo lo standard XML e incapsulati e trasportati tramite i protocolli del Web (solitamente HTTP), da cui appunto il nome web service."

I Servizi Web sono una nuova generazione delle applicazioni web ovvero tutte le applicazioni distribuite web-based. Essi eseguono delle funzioni di qualsiasi genere, da semplici richieste a complessi processi aziendali. Quando i servizi, cioè le risorse astratte che rappresentano la capacità di eseguire queste funzioni vengono sviluppate, altre applicazioni sono in grado di richiamare uno di questi servizi. Si può affermare che i Web Service sono il punto di riferimento nell'architettura di applicazioni distribuite sul web, essi sono infatti

i componenti principali per costruire processi distribuiti. Con l'introduzione dei diversi linguaggi di programmazione orientati al web, le applicazioni web, sono sempre più dipendenti dalla piattaforma con cui vengono sviluppate, per questo i Web Service, vengono considerati la soluzione tecnologica per l'interoperabilità di questi sistemi eterogenei permettendo di sviluppare nuovi servizi ad-hoc. Infatti essi si occupano della logica applicativa e non della logica di presentazione cioè di come viene descritta l'interfaccia. In questo modo chi fruisce di questi servizi si preoccuperà personalmente dello sviluppo della logica di presentazione per mostrare i dati ottenuti servendosi di stili grafici propri.

Per esempio, prendiamo in considerazione Flickr[3] considerato attualmente la migliore applicazione web per le gestione e la condivisione di foto e supponiamo che voglia offrire mediante un Web Service un servizio di condivisione delle foto. In questo modo Flickr si preoccuperà di pubblicare il servizio con le specifiche per l'interrogazione e lo renderà visibile a chiunque voglia utilizzarlo. Ora pensiamo a uno sviluppatore che vuole creare un'applicazione web o un sito web e voglia mostrare ai propri visitatori tutte le foto presenti in Flickr che corrispondono alla parola di ricerca "Computer". Lo sviluppatore, per implementare il servizio, utilizzerà la risorsa messa a disposizione da Flickr e si preoccuperà solo della rappresentazione grafica all'interno della sua applicazione del contenuto delle risposte che otterrà.

In questo scenario, Flickr, può decidere di realizzare il lato server in qualsiasi tecnologia (Java, PHP, .NET). Anche lo sviluppatore è altrettanto libero di usare la tecnologia che preferisce. Anche se entrambi utilizzeranno tecnologie diverse, memorizzando dati su database diversi e girando su sistemi operativi diversi, a livello dei Web Services tali differenze non esistono. Se un domani una delle due parti volesse migrare su un altro sistema, cambiare applicazione, database o architettura, l'altra parte non si accorgerà di nulla: tutto continuerà a funzionare come prima, senza alcuna modifica.

I principali benefici derivanti dall'uso dei Web Services sono:

- ***Interoperabilità*** - Viene definito un set di protocolli standard, che permettono l'interoperabilità e la comunicazione tra applicazioni software eterogenee che risiedono su diverse piattaforme hardware/software.
- ***Riusabilità*** - è possibile riutilizzare il software implementato precedentemente riducendo i costi di sviluppo, testing e manutenzione.
- ***Loose Coupling*** (accoppiamento indipendente) - ogni servizio è indipendente dagli altri servizi che costituiscono l'applicazione. Questo permette di modificare singole parti di un'applicazione senza toccare le aree non interessate.
- ***Accessibilità*** - i Web Services utilizzano HTTP[4] come protocollo di trasporto permettendo alle informazioni di essere scambiate in tempo reale. Tutti i dispositivi, che supportano HTTP, possono accedere ai Web Services e consumarli.

D'altro canto i Web Service presentano performance drasticamente inferiori rispetto ad altri metodi di comunicazione utilizzabili in rete. Infatti come per tutti i sistemi che utilizzano la rete come mezzo di comunicazione si deve tenere in considerazione dei ritardi (delay) che non sono controllabili e variano a seconda delle condizioni del traffico.

I messaggi trasmessi, sono basati sul linguaggio XML che può essere considerato un vantaggio per quanto riguarda la flessibilità e la manutenibilità, ma richiede l'inserimento di un notevole numero di elementi supplementari (elementi XML) indispensabili per la descrizione dell'operazione. Inoltre tutti i messaggi inviati richiedono di essere generati e interpretati ai capi della connessione attraverso un parser XML[1]. Queste caratteristiche dei Web service li rendono poco adatti a flussi di dati intensi o dove la velocità dell'applicazione rappresenti un fattore critico.

[1]Un software in grado di effettuare l'analisi sintattica di un documento XML, ad esempio è in grado di verificare se un documento XML è ben formato.

1.1 Un modello concettuale

Prima di procedere nella trattazione delle tecnologie che caratterizzano un Web Service, è utile chiarire alcuni concetti fondamentali. L'architettura di un Web Service è ben definita ed è sempre la stessa, indipendentemente dai Web Service specifici dei vari fornitori utilizzati. Da un punto di vista orientato ai servizi *SOA - Service Oriented Architecture*, il modello dei Web Services è basato su tre ruoli fondamentali e sulle interazioni tra essi, vedi Figura 1.1.

- ***Service Provider*** - Sviluppa e definisce le specifiche per l'interrogazione dei servizi e li pubblica all'interno dei *Service Registries* oppure li rende direttamente disponibili ai *Service Requestor*. Per esporre un'applicazione come un servizio, il provider deve fornire un accesso basato su un protocollo standard e una descrizione standard del servizio (meta data). Dal punto di vista architetturale, questa è la piattaforma che ospita l'accesso al servizio.

- ***Service Requestor o User*** - E' un'applicazione (Browser o un altro Web Service) che invoca o avvia un'interazione con i servizi resi disponibili dal *Service Provider*. Il *Service Requestor* e il *Service Provider* devono condividere le stesse modalità di accesso e interazione con il servizio.

- ***Service Registry*** - E' un registro, (una directory centralizzata), dove sono depositati tutti i servizi definiti e pubblicati dal *Service Provider*, in modo tale che i *Service Requestor* possono trovare i servizi desiderati. Il *Service Registry* è un ruolo opzionale nell'architettura Web Service perché un *Service Provider* può inviare la descrizione di un servizio direttamente al richiedente (*Service Requestor*).

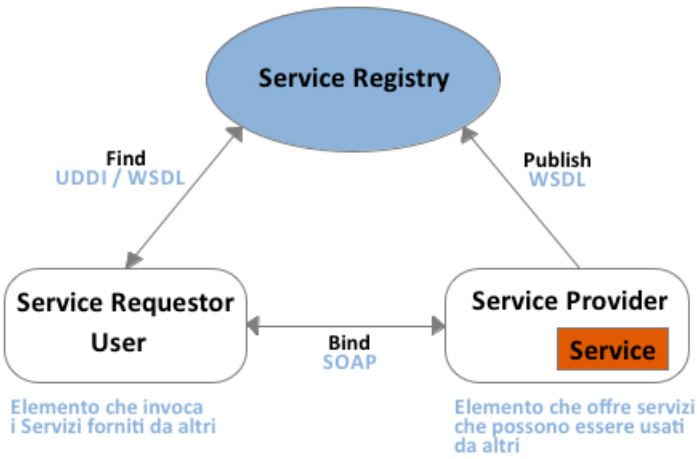

Figura 1.1: Architettura SOA di un Web Service.

Questi tre attori possono essere distribuiti ed utilizzare piattaforme tecnologiche differenti servendosi però di un canale trasmissivo comune. Con l'approccio *SOA - Service Oriented Architecture* è possibile integrarsi con diversi ambienti di sviluppo, permettendo in tal modo di realizzare applicazioni che possono essere utilizzate attraverso diversi dispositivi. E' importante ricordare che SOA è solo un principio architetturale e non è legato a una tecnologia specifica. Molte tecnologie come CORBA e Java RMI possono essere utilizzate quando viene implementata un'architettura di tipo SOA ma tuttavia l'approccio più diffuso per implementare SOA è l'utilizzo di Servizi Web. E' importante notare però che l'uso di tecnologie basate su Web Service non implica l'implementazione di un'architettura SOA, è possibile utilizzare la tecnologia Web Service senza aderire ai principi SOA. Anche se i Web Services stanno diventando uno standard, non sono l'unica tecnologia che può essere utilizzata per implementare SOA.

L'interazione tra i tre componenti appena descritti, avviene attraverso tre tipi di operazioni: pubblicazione dei servizi, ricerca dei servizi e l'invocazione dei servizi. Queste operazioni possono verificarsi singolarmente o in modo iterativo.

Nel dettaglio le operazioni sono:

- ***Publish*** - Il *Service Provider* invia la descrizione del servizio, che ospita, al *Service Registry*. In questo modo il servizio viene pubblicato in modo tale che un *Service Requestor* possa trovarlo. Il *Service Provider* usa *WSLD - Web Services Description Language* per descrivere i dettagli tecnici dei suoi servizi.

- ***Find*** - In questo caso, il *Service Requestor* interroga il *Service Registry* per ottenere la descrizione del servizio richiesto che vuole utilizzare.

- ***Bind*** - Il *Service Requestor* utilizza i dettagli presenti nella descrizione del servizio per contattare il *Service Provider* mediante il quale interagisce con l'implementazione del servizio.

1.2 Tecnologie di base

Il modello di comunicazione illustrato in Figura 1.1, si basa fondamentalmente su degli standard esistenti come: *HTTP - HyperText Transfer Protocol*, *SOAP - Simple Object Access Protocol*, *XML - Extensible Markup Language*, *WSDL - Web Services Description Language* e *UDDI - Universal Discovery Description and Integration*. Ciascun Web Service ha un unico *URI - Uniform Resource Identifier* e i servizi possono essere definiti, descritti e scoperti utilizzando messaggi SOAP che vengono tipicamente trasmessi tramite HTTP. Un Client può essere qualunque dispositivo: un computer, uno smartphone o un tablet che interagisce con il Web Service utilizzando messaggi SOAP. Tramite questo protocollo, una qualunque applicazione può invocare un servizio e grazie a un documento XML è possibile definire i parametri di ingresso e uscita del metodo che vengono trasmessi tramite HTTP al Web Server.

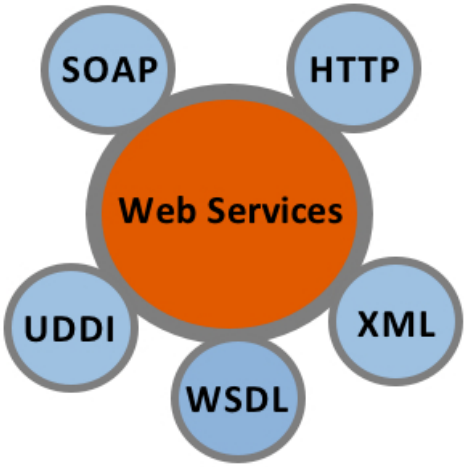

Figura 1.2: Tecnologie di base per un Web Service.

Un Web Service è essenzialmente un insieme di standard basati su XML (vedi Figura1.2) che vengono utilizzati per implementare un'architettura di tipo SOA. Lo standard WSDL viene utilizzato per la descrizione dei servizi, il protocollo SOAP viene utilizzato per il trasporto dei messaggi e UDDI anch'esso basato su XML permette alle aziende la pubblicazione dei propri dati e dei servizi offerti su internet comprese le *API - Application Programming Interface* cioè l'insieme di funzionalità disponibili al programmatore per l'esecuzione di un determinato compito all'interno di un certo programma.

Le organizzazioni più importanti per la definizione degli standard per i Web Service sono: *OASIS - Organization for the Advancement of Structured Information Standards*[5], *W3C - World Wide Web Consortium*[2] e *IETF - Internet Engineering Task Force*[6].

1.2.1 HTTP - HyperText Transfer Protocol

HTTP[4] è l'acronimo di HyperText Transfer Protocol, protocollo dello strato dell'applicazione dei livelli ISO/OSI, ed è utilizzato per la trasmissione delle informazioni sul Web. Prima di procedere con la spiegazione del protocollo HTTP, è opportuno distinguere fra **applicazioni della rete** e **protocolli dello strato dell'applicazione**. Un protocollo dello strato dell'applicazione è solo una parte di un'applicazione della rete. Il Web è un'applicazione della rete che permette di ottenere su richiesta degli utenti risorse e documenti dai server Web. L'applicazione Web è costituita da molti componenti, che comprendono gli standard per i formati dei documenti (es. HTML), i browser Web (es. Firefox), i server Web (es. i server Apache) e un protocollo dello strato dell'applicazione HTTP che definisce come i messaggi vengono passati fra browser e server Web. Il protocollo dello strato dell'applicazione definisce come i processi delle applicazioni, che funzionano su differenti terminali, si scambiano i messaggi. In particolare HTTP definisce:

- i tipi di messaggi scambiati, per esempio, messaggi di richiesta e messaggi di risposta;
- la sintassi dei vari tipi di messaggio, per esempio i campi del messaggio e come questi campi vengono caratterizzati;
- la semantica dei campi, cioè il significato dell'informazione nei campi;
- le regole per determinare quando e come un processo invia o risponde a messaggi;

Ogni transazione HTTP consiste di una richiesta da parte del **client** e una risposta da parte del **server** che generalmente è in ascolto sulla porta 80 usando il protocollo TCP a livello di trasporto. Nel caso dell'HTTP, un browser web implementa il lato client e un server web ne implementa il lato server, l'host che inizia la sessione è etichettato come client. Dato che HTTP utilizza TCP come protocollo di trasporto è necessario richiedere una connessione TCP. Come mostrato in Figura 1.3, questo processo richiede lo scambio

di tre segmenti TCP. Dopo tale scambio, il client può mandare la sua richiesta HTTP. Una volta soddisfatta la richiesta con un opportuno messaggio di risposta, è il server a chiudere la connessione tramite un ulteriore scambio di segmenti TCP (TCP handshake).

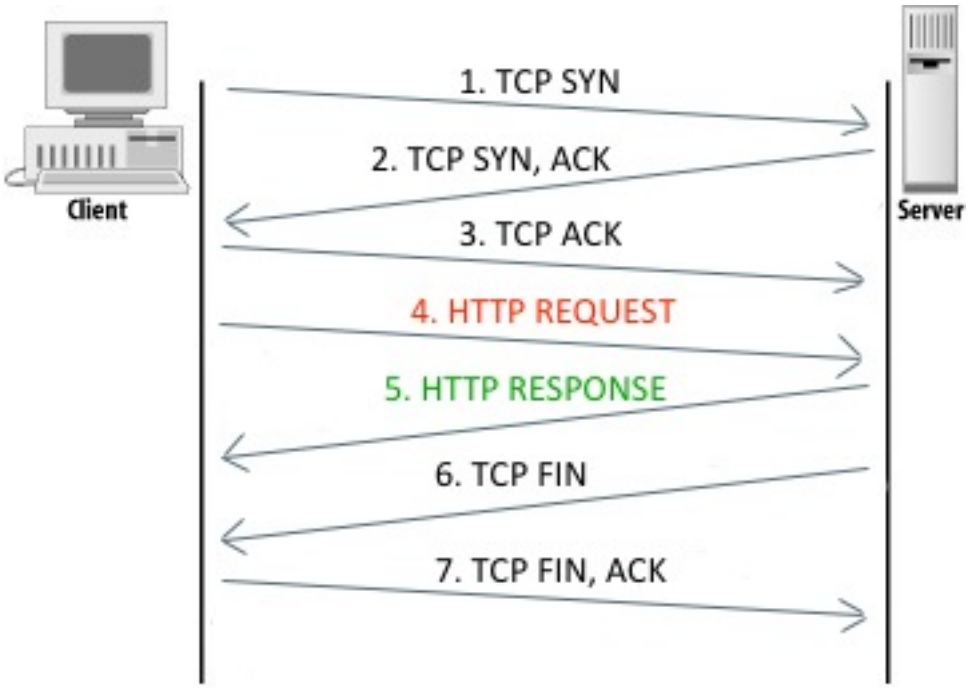

Figura 1.3: Connessione TCP Client-Server.

Il protocollo HTTP può impiegare sia la connessione non permanente sia quella permanente. L'HTTP/1.0 usa la connessione non permanente, mentre HTTP/1.1 utilizza la conessione permanente di default. HTTP/1.0 richiedeva ai client di stabilire una connessione TCP separata per ogni richiesta. Quindi per ogni oggetto (file, immagine, audio, Java Applet) presente nella pagina web richiesta, bisognava stabilire una connessione TCP, riducendo in questo modo le performance del Web. Con HTTP/1.1 è stato eliminato il problema delle connessioni multiple TCP e in questo modo il client può continuare ad usare una connessione TCP esistente dopo che la sua iniziale richiesta è stata soddisfatta dal server. Quando accediamo a una risorsa tramite una URI e HTTP, viene specificata anche l'azione da eseguire su tale

risorsa che viene definita utilizzando un metodo HTTP. Vi sono una serie di metodi HTTP (vedi Tabela 1.1) e ad ogni metodo è associata una semantica che aiuta ad identificare l'azione da eseguire sulla risorsa.

VERBO	**DESCRIZIONE**
GET	Recupera una risorsa identificata da un URI.
POST	Invia la risorsa al server, aggiorna la risorsa nella posizione individuata dal URI
PUT	Invia una risorsa al server, memorizzandola nella posizione individuata dal URI.
DELETE	Elimina una risorsa identificata da un URI.
HEAD	Recupera i metadati di una risorsa identificata dal URI.
TRACE	Traccia una richiesta, visualizzando come viene trattata dal server.
OPTIONS	Richiede l'elenco dei metodi permessi dal server.
CONNECT	Riservato all'uso con proxy capaci di creare un tunnel sicuro (es. SSL)

Tabella 1.1: Metodi HTTP

Dato che HTTP si basa su un meccanismo di richiesta/risposta, è possibile distinguere due tipi di messaggi.

Messaggio di richiesta

Questo tipo di messaggio viene mandato dal client verso il server ed è composto da tre parti (Figura 1.4):

- **Linea di richiesta -** *Request Line,* che contiene tre campi: il metodo richiesto (per esempio GET o POST), l'URI che identifica l'oggetto della richiesta (ad esempio la pagina web che si intende ottenere) e la versione HTTP utilizzata per la comunicazione.
- **Sezione header** - E' composto da un'insieme di informazioni aggiuntive sulla richiesta e/o il client (browser che effettua la richiesta, sistema

operativo, l'host che serve la richiesta, ecc.) attraverso coppie del tipo nome:valore.

- **Body** - Corpo del messaggio, è un'insieme di linee opzionali che permettono l'invio di dati al server.

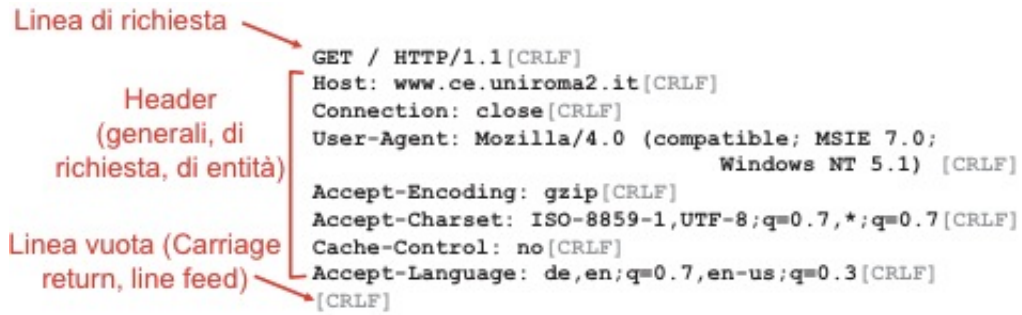

Figura 1.4: Esempio di messaggio di richiesta HTTP

La richiesta deve terminare con una riga vuota, cioè con due "a capo" consecutivi. Come mostrato in Figura 1.4 il metodo richiesto è GET ma HTTP definisce un totale di otto metodi differenti, ognuno dei quali viene descritto nella Tabella 1.1. l'URI è /, che indica una richiesta per la radice della risorsa. Per richieste che non si applicano a nessuna specifica risorsa, come una richiesta TRACE o in alcuni casi una richiesta OPTIONS, il client può usare un asterisco (*) come URI.

Messaggio di risposta

Una volta che il server ha ricevuto dal client una richiesta HTTP, effettua le operazioni necessarie a soddisfarla ed invia una risposta al client. Il messaggio di risposta è di tipo testuale ed è composto da tre parti:

- **Linea di Stato** - *Status Line* che contiene tre campi: versione del protocollo, ID stato che indica il risultato della richiesta e il messaggio di stato corrispondente.

- **Sezione header** - E' composto da un insieme di linee facoltative che permettono di dare delle informazioni supplementari sulla risposta e/o il server. Anche in questo caso come per il messaggio di richiesta, ognuna di queste linee è composta da coppie del tipo nome:valore.

- **Body** - Contenuto della risposta che contiene il documento richiesto o informazioni necessarie per considerare soddisfatta la richiesta.

Un'esempio del formato generale di un messaggio di risposta HTTP è mostrato in Figura 1.5.

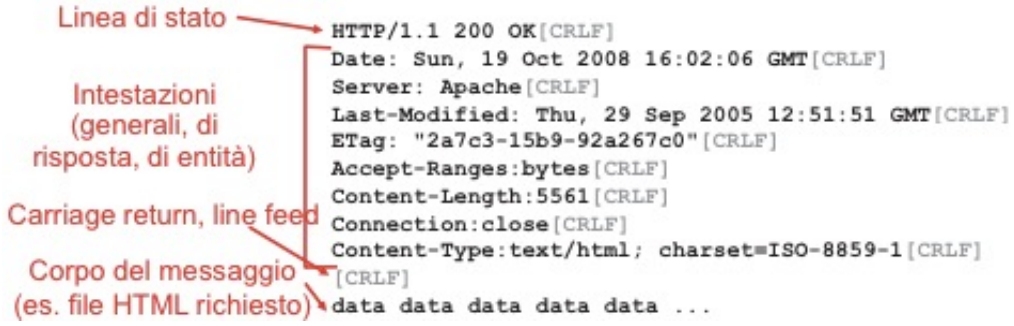

Figura 1.5: Esempio di messaggio di risposta HTTP

Il codice di stato 200, indica che la richiesta del client è stata soddisfatta. OK è il messaggio di stato corrispondente al codice di stato 200. L'ID di stato è un numero di tre cifre che indica appunto lo stato di una richiesta. La prima cifra identifica il tipo di risultato e fornisce un'indicazione di alto livello, le altre forniscono dettagli aggiuntivi.

La seguente tabella riassume i codici di stato HTTP:

Codice	Descrizione
1xx	**Informational** - messaggi informativi
2xx	**Successful** - la richiesta è stata soddisfatta
3xx	**Redirection** - non c'è risposta immediata, ma la richiesta è sensata e viene detto come ottenere la risposta
4xx	**Client error** - la richiesta non può essere soddisfatta perché sbagliata
5xx	**Server error** - la richiesta non può essere soddisfatta per un problema interno del server

Tabella 1.2: Codici di stato HTTP.

HTTP è un protocollo di trasporto *stateless* (senza stato) ciò significa che ogni richiesta viene eseguita in modo indipendente dalle altre e il server non tiene traccia delle richieste che si sono avute in precedenza.

Per questo una richiesta si conclude al momento della chiusura della connessione cioè quando il server comunica che ha finito di servire il documento.

Il server non ha "memoria" di quanto comunicato precedentemente con un certo client: per questo motivo, per realizzare applicazioni web complesse, sono stati sviluppati meccanismi a livello superiore come i cookies per costruire sessioni e permettere al server di recuperare informazioni riguardanti un certo client.

Questi metodi possono essere visti in termini di come è possibile interagire con una risorsa, durante il suo ciclo di vita. GET, HEAD e POST possono essere usati per fare uso di una risorsa durante il suo ciclo di vita. DELETE invece termina il ciclo di vita di una risorsa. Il browser tramite una richiesta HTTP GET, recupera le risorse, e il Web Server risponde con un messaggio riempito con il contenuto della richiesta. In Figura 1.4 è possibile vedere un'esempio di messaggio di richiesta che utilizza il metodo GET.

```
/test/demo_form.asp?name1=value1&name2=value2
```

Come si può notare dal listato, per il metodo GET la *query string* (?name1=value1&name2=value2) è inviata all'interno dell'URL di una richiesta GET, mentre nel metodo POST la *query string* viene inviata nel corpo del messaggio HTTP di una richiesta POST.

```
POST /test/demo_form.asp HTTP/1.1
Host: w3schools.com
name1=value1&name2=value2
```

Un web browser quindi può utilizzare le richieste HTTP POST per inviare i dati compilati (ad esempio un form di registrazione) dagli utenti. In questo caso il Web Server una volta che ha elaborato questi dati risponderà con un messaggio che conterrà l'esito della richiesta. GET e POST sono i metodi HTTP usati più frquentemente nelle richieste HTTP ma è fondamentale capire quando utilizzarli. In Tabella 1.3 vengono riassunte le principali differenze tra questi due metodi.

GET	**POST**
Le richieste possono essere memorizzate nella cache.	Le richieste non sono mai memorizzatie nella cache.
Le richieste rimangono nella cronologia del browser.	Le richieste non rimangono nella cronologia del browser.
Le richieste possono essere salvate nel segnalibri.	Le richieste non possono essere salvate nel segnalibri.
Le richieste hanno limiti di lunghezza (255 Caratteri).	Le richieste non hanno limiti di lunghezza.
Le richieste devono essere utilizzate solo per recuperare i dati.	Le richieste devono essere utilizzate solo per inviare dati.

Tabella 1.3: GET vs POST.

Questi metodi HTTP contribuiscono a fornire un' interfaccia uniforme per l'interazione con le risorse, che come vedremo nei prossimi paragrafi è il principio fondamentale di un'architettura REST.

1.2.2 SOAP - Simple Object Access Protocol

SOAP[7][8] è un protocollo leggero progettato per essere utilizzato in un ambiente decentralizzato e distribuito basato sulla trasmissione di semplici messaggi di richiesta e risposta tra i nodi.

SOAP è sostanzialmente stateless, indipendente dalla piattaforma, indipendente dal protocollo di trasporto (HTTP, SMTP, FTP) e dal sistema operativo e utilizza messaggi XML per lo scambio di informazioni. Queste caratteristiche, permettono a SOAP di risolvere i maggiori problemi di interoperabilità tra applicazioni eterogenee. Inoltre, incapsulando i messaggi SOAP all'interno di HTTP e utilizzando una codifica XML si riescono a bypassare molte problematiche che avevano le precedenti soluzioni proprietarie (es. CORBA) permettendo un uso estremamente flessibile. Infatti essendo il trasporto dei messaggi basato su HTTP i comandi SOAP riescono ad attraversare i firewall delle aziende senza controllo, in quanto appaiono ai Web Server come pagine Web HTML standard.

Ci sono due tipi di richieste SOAP, la prima basata su *RPC - Remote Procedure Call* simile ad altre architetture distribuite e la seconda basata sullo scambio di documenti XML. Nel caso di RPC, le richieste sono sincrone, il client invia un messaggio e attende di ottenere una risposta o un messaggio di errore dal server. Mentre nel secondo caso un documento XML viene passato tra il client e il server all'interno di un messaggio SOAP (vedi paragrafo 1.2.2.1).

1.2.2.1 Messaggio SOAP

Un messaggio SOAP è un documento XML standardizzato e la sua struttura è costituita da tre parti (Figura 1.6). L'elemento principale è il *SOAP Envelope*, che contiene altri due elementi, uno opzionale *Header* e l'altro obbligatorio *Body*.

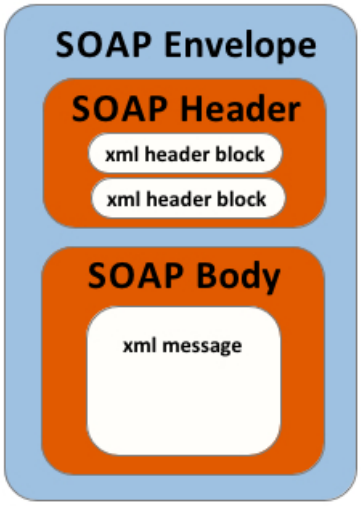

Figura 1.6: Struttura del messaggio SOAP.

- ***SOAP Envelope*** - Identifica il documento come un messaggio SOAP, esprime ciò che si trova nel messaggio e che cosa tratta.
- ***SOAP Header*** - E' un elemento opzionale, può contenere da zero a molti elementi figli personalizzati e viene utilizzato per passare le informazioni relative all'applicazione (per esempio la modalità di autenticazione, di gestione delle transazioni, etc.) che deve essere eseguita dal nodo SOAP una volta ricevuto il messaggio. Viene utilizzato anche per definire messaggi con diversi destinatari nel caso il messaggio dovesse attraversare più punti di arrivo.
- ***SOAP Body*** - Questo elemento viene utilizzato per lo scambio di informazioni obbligatorie destinate al ricevente finale del messaggio. In questo caso il messaggio è contenuto in un formato XML.

- ***SOAP Fault*** - E' un sotto elemento (opzionale) del SOAP Body che viene utilizzato per la segnalazione degli errori.

1.2.2.2 Vantaggi e svantaggi del protocollo SOAP

Molti punti di forza di SOAP sono tuttavia anche delle sue debolezze.

Vantaggi:

- ***Eterogeneità*** - SOAP, può operare su qualsiasi piattaforma, qualsiasi sistema operativo con qualsiasi linguaggio di programmazione per questo è diventato un middleware popolare tra i sistemi legacy.
- ***XML-Based*** - Il protocollo SOAP è basato su XML quindi in un certo senso eredita anche i vantaggi che quest'ultimo linguaggio possiede.
- ***Indipendenza dalla piattaforma*** - Questo permette a un client e a un server di comunicare tra loro indipendentemente dal linguaggio di programmazione con cui sono stati sviluppati.
- ***Indipendenza dal livello di trasporto*** - I messaggi SOAP possono essere spediti su qualsiasi protocollo di trasporto inclusi i protocolli sincroni e asincroni come HTTP, SMTP, FTP aumentando la flessibilità di un Web Service in modo da adattarsi a futuri sviluppi di Internet.

Svantaggi:

- ***Dimensione dei messaggi*** - Un messaggio SOAP ha un alto overhead a causa dell'utilizzo di XML. Questo può diventare un problema quando si trasferiscono un numero elevato di dati a causa delle limitazioni di banda della rete. Infatti con messaggi molto grandi si ha il problema del parsing di tutto l'XML ricevuto aumentando i ritardi e l'overhead tra il mittente e il destinatario.
- ***Implementazione*** - Diversi produttori interpretano le specifiche SOAP in modo diverso creando delle incompatibilità tra le applicazioni. Per

aiutare i produttori è stato creato *WS-I - Web Service Interoperability Organization* un gruppo nato per migliorare l'interoperabilità tra i Web Services.

- ***Sicurezza*** - SOAP non supporta alcun tipo di sicurezza. Un'idea potrebbe essere quella di utilizzare HTTPs che però protegge i dati solo a livello di rete e non protegge il contenuto dei dati. Esistono alcuni draft per la sicurezza dei documenti XML e che possono essere utilizzati all'interno di SOAP. Per le applicazioni punto-a-punto, viene utilizzato SSL per garantire la sicurezza delle connessioni.

- ***Latenza*** - La latenza è un problema difficile da risolvere nei sistemi distribuiti. In SOAP oltre ai classici problemi come bassa larghezza di banda, router congestionati e server sovraccarichi, bisogna aggiungere i ritardi dovuti al parsing del messaggio XML.

- ***Statelessness*** - I messaggi SOAP sono trasmessi in modo unidirezionale da un mittente a un destinatario tramite il protocollo HTTP che è appunto stateless, quindi per aggiungere una sorta di stato tra i singoli messaggi, il campo HEADER è stato lasciato aperto per aggiungere ulteriori tag come ad esempio un identificatore univoco o sessionID utilizzati per rappresentare lo stato tra il client e il server.

1.2.3 XML - Extensible Markup Language

XML - Extensible Markup Language[9][10] è una raccomandazione del *W3C - World Wide Web Consortium* ed è definita come un sottoinsieme di SGML; è un metalinguaggio di markup per lo scambio e l'interusabilità di documenti strutturati su Internet. Insieme ad XML, vi sono altri linguaggi ad esso direttamente correlati:

- **XSL - Extended Stylesheet Language**[11] - è un working group del W3C che si occupa di attribuire significati agli elementi di un documento XML. Esso si compone di tre parti: *XSLT - XSL Transformations* linguaggio per la trasformazione di documenti XML in altri documenti XML, *XML Path Language* un linguaggio utilizzato da XSL per accedere o riferirsi a parti di un documento XML e *XSL-FO XSL Formatting Objects* un linguaggio per la formattazione di documenti basato sulla paginazione, permette la formattazione device-indipendent.

- **XBase, XPath, XPointer, XLink** - linguaggi per la specifica dei link ipertestuali sui documenti XML. XBase specifica un meccanismo per la definizione degli URI di base, XPath[12] specifica un meccanismo per indicare percorsi all'interno di un documento, XPointer[13] permette di riferirsi a parti del documento e infine XLink utilizza i meccanismi di indirizzamento di XPointer per definire link tra documenti.

- **XML Schema**[14] - raccomandazione del W3C, fornisce un supporto alla validazione dei documenti attraverso la definizione di regole di costruzione precise.

- **XML NameSpace [15]** - i namespace sono un meccanismo di XML per consentire la distinzione dei nomi degli elementi utilizzati in un documento e quelli provenienti da un'altro documento, attraverso identificatori univoci (es. un documento che contiene elementi di HTML e altri di XML).

Il punto di forza di XML è la sua versatilità e la sua portabilità ad un gran numero di applicazioni; infatti, essendo un linguaggio di markup descrittivo è possibile utilizzarlo per descrivere ogni cosa; inoltre, essendo leggibile, si presta bene a futuri riutilizzi anche senza conoscere necessariamente la sua sintassi.

La sua interoperabilità con molte applicazioni è resa anche possibile grazie al formato delle codifica usata nei suoi documenti: Unicode ed in modo particolare supporta le codifiche UTF-8[16] e UTF-16[17]; tale codifica permette di scrivere un documento di testo utilizzando caratteri prevenienti da alfabeti differenti, senza dover ricorrere a trucchi particolari per poterli rappresentare correttamente. La sintassi di XML prende spunto da quella di SGML senza però raggiungere lo stesso livello di formalismo, pertanto i documenti XML sono molto più semplici da scrivere.

Nei documenti XML il markup viene utilizzato per conferire un ruolo agli elementi in esso contenuti e tale ruolo può anche essere utilizzato per la verifica della correttezza dello stesso. Un documento XML viene definito **ben formato** se presenta una struttura sufficientemente regolare e comprensibile da poter essere controllata. In particolar modo deve rispettare i seguenti vincoli:

- ad ogni tag di apertura ne corrisponde uno di chiusura e sono ben annidati;
- esiste un solo elemento radice;
- i tag vuoti sono indicati dal simbolo speciale di fine tag (<tag/>)
- tutti gli attributi sono sempre racchiusi tra virgolette
- tutte le entità sono definite.

A ogni documento XML ne viene associato un'altro che ne descrive la grammatica cioè definisce quali elementi sono utilizzabili, la loro struttura e le relazioni fra essi. Un documento XML che rispetta le regole definite da una

grammatica viene detto **valido**. Le soluzioni più diffuse per la creazione di grammatiche per documenti XML sono:

- ***DTD - Document Type Definition*** - è un documento attraverso cui si specificano le caratteristiche strutturali di un documento XML attraverso una serie di 'regole grammaticali'. In particolare definisce l'insieme degli elementi del documento XML, le relazioni gerarchiche tra gli elementi, l'ordine di apparizione nel documento XML e quali elementi e quali attributi sono opzionali o meno.
- ***XSD - XML Schema Definition*** - come il DTD, serve a definire la struttura di un documento XML. Oggi il W3C consiglia di adottarlo al posto del DTD stesso, essendo una tecnica più recente ed avanzata.

La distinzione in documenti validi e ben formati si riflette nelle applicazioni che dovranno dunque disporre di parser validanti e non. Esistono due diverse tipologie di parser XML a seconda del modello di elaborazione del documento:

- **Parser DOM - Document Object Model** - è un modello gerarchico, fornisce l'accesso ai singoli elementi dopo aver completato il parsing di tutto il documento. DOM è una serie di raccomandazioni del W3C ed è disponibile in tutte le architetture tramite API - Application Programming Interface.
- **Parser SAX - Simple API for XML** - è un modello ad eventi al verificarsi dei quali è possibile attivare delle funzioni che gestiscono l'elemento incontrato. SAX non è uno standard ma una serie di API originarie per Java ed ora disponibili anche per altre architetture.

Un documento XML è come detto prima, un documento SGML e come tale ha una parte dedicata alle dichiarazioni, una parte di DTD ed infine contiene l'istanza del documento vera e propria. Un documento XML può includere una dichiarazione che specifica caratteristiche opzionali del documento come la codifica utilizzata e la versione di XML.

Un esempio di dichiarazione è la seguente:

```
<?xml version ="1.0" encoding="UTF-8" ?>
```

Il DTD viene utilizzato per specificare le regole che permettono di verificare la correttezza strutturale di un documento. In tale dichiarazione vanno dunque elencati tutti gli elementi che compongono il documento, i loro attributi ed il contesto in cui possono apparire. XML al contrario di SGML permette la definizione di documenti che siano strutturalmente corretti pur non avendo un DTD. Un esempio di DTD è il seguente:

```
<!DOCTYPE records [
        <!ELEMENT records (record+)>
        <!ELEMENT record  (autore+titolo)>
        <!ELEMENT autore  (CDATA)>
        <!ELEMENT titolo  (CDATA)>
        <!ATTLIST titolo
                lang (it|en|fra)        #REQUIRED
        >
]>
```

Questo DTD, espreme che il documento deve essere costituito da un unico elemento radice **records** che ha almeno un elemento **record**; il **record** deve contenere almeno un **autore** e un **titolo** in questo preciso ordine. Inoltre indica che autore e titolo possono contenere solo caratteri stampabili. Infine viene dichiarato un attributo **lang** di **titolo** che può assumere tre valori ed è un attributo obbligatorio.

Il corpo del documento XML, è composto da un insieme di elementi, entità e commenti, strutturati ad albero e che possono essere definiti liberamente dall'utente aggiungendo opzionalmente attributi all'interno degli elementi. Un'attributo aggiunge informazioni più specifiche a un determinato elemento ad esempio per denotare una sua proprietà intrinseca.

Una istanza lecita per il DTD sopra riportato potrebbe essere la seguente:

```
<?xml version ="1.0" ?>
<records>
        <record>
                <autore>Helbert Van de Sompel</autore>
                <autore>Carl Lagoze</autore>
                <titolo lang="en">The Santa Fe Convention of
                the Open Archives Initiative
                </titolo>
        </record>
        <record>
                <autore>Stevan Harnad</autore>
                <titolo lang="en">Free at Last: The future of
                pre-reviewed journals
                </titolo>
        </record>
</records>
```

Per quanto riguarda la sintassi, viene fatta distinzione tra lettere maiuscole e minuscole nel markup e tutti gli attributi devono essere racchiusi tra virgolette. Un'ulteriore caratteristica di XML è la gestione del whitespace; in XML viene utilizzata la semplice regola che tutto ciò che non è markup sono dati da passare all'applicazione, pertanto vengono riportati anche i whitespace.

E' possibile definire XML come le fondamenta per i Web Services, il quale gli attribuisce molti punti di forza. Come vedremo nei prossimi paragrafi è quella tecnologia dei Web services su sui si basano tutte le altre tecnologie.

1.2.4 WSDL - Web Services Description Language

WSDL[18] è un file XML che descrive i dettagli tecnici (nomi dei metodi, argomenti, tipi di dati) di come implementare un servizio web. Più precisamente un documento WSDL contiene le seguenti informazioni su un determinato Web Service:

- le operazioni messe a disposizione dal servizio;
- il protocollo di comunicazione da utilizzare per accedere al servizio;
- il formato dei messaggi;
- i vincoli del servizio;
- l'endpoint del servizio che solitamente corrisponde all'indirizzo in formato URI che rende disponibile il Web Service.

L'endpoint del servizio e le operazioni offerte, vengono specificate in un documento WSDL mediante una parte astratta di un descrittore WSDL la quale specifica un'insieme opzionale di tipi (TYPES), un insieme di messaggi (MESSAGES), un'insieme di operazioni (OPERATIONS) e uno o più PORT TYPE. Mentre una parte concreta di un descrittore WSDL specifica il BINDING cioè il protocollo di comunicazione utilizzato per un certo PORT TYPE (es. HTTP, SMTP) e il SERVICE.

Il listato seguente mostra appunto un esempio di struttura di un documento WSDL in cui gli elementi **types**, **message** e **portType** rappresentano la parte astratta mentre gli elementi **binding** e **service** rappresentano la parte concreta.

```
<definition>
        <types>
                i tipi di dati che verranno trasmessi
        </types>
        <message>
                i messaggi scambiati
        </message>
        <portType>
                le operazioni (funzioni) supportate
        </portType>
        <binding>
                come si scambiano i messaggi (dettagli SOAP)
        </binding>
        <service>
                dove trovare il servizio
        </service>
</definition>
```

Per ricevere il documento WSDL, il Service Requestor, può effettuare una ricerca presso un catalogo di Web Service e quindi scaricarlo dal sito del Service Provider o anche riceverlo per posta elettronica. Come mostrato in Figura 1.7, una volta ottenuto il documento con la descrizione del servizio, lo sviluppatore tramite l'utilizzo di un tool può generare il modulo client (detto proxy) da incorporare nell'applicazione che consente di comunicare attraverso il protocollo SOAP, con il Web Service.

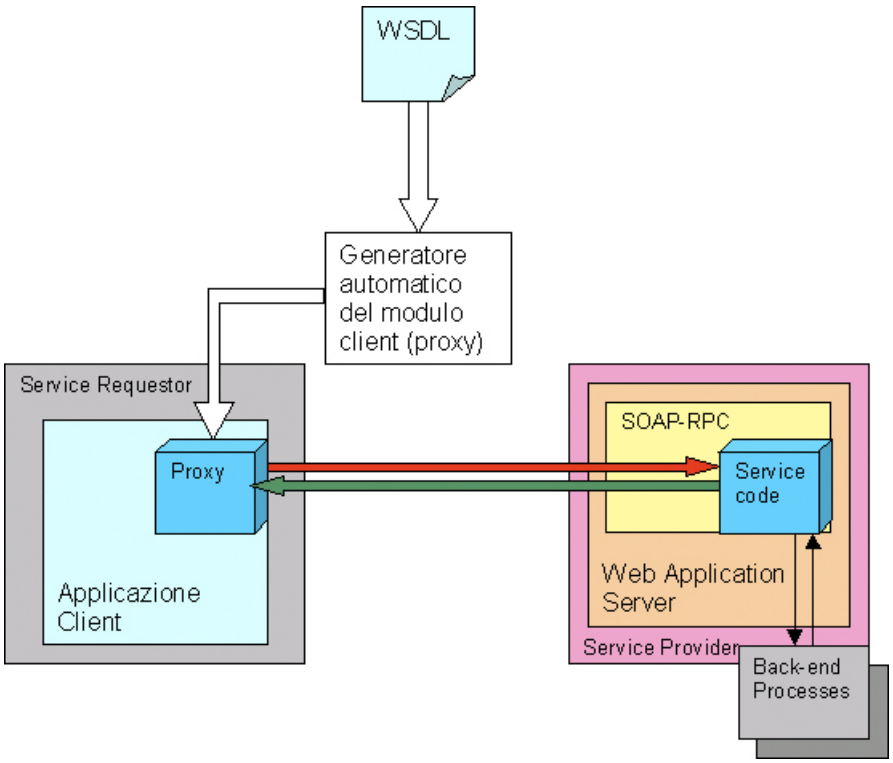

Figura 1.7: Integrazione di un Web Service in un'applicazione a partire dalla sua descrizione.

1.2.5 UDDI - Universal Discovery Description and Integration

Un servizio web per seguire un'architettura *SOA - Service Oriented Architecture* deve essere pubblicato su un registro e deve poter essere ricercato da chiunque ne abbia bisogno. UDDI[19] si comporta come un server dei nomi di dominio *DNS - Domain Name Server* solo che al posto di risolvere i nomi di dominio, risolve i nomi dei servizi. Quindi possiamo definirlo una directory distribuita a livello mondiale (Web based), una specie di pagine gialle elettroniche alle quali un *Service Requestor* può rivolgersi per cercare i Web Services, le informazioni su una certa azienda o avere accesso alle informazioni necessarie ad interfacciarsi con il servizio (WSDL), oppure i *Service Provider* possono pubblicare il loro profilo e i loro servizi.

Come tutti i precedenti standard anche UDDI si basa su linguaggio XML ed utilizza il protocollo SOAP per la comunicazione da e verso l'esterno. Tutti i servizi contenuti nel registry devono fornire tutte le informazioni necessarie per garantire ad un *Service Provider* di poter interagire con lui.

Le informazioni relative a un particolare servizio sono suddivise concettualmente in tre parti o pagine:

- ***Pagina bianca*** - Queste pagine contengono le informazioni base sul fornitore del servizio (nome, telefono, URL) e una panoramica dei principali servizi che fornisce. Utile per trovare un web service di cui si conoscono già alcuni dettagli;

- ***Pagina gialla*** - In cui vengono categorizzati i servizi in base a tassonomie standardizzate. Dato che un'azienda può fornire più servizi, possono esistere più pagine gialle associate ad una pagina bianca.

- ***Pagina verde*** - Le pagine verdi sono progettate per contenere informazioni tecniche sul servizio web e su come interfacciarsi con esso.

In realtà, come mostrato in Figura 1.8, le informazioni memorizzate da UDDI sono suddivise in cinque parti, ognuna appartenente ad una delle tre pagine precedentemente descritte e definite attraverso XML.

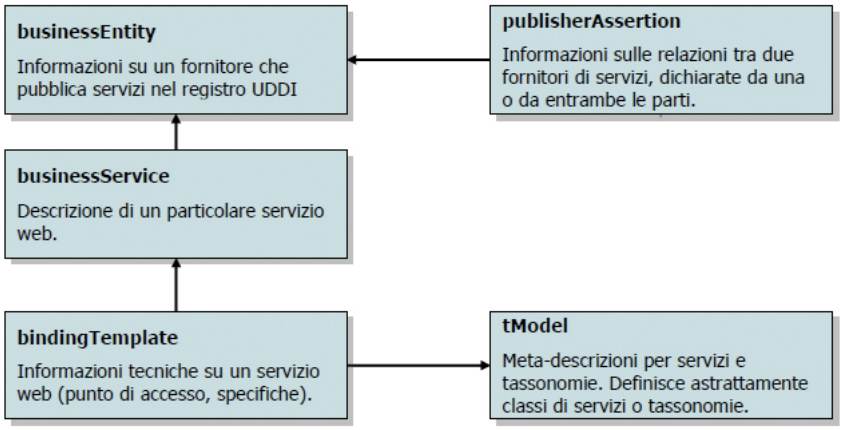

Figura 1.8: Struttura registrazione UDDI.

- *businessEntity* - racchiude le informazioni sull'azienda per esempio, il nome dell'azienda, le informazioni di contatto e il tipo di settore in cui opera l'azienda (Pagina bianca).

- *businessService* - racchiude informazioni sul servizio pubblicato dall'azienda come per esempio nome, descrizione, categoria di appartenenza. Ogni *businessService* è figlio di una singola struttura *businessEntity* (Pagina gialla).

- *bindingTemplate* - racchiude le informazioni tecniche sul servizio, utili per connettersi e comunicare con esso (Pagina verde).

- *pubblisherAssertion* - contiene informazioni relative a relazioni tra i fornitori di servizio permettendo di creare delle relazioni tra differenti aziende registrate presso il registro UDDI realizzando così delle comunità (Pagina bianca).

- *tModel* - è una descrizione astratta del servizio, per modellare informazioni descrittive del comportamento del servizio, tipi di specifiche utilizzate o standard a cui un servizio è conforme.

Per garantire l'accesso e fornire un modo semplice per interagire con i dati memorizzati in un registry UDDI da ogni piattaforma, vengono utilizzate le API UDDI che si suddividono in tre categorie:

- *inquiry* - permettono agli utenti di effettuare una ricerca all'interno di un registro;

- *publishing* - permettono ai fornitori di inserire, modificare e cancellare informazioni su se stessi e su i propri servizi;

- *replication* - permettono la comunicazione tra server per la gestione e sincronizzazione dei registri.

1.3 Servizi Web - principali architetture

Una classificazione delle principali architetture per i Web Service potrebbe essere quella di suddividerle in base al modo con cui comunicano con i client. In particolare le diverse architetture si possono suddividere in[7]:

- ***Servizi REST*** - rispettano i vincoli del paradigma REST;
- ***Servizi RPC / SOAP*** - basati su chiamate a funzione remota;
- ***Servizi Ibridi*** - sfruttano caratteristiche di entrambi gli stili REST e RPC.

1.3.1 Servizi REST

Questi tipi di servizi si basano sul concetto di risorsa e non sul concetto di servizio e di chiamata remota come avviene nei Web Service SOAP, per questo si parla di *ROA - Resource Oriented Architecture*.

Per **risorsa** si intende un qualsiasi elemento oggetto di elaborazione (un cliente, un libro), un qualsiasi oggetto su cui è possibile effettuare operazioni.

Per **rappresentazione** si intende una descrizione dello stato corrente o voluto di una risorsa che può essere inviata dal Web Service al client preferibilmente in un formato standard ad esempio utilizzando XML o JSON.

Tramite l'URI è possibile identificare univocamente ciascuna risorsa. Un esempio di possibile identificatore di risorsa potrebbe essere:

```
http://www.provasito.it/clienti/456
```

tramite il quale è possibile accedere alla risorsa "456" che corrisponde in questo caso a un possibile cliente. Attraverso un URI scritto in questo modo, è possibile capire subito che tipo di operazione si vuole eseguire sulla risorsa (in questo caso una GET), infatti è semplice intuire che l'informazione richiesta è di lettura e che si tratta di una ricerca tra tutti i clienti con id "456". Questo mostra che i sistemi REST hanno delle proprietà desiderabili come la scalabilità, la modificabilità e la semplicità che in fase di progettazione e

sviluppo potrebbero avere molta importanza in modo tale da preferire questo tipo di servizi al posto di altri tipo RPC. Nel Paragrafo 1.4 verrà trattato e approfondito in maniera più dettagliata il paradigma architetturale REST focalizzando lo studio sui servizi RESTful e sul concetto di risorsa.

1.3.2 Servizi RPC-SOAP

SOAP-RPC è l'implementazione di una chiamata di procedura remota *RPC - Remote Procedure Call* in cui una funzione in un computer remoto è chiamata come se fosse una funzione locale nascondendo quindi all'utilizzatore i dettagli della comunicazione. In un'architettura di questo tipo, il client invoca il Web Service inviando parametri e ricevendo dei valori di ritorno. I servizi SOAP-RPC seguono una semantica "chiamata / risposta" per questo sono definiti sincroni cioè il client invia la richiesta al server e attende la risposta finché la richiesta non viene elaborata completamente. La richiesta ovvero il messaggio verso la funzione remota, viene formattato (di solito in XML), impacchettato e spedito al nodo della rete che si occuperà di interpretare il messaggio e rispondere in maniera adeguata. In questo caso al contrario dei servizi REST, non vengono sfruttati tutti i metodi HTTP in quanto le richieste non trasportano le informazioni sul metodo da eseguire ma vengono inviati nel messaggio XML tramite una richiesta di POST.

Questo tipo di API hanno un singolo endpoint e quindi tutte le richieste vengono inviate allo stesso URL, ciò che cambia è il contenuto del messaggio XML.

Il server una volta ricevuto il messaggio XML, lo passa a un listener XML-RPC il quale fa il parsing dell'XML per avere il nome del metodo e i parametri dopo di che chiama il metodo appropriato passandogli i parametri. Il metodo restituirà una risposta che viene impacchettata come XML e il server restituirà tale XML come risposta alla richiesta del client. Di conseguenza il client farà il parsing dell'XML ricevuto per estrarre il risultato e lo passerà al programma client.

L'utilizzo di HTTP implica che le richieste siano sincrone e stateless.

1.3.3 Servizi Ibridi

In questa categoria di servizi, troviamo tutta quella fascia di servizi che seguono i principi definiti da REST (Client - server, Stateless, Cacheable, Layered system, Code on demand, Uniform interface) ma non sono classificabili come *Resource Oriented*. Questa tipologia di servizi, cerca di inserire all'interno dell'URI tutte le informazioni utili all'esecuzione della richiesta. Per esempio se prendiamo in considerazione l'URI utilizzata da Flickr,

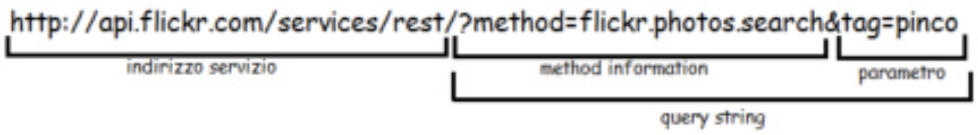

Figura 1.9: Esempio Flickr URI.

è possibile notare che nonostante venga indicato nell'URI la parola "REST", questo tipo di chiamata è stata progettata come un servizio RPC. Come si può vedere dalla Figura 1.9, in questo caso tramite la query string è possibile conoscere sia l'operazione che il client intende eseguire e sia i parametri richiesti (*Scoping Information*) per eseguire tale operazione i quali vengono inviati all'interno dell'url allo stesso modo di come verrebbero inviati utilizzando un'architettura di tipo REST. Ma in questo caso i metodi HTTP non mappano le effettive operazioni che poi saranno eseguite.

Dall'esempio di Figura 1.9 non è facile capire se una richiesta è di tipo REST o RPC ma è semplice capire che si sta eseguendo una ricerca di immagini taggate pinco. Essendo il nome del metodo da eseguire all'interno dell'URI, potrebbe far pensare che sia un servizio rpc, ma è anche vero che quest'informazione può essere interpretata come un'informazione di scooping che è conforme a un web service ti tipo REST. La differenza si nota però quando si vanno ad eseguire operazioni di modifica come ad esempio la delete.

```
http://api.flickr.com/services/rest/?method=flickr.photos.delete&id=xxxxx
```

Come si può notare in questo esempio, le API di Flickr richiedono l'uso del metodo HTTP GET anche quando si vogliono modificare dei dati.

Infatti per eliminare una foto si dovrà inviare una richiesta GET specificando nell'URI il metodo "flickr.photos.delete". A questo punto è evidente che questo tipo di architettura non è REST, si stà cercando di eliminare una risorsa con id xxxx non ricorrendo al metodo HTTP DELETE ma utilizzando una GET. Per questo le API di Flickr possono essere definite Servizi Ibridi in quanto sono RESTful quando un client vuole recuperare dei dati attraverso il metodo GET, mentre sono RPC quando il client vuole modificare uno specifico dato. Un servizio ibrido quindi utilizzerà sempre il metodo GET e avrà URI diverse per la stessa porzione di dati.

```
GET /items/id?method=delete
GET /items/id?method=update
```

Mentre un servizio RESTful avrà sempre la stessa URI per un dato indipendentemente dal tipo di operazione che verrà effettuata su di esso.

```
GET /items/id
PUT /items/id
```

1.4 REST - REpresentational State Transfer

Il termine REST che indica un paradigma architetturale venne introdotto nel 2000 nella tesi di dottorato di Roy Fielding[20] (uno degli autori del protocollo HTTP). REST può essere definito come una filosofia o un'insieme di principi piuttosto che un protocollo a sé stante come SOAP che i sistemi distribuiti eterogenei dovrebbero avere per soddisfare determinate caratteristiche come, ad esempio la scalabilità. Questi insieme di vincoli che caratterizzano il design pattern denominato REST sono:

- ***Client / Server*** - Fornisce il concetto di separazione delle competenze in cui il server offre una o più risorse e rimane in attesa di richieste da parte del client che vuole accedere alla risorsa. Queste richieste possono essere accettate o meno.

- ***Stateless*** - Ogni richiesta deve contenere tutte le informazioni necessarie per essere elaborata, senza aver bisogno di riferimenti alle precedenti richieste. Quindi le interazioni tra client e server devono essere senza stato in modo da migliorare la scalabilità e l'affidabilità.

- ***Caching*** - Al fine di migliorare l'efficenza della rete, è possibile memorizzare il risultato di una richiesta da parte del client in modo da diminuire la latenza.

- ***Layered*** - è possibile inserire dei livelli tra client e server sotto forma di componenti intermediari, i quali trasmettono i messaggi e possono offrire ulteriori servizi. Ogni livello è visibile solo dal suo immediato vicino in modo da essere disaccoppiati tra loro e migliorare la flessibilità in caso di aggiornamenti. Lo svantaggio principale dei sistemi a strati è che aggiungono overhead e latenza in fase di manipolazione dei dati, riducendo così le prestazioni percepite dall'utente.

- ***Uniform interface*** - Questo è l'elemento chiave che contraddistingue lo stile architetturale REST da altre architetture di rete. Ogni risorsa deve essere accessibile attraverso un insieme standard di operazioni in modo da permettere al client di identificarla e interagire con essa. Questo principio è descritto da quattro vincoli: individuazione delle risorse (tramite URI), manipolazione delle risorse attraverso le loro rappresentazioni, messaggi auto descrittivi (ogni messaggio contiene tutte le informazioni necessarie per essere processato) e infine il client deve essere responsabile del mantenimento del proprio stato e può effettuare una transizione solo attraverso hyperlinks.

- ***Code on demand*** - Un client ha la possibilità di estendere le sue funzionalità attraverso il download e l'esecuzione di applet o script. Semplifica i client e aumenta le possibilità di estensione.

Quando si parla di REST non si fa più riferimento all'architettura *SOA - Services Oriented Architecture* dove l'interazione tra client e server è effet-

tuata tramite delle invocazioni a procedure remote ma ci si riferisce a un nuovo paradigma architetturale che pone al centro dell'attenzione non più i "Servizi" ma le "Risorse" dando vita così a un'architettura *ROA - Resource Oriented Architecture*, ovvero un'architettura orientata alle risorse.

L'architettura REST è di tipo client - server in cui i client fanno richieste ai server che a loro volta processano le richieste e restituiscono una risposta. Richieste e risposte contengono delle rappresentazioni di risorse. In particolare, un'applicazione distribuita è **RESTful** se rispetta le sei proprietà fondamentali elencate precedentemente, che caratterizzano il design pattern REST. Dato che REST e uno stile architetturale (non solo per le applicazioni web) e non una tecnologia, i dettagli su come questi vincoli devono essere implementati vengono lasciati allo sviluppatore.

L'idea in questo tipo di servizi e quella di utilizzare HTTP per effettuare le chiamate invece di utilizzare meccanismi complessi come CORBA, RPC o SOAP per la connessione tra sistemi. Come specificato da Roy Fielding nella sua tesi, i servizi RESTful usano le richieste HTTP per inviare i dati (creazione e/o aggiornamento), leggere dati (ad esempio, fare delle query) o eliminare dei dati. In questo modo questi servizi utilizzano i metodi HTTP (GET, PUT, POST, DELETE) per eseguire le quattro operazioni CRUD (Create - Read - Update - Delete), vedi Tabella 1.4.

Metodo HTTP	**Operazione CRUD**	**Descrizione**
POST	Create	Crea una nuova risorsa
GET	Read	Ottiene una risorsa esistente
PUT	Update	Aggiorna una risorsa o ne modifica lo stato
DELETE	Delete	Elimina una risorsa

Tabella 1.4: Match tra metodi HTTP e operazioni CRUD.

Questo stile architetturale, è in contrasto con quella che e la tendenza generale nell'utilizzo dei metodi HTTP. Infatti il metodo GET molto spesso viene utilizzato per eseguire qualsiasi tipo di interazione con il server, ad

esempio per l'inserimento di un nuovo cliente. Questo approccio pero risulta essere in contrasto con i principi REST perche il metodo GET serve per accedere alla rappresentazione di una risorsa e non per crearne una nuova.

1.4.1 Il concetto di risorsa

La risorsa è l'elemento chiave di un'architettura REST che può essere descritta in maniera autonoma per permettere agli utenti ad esempio di creare un collegamento ipertestuale su di essa, recuperare o memorizzare una rappresentazione di essa o eseguire altre operazioni su di essa. Se consideriamo ad esempio un sistema informatico che gestisce una biblioteca e vogliamo costruire un servizio web che permetta di gestire il catalogo della biblioteca in modo da permettere le operazioni di inserimento e aggiornamento dei libri a catalogo ci rendiamo conto da subito che possiamo individuare banalmente come risorsa il "libro".

Per descrivere una risorsa, bisogna individuare tre elementi (vedi Figura 1.10):

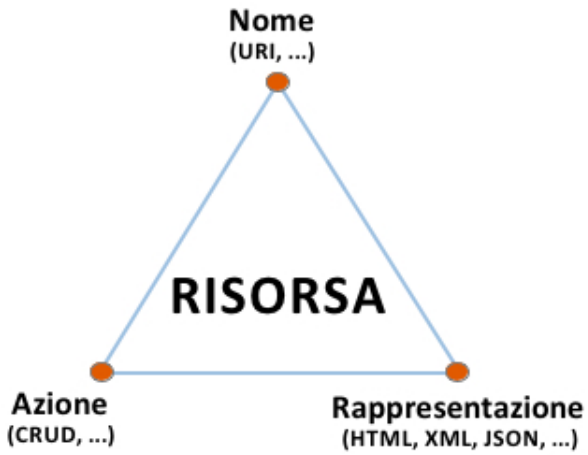

Figura 1.10: Elementi di una risorsa.

Nome

In un'architettura REST, ogni risorsa viene identificata univocamente tramite un "nome" che può essere visto come l'indirizzo della risorsa definito tramite l'URI. Ad esempio se consideriamo l'URI:

```
http://www.bibliotecaonline.it/libri/storia/345
```

stiamo richiedendo al server "www.bibliotecaonline.it" una risorsa identificata con il percorso /libri/storia/345. Una proprietà fondamentale delle URI è che devono essere descrittive e ben strutturate. Si deve capire bene ed in maniera intuitiva leggendo un URI cosa effettivamente stiamo richiedendo.

Azione

Una volta descritta la risorsa in maniera univoca attraverso l'URI, è necessario definire le operazioni che consentono di manipolare le informazioni di tale risorsa. Infatti è possibile interagire con le risorse per mezzo di azioni che vengono messe a disposizione dal protocollo di comunicazione. Il protocollo di comunicazione utilizzato dai servizi REST è l'HTTP mentre i metodi utilizzati sono GET, POST, PUT e DELETE. Queste operazioni hanno una corrispondenza con le quattro operazioni base utilizzate nei sistemi di storage persistenti identificate con l'acronimo CRUD (Create, Read, Update e Delete).

HTTP	CRUD	SQL
POST	Create	INSERT
GET	Read	SELECT
PUT	Update	UPDATE
DELETE	Delete	DELETE

Tabella 1.5: Azioni su una risorsa.

Prendendo in considerazione l'esempio della biblioteca ed eseguendo sullo stesso URI, quindi sulla stessa risorsa i diversi metodi, possiamo analizzare gli effetti ottenuti:

```
http://www.bibliotecaonline.it/libri/storia/
```

Con il metodo GET della URI sopracitata, richiediamo al Web Service di fornirci una lista dei libri di storia. Se sulla stessa URI che identifica la stessa risorsa, effettuiamo una POST, indicando le informazioni necessarie a descrivere una nuova istanza, chiediamo al servizio di creare una nuova risorsa che identifica un nuovo libro.

```
http://www.bibliotecaonline.it/libri/storia/345
```

Se consideriamo l'URI che identifica la singola risorsa "345", attraverso il metodo GET riceveremo le informazioni correlate al singolo libro. Se utilizziamo la PUT con le informazioni necessarie, andremo ad aggiornare i dettagli del libro indicato. Il metodo DELETE invece eliminerà la risorsa "345" (il libro) dal server. I metodi POST e PUT, richiedono altre informazioni per completare la richiesta. Per esempio per il metodo PUT bisogna fornire quali sono i dettagli da sostituire e con quali informazioni.

Rappresentazione

Quando il client invia delle richieste al server, si aspetta una risposta cioè un'insieme di informazioni strutturate in un formato specifico e con una codifica specifica. Una risorsa quindi oltre a un nome e al verbo, ha bisogno anche di un terzo elemento caratteristico, la rappresentazione cioè il modo e la struttura con cui il servizio mostra al destinatario lo stato della risorsa. La rappresentazione, può essere destinata all'interazione umana (ad esempio una pagina HTML) o all'interazione machine-to-machine (ad esempio XML, JSON). REST non stabilisce nessun formato standard per la rappresentazione delle risorse, quindi è possibile utilizzare il formato con cui si è più familiari o comodi.

1.4.2 REST vs SOAP

Come abbiamo visto, i due approcci hanno più o meno lo stesso obiettivo quello di sfruttare il web come piattaforma di l'elaborazione, ma la loro visione e soluzione suggerita è totalmente diversa. Infatti REST propone una visione del Web incentrata sul concetto di "risorsa" mentre i SOAP Web Service mettono in risalto il concetto di "servizio". REST al contrario di SOAP non è uno standard e mai lo sarà ma è uno stile architetturale per la progettazione e lo sviluppo di applicazioni di rete. Un Web Service REST è custode di un insieme di risorse sulle quali un client può effettuare le operazioni previste dal protocollo HTTP. Un Web Service basato su SOAP invece, espone un insieme di metodi richiamabili da remoto da parte di un client. REST può essere definito come un'alternativa leggera ai Web Services SOAP, infatti in maniera molto simile ai Web Service, un servizio REST è:

- Indipendente dalla piattaforma;
- Indipendente dal linguaggio di programmazione;
- Basato su un protocollo di comunicazione Standard (HTTP)
- Non ha bisogno di particolari configurazioni del firewall (Poichè usa la porta 80 default per traffico HTTP).

La chiave della metodologia REST è di scrivere servizi Web utilizzando una interfaccia che è già nota e ampiamente utilizzata: l'URI. La semplicità dei messaggi scambiati in un Web Service REST non ha paragoni se confrontata con i messaggi utilizzati da SOAP.

Come già descritto nel paragrafo 1.2.4,Usando un Web Service e SOAP, i messaggi di richiesta vengono descritti in linguaggio WSDL,

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:body pb="http://www.bibliotecaonline.it/libri">
<pb:GetBookDetails>
<pb:BookID>12345</pb:BookID>
</pb:GetBookDetails>
</soap:Body>
</soap:Envelope>
```

In questo caso, l'intero xml verrà inviato usando una richiesta HTTP POST al server e il risultato sarà probabilmente un file XML, ma sarà incluso, come payload, dentro l'envelope della risposta SOAP. Utilizzando un Web Service REST, la stessa richiesta potrebbe essere la seguente:

```
http://www.bibliotecaonline.it/libri/BookDetails/12345
```

Come si può notare, la richiesta è un semplice URL trasmesso al server usando una semplice richiesta GET. La risposta a questa richiesta è spesso un file XML che non verrà incluso all'interno di nulla, i dati verranno trasmessi in maniera tale da essere subito disponibili per un eventuale utilizzo.

In un servizio SOAP, le richieste vengono indirizzate sempre verso un unico indirizzo chiamato endpoint e all'interno del messaggio vengono definire le operazioni (es: GetBookDetails). Tutte le richieste vengono effettuate sempre con il metodo POST e per questo ci sarà maggior traffico verso l'endpoint in quanto in tutte le richieste viene sempre inviato un documento XML. Nei servizi REST invece le richieste vengono indirizzate verso URI differenti che si mappano sulle risorse e come visto in precedenza, la richiesta è un semplice URL quindi il consumo di banda è ridotto.

Un'altro aspetto interessante è la sicurezza. SOAP utilizza il metodo POST per comunicare con un servizio per questo è necessario controllare sempre il contenuto dei messaggi per verificare se la richiesta può modificare alcune informazioni. Mentre REST che utilizza il metodo GET per effettuare le richieste, può essere considerato più sicuro in quanto per definizione la richiesta GET non può modificare nessun dato. REST non è perfetto, non è la soluzione migliore per ogni servizio web. Per esempio grandi quantità di dati come ad esempio i dati di un'ordine di acquisto, possono superare il limite previsto di una URI (255 caratteri). In questo caso SOAP risulta essere un'alternativa migliore in quanto non ha limiti sulla quantità di dati che possono essere trasmessi perchè utilizza il metodo POST. Effettuare chiamate ad una API HTTP è molto più semplice che fare le chiamate a un API SOAP.

Dal momento in cui le API REST possono essere utilizzate usando semplicemente delle richieste GET, server proxy intermedi, possono memorizzare facilmente nella cache le loro risposte. Le API SOAP invece sono basate su richieste POST e richiedono la creazione di richieste XML che rende difficile la memorizzazione in cache delle risposte. Lo stile di REST è semplice da mettere in pratica e di fatto non serve nessun toolkit costoso di terze parti. Quindi realizzare API basate su HTTP con risposte XML JSON rappresenta attualmente l'alternativa più semplice da implementare lato server ma anche la più facile da utilizzare per un client. Un servizio REST non permette la sostituzione di un servizio creato su SOAP, ma ne è un'ottima alternativa. Entrambi hanno vantaggi e svantaggi in base alla loro applicazione. Per cui l'uso di REST o di SOAP è sicuramente determinato dal contesto in cui viene utilizzato.

1.4.3 ROA vs SOA

SOA - Service Oriented Architecture e *ROA - Resource Oriented Architecture* sono due approcci architetturali per lo sviluppo di un Web Service. In fase di progettazione della piattaforma è opportuno scegliere quale delle due architetture seguire. La scelta non si basa su quale è migliore dell'altro ma è necessario capire bene quale delle due architetture è più appropriata per le funzionalità previste. SOA si basa sul concetto di "servizio" ognuno dei quali è un'unità distinta di funzionalità e non interagisce con altri servizi. Un servizio, è progettato per funzionare in qualsiasi contesto altrimenti le sue possibilita di riutilizzo sarebbero molto basse ed inoltre sarebbe piu probabile che vada in errore quando riceve messaggi non semanticamente corretti. Come per la programmazione ad oggetti in cui è importate il concetto di riutilizzo del codice anche in SOA l'obiettivo è quello di consentire a grandi blocchi di codice di essere riorganizzati e riutilizzati per formare nuove applicazioni. SOA promuove il riuso, l'interoperabilita e la facile evoluzione dei sistemi ed è orientata verso applicazioni *activity-based* (basate su attività) come ad esempio un'applicazione bancaria in cui gli utenti sono interessati a eseguire operazioni quali deposito, prelievo, ecc.

ROA è invece un architettura software specifica per i servizi REST che si basa sulla definizione di risorse. In ROA qualsiasi funzionalità offerta da un servizio deve essere esposta come risorsa. Si ha perciò un numero elevato di risorse ognuna corrispondente ad una operazione utilizzabile dal client. Lo stile architetturale ROA è stato creato perché le architetture SOA che utilizzano i servizi web tradizionali risultano essere molto più complesse. ROA risulta essere quindi uno stile più semplice per esporre e consumare i servizi web senza ricorrere a uno scambio consistente di messaggi XML (WSDL e SOAP) ma semplicemente sfruttando quello che già è reso disponibile attraverso l'HTTP.

Capitolo 2

Autenticazione in Servizi Web

La sicurezza è un requisito rilevante per qualsiasi applicazione distribuita. I servizi Web si basano sullo scambio di messaggi SOAP attraverso il protocollo HTTP il quale non è stato realizzato con l'obiettivo di creare connessioni sicure dal punto di vista della privacy e integrità dei dati e da solo non può offrire garanzie sullo scambio sicuro dei messaggi. I Web Service oltre a fronteggiare le stesse vulnerabilità delle applicazioni Web, quali ad esempio l'SQL injection, si trovano ad affrontare ulteriori minacce e rischi come ad esempio l'inserimento di parti di codice XML nei messaggi scambiati che portano il Web Service al crash. Oppure l'ascolto della comunicazione tra client e server al fine di carpire l'identificatore della sessione che viene usato per portare un attacco al Web Service (attacco man-in-the-middle). A livello di contenuto, i firewall non sono in grado di filtrare messaggi XML maliziosi. Anche *SSL - Secure Sockets Layer* che è il meccanismo maggiormente usato per le transazioni e-commerce, non può essere utilizzato per la sicurezza nei Web Services perchè i Web Services richiedono una sicurezza end-to-end piuttosto che point-to-point che è quella fornita da SSL.

Gli obiettivi della sicurezza sono:

- **Autenticazione** - è il processo di verifica dell'identità di una persona o software di un'entità coinvolta nella comunicazione autorizzandolo ad

usufruire dei relativi servizi associati. Può essere implementato come una semplice combinazione di username e password.

- **Autorizzazione** - una volta determinata l'autenticità di un'entità attraverso l'autenticazione, bisogna verificarne i permessi, vedere quali attività è abilitata a svolgere e a quali risorse può accedere.

- **Integrità dei dati** - è il processo che assicura che i messaggi non vengano intercettati e alterati durante lo scambio fra entità.

- **Privacy** - è importante assicurare che i dati non solo non devono essere alterati, ma devono essere letti solo da chi ha il permesso di farlo.

Molte delle soluzioni sviluppate per garantire la sicurezza nei Web Services riescono a funzionare bene in ambienti semplici e controllati, per garantire una reale protezione end-to-end in questi ambienti dinamici è richiesto un mix di tecnologie e standard attualmente utilizzate.

2.1 Autenticazione base HTTP

Il termine autenticazione indica, nel caso più generale, un metodo mediante il quale si prova l'identità di qualcuno allo scopo di consentirne l'accesso a risorse di qualsiasi genere. Le tecniche di autenticazione sono estremamente differenziate, sia come metodo che come efficacia, in funzione di diversi fattori. In termini semplici si può dire che una tecnica di autenticazione è efficace quando garantisce con ottima probabilità che l'individuo che ha richiesto l'accesso sia effettivamente quello che ne ha il diritto.

HTTP fornisce una tecnica di autenticazione basata sull'invio di username e password per effettuare delle richieste e quindi accedere a determinate risorse. L'autenticazione di base fornita da HTTP (RFCs 2617)[21] è la tecnica più semplice per controllare l'accesso alle risorse sul Web in quanto non richiede cookie, ID di sessione e pagine di login ma sfrutta l'HEADER di HTTP.

Supponiamo di avere sul server una risorsa che utilizza un' autenticazione di base per poter accedere al contenuto. Il client, (quindi il browser) farà una richiesta per poter accedere al contenuto di questa risorsa tramite una chiamata GET. Essendo una semplice richiesta, il client non fornisce un nome utente e una password durante la richiesta per esempio:

```
GET /private/index.html HTTP/1.1
```

Il server risponderà con un errore di tipo HTTP 401 Unauthorized (Autorizzazione negata)

```
HTTP/1.1 401 Authorization Required
WWW-Authenticate: Basic realm="Private Area"
```

Il client presenterà all'utente in una finestra separata (quindi non in un form HTML) la descrizione del computer o del sistema a cui si accede richiedendo un nome utente e una password. In particolare, tramite l'header WWW-Authenticate vengono fornite delle informazioni aggiuntive tra cui il tipo di autenticazione utilizzato che è quello Basic e viene anche segnalato a quale dominio di autenticazione appartiene la risorsa a cui si sta tentando di accedere (realm) in questo caso 'Private Area'. A questo punto l'utente ha la possibilità di proseguire o annullare la connessione. Il client può ricomporre la richiesta in modo corretto includendo i dati di autenticazione che dovranno essere opportunamente concatenati in un'unica stringa separandoli con il carattere ':' (username:password). Una volta che sono stati forniti il nome utente e la password il client aggiunge un header di autenticazione dove le credenziali vengono codificati in Base64 e invia la risposta. L'operazione di codifica in base 64 non compromette la complessità del client in quanto tutti i linguaggi di programmazione hanno nelle loro librerie built-in una funzione di codifica in base 64. Nel caso in cui le credenziali fornite dal client coincidono, il server accetta l'autenticazione e viene restituita la risorsa richiesta.

I principali vantaggi di questa tecnica sono la semplicità ed il largo supporto da parte dei browser web. Lo svantaggio consiste nella trasmissione delle credenziali in chiaro. Infatti l'autenticazione base HTTP non offre al-

cuna protezione per la riservatezza delle credenziali trasmesse. Esse sono solamente codificate in Base64 in modo da garantire la corretta comunicazione di caratteri speciali ma non vengono criptate in nessun modo e quindi possono essere facilmente lette e carpite. Una possibile soluzione potrebbe essere quella di rendere sicura la connessione tra client e server utilizzando HTTPS che è un protocollo che integra l'interazione del protocollo HTTP attraverso un meccanismo di crittografia di tipo *Transport Layer Security* (SSL/TLS).

2.2 Autenticazione Digest

E' la tecnica di autenticazione standard proposta come metodo sostitutivo alla tecnica di autenticazione base proposta dal protocollo HTTP. L'autenticazione Digest viene descritta nella RFC 2617[21] e si propone come soluzione per superare i problemi legati allo scambio in chiaro dello username e della password tra browser web e server HTTP. Digest per cifrare username e password, si basa sul meccanismo di *Message Digest*, una sorta di funzione hash facile da calcolare ma difficile da invertire.

Questo protocollo è di tipo *challenge-response* dove:

- il challenge, inviato dal server al client, contiene un valore detto 'nonce'[1] (number used once) che è ogni volta diverso;
- la response, inviata dal client al server, è il Message Digest (calcolato con l'algoritmo MD5) di:
  - nonce ricevuto dal server;
  - username e password dell'utente;

[1] In crittografia il termine nonce indica un numero, generalmente casuale o pseudo-casuale, utilizzato spesso nei protocolli di autenticazione per assicurare che i dati scambiati nelle vecchie comunicazioni non possano essere riutilizzati in attacchi di tipo replay attack.

In questo modo i dati riservati dell'utente (username e password) non viaggiano mai in chiaro sulla rete.

Anche in questo caso quando il client invia una richiesta di accesso ad una risorsa protetta omettendo le opportune credenziali, il server negherà l'accesso rispondendo con un errore di tipo HTTP 401 Unauthorized (Autorizzazione negata).

```
HTTP/1.1 401 Authorization Required
WWW-Authenticate: Digest realm="Private Area",
qop="auth",
nonce="0cc175b9c0f1b6a831c399e269772661",
opaque="92eb5ffee6ae2fec3ad71c777531578f"
```

Come si può notare, nel messaggio ci sono dei parametri aggiuntivi che nell'autenticazione base non comparivano. In particolare:

- **Qop**: è un parametro che identifica la qualità della protezione "quality of protection" supportata dal server (RFC 2069)[22] in cui il valore "auth" indica l'autenticazione base, mentre il valore "auth-int" indica l'autenticazione con protezione dell'integrità;
- **nonce**: è una chiave univoca generata dal server (solitamente con algoritmo MD5) per prevenire replays attacks;
- **opaque**: è un'altra chiave che viene generata dal server ed è relativa alla specifica risorsa che è stata richiesta dal client;

Il client, sulla base di questi parametri creerà un digest o "un'impronta" che sarà una combinazione tra tutte le chiavi che gli sono state inviate dal server, più altre create in locale concatenate in ordine sparso e criptate più volte utilizzando l'algoritmo MD5. Una volta che il client ha creato questa impronta, la trasmetterà al server per dimostrare di possedere le credenziali di accesso; quindi in questo caso le credenziali username e password non saranno mai inviate in chiaro come succedeva nell'autenticazione base.

Dopodiché partirà la fase di costruzione dell'impronta:

```
ha1 = MD5(user+':'+relam+':'+password);
ha2 = MD5(method+':'+path);
ha3=MD5(ha1+':'+nonce+':'+NC+':'+Lnonce+':'+Qop+':'+ha2);
```

Dove user e password sono le credenziali, method e path sono rispettivamente il tipo di metodo utilizzato per l'accesso alla risorsa e il suo percorso. L'impronta sarà ha3. In questa tecnica, il server non ha idea di quale sia la password dell'utente da validare. In fase di creazione dell'account, il server memorizza la codifica in MD5 della stringa user:realm:password in un file, in modo tale da verificare che il digest (l'impronta) inviata dal client sia esatta, senza però conoscere o memorizzare la password di ogni account.

Una sostanziale differenza tra l'autenticazione base e l'autenticazione con digest sta nel fatto che in questo scenario ogni fase di autenticazione deve essere costituita da due richieste. La prima, in cui il client acquisisce dal server i parametri che gli consentono di creare l'impronta; la seconda in cui viene realizzata la stringa digest. Nella versione basic, le credenziali venivano inviate direttamente all'interno dell'header della prima richiesta.

Quando il server riceve il Message Digest dal client, effettua anch'esso un identico calcolo e confronta i due valori. Se sono uguali il server restituirà la risorsa richiesta.

L'autenticazione Digest è vulnerabile agli attacchi di tipo *man-in-the-middle*. Infatti in una comunicazione client / server il client potrebbe ricevere una falsa richiesta (quindi non dal server con cui crede di comunicare) di autenticazione utilizzando l'autenticazione base HTTP inviando così le proprie credenziali che saranno intercettate e memorizzate da terzi parti. Infatti nell'autenticazione Digest, non esistono meccanismi per i client di verificare l'identità del server.

2.3 WS-Security

WS-Security[23] è un'estensione di SOAP per garantire i quattro aspetti della sicurezza: autenticazione, autorizzazione, integrità dei dati e privacy. Fa parte dalle famiglia WS-* che comprende le specifiche per i servizi web ed è stato pubblicato da *OASIS - Organization for the Advancement of Structured Information Standards*[5]. WS-Security si colloca fra la messaggistica SOAP e le specifiche sulla sicurezza e definisce un modo di autenticazione SOAP all'interno degli headers.

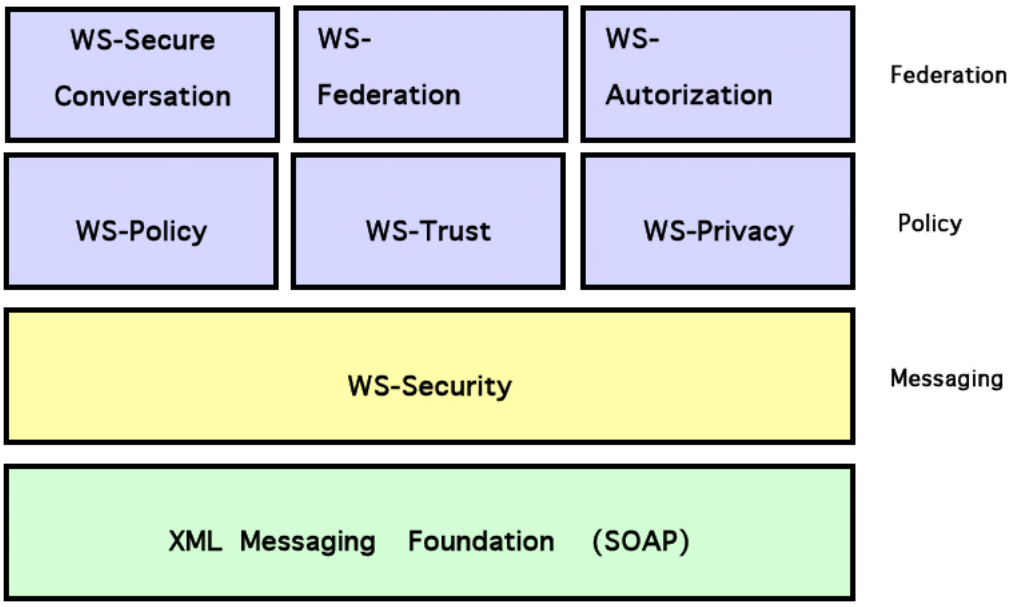

Figura 2.1: Specifiche per la sicurezza nei Web Services

Come è possibile vedere dalla Figura 2.1, il modello si suddivide in tre livelli: messaging, policy e federation.

- ***Messaging*** - Descrive le estensioni al protocollo SOAP per garantire uno scambio di messaggi sicuro. In particolare WS-Security definisce come includere i security token, inclusi i binary security token come i certificati X.509 e i ticket di Kerberos, nei messaggi SOAP e come proteggere i messaggi con le firme digitali e la cifratura.

- ***Policy layer*** - Fornisce un framework per descrivere le metainformazioni dei Web Services. In particolare WS-Privacy definisce un modello su come dichiarare le preferenze in fatto di privacy tra le parti. Mentre WS-Trust definisce un modello per creare e gestire le relazioni fidate fra le parti.

- ***Federation layer*** - Questo livello si occupa di tre specifiche:

  - *WS-SecureConversation*: descrive come gestire e autenticare gli scambi dei messaggi fra le parti;
  - *WS-Federation*: definisce come ottenere un'identità temporanea per accedere a un Web service in un altro security domain.
  - *WS-Authorization*: definisce come gestire l'autorizzazione dei dati e le politiche di autorizzazione.

In WS-Securtiy, all'interno dello Header SOAP vengono trasportati i dati relativi alla protezione. Se l'autenticazione avviene tramite login e password allora viene utilizzato l'elemento **UsernameToken**. Altrimenti se vengono utilizzati i token di autenticazione binari, quali i ticket Keberos e le coppie chiavi pubbliche-private (X.509) si utilizza l'elemento **BinarySecurityToken** all'interno dello Header SOAP. Questi security tokens sono quindi usati per effettuare il binding tra autenticazione e identità. Per rafforzare il meccanismo di sicurezza, è possibile firmare il messaggio. In questo caso, l'intestazione deve contenere informazioni su come è stato firmato il messaggio e su dove sono memorizzate le informazioni relative alla chiave.

Ecco un esempio di autenticazione tramite nome utente e password all'interno dell' header SOAP:

```
  ...
<wsse:Security>
  <wsse:UsernameToken>
    <wsse:Username>giovanni</wsse:Username>
    <wsse:Password Type="wsse:PasswordText">password</wsse:Pass>
  </wsse:UsernameToken>
</wsse:Security>
  ...
```

L'elemento wsse:Security presente nell' header sta ad indicare un contenuto prettamente relativo alla sicurezza.

Per garantire la protezione, la confidenzialità e l'integrità dei messaggi scambiati, vengono utilizzati meccanismi di cifratura e firma digitale realizzati utilizzando altri due standard:

- **XML Signature** [XMLSIG] - che utilizzato insieme ai Security Tokens assicura l'integrità del messaggio rilevando eventuali modifiche al messaggio.
- **XML Encryption** [XMLENC] - che utlizzato insieme ai Security Tokens garantisce la confidenzialità di un messaggio SOAP.

Entrambi vengono inclusi nel messaggio all'interno dell'elemento wsse:Security.

2.3.1 WSSE UsernameToken

Tramite il protocollo HTTP che consente di gestire in maniera personalizzata gli header WWW-Autenticate e Authorization, è possibile sfruttare le specifiche del protocollo WS-Security. In particolare è possibile sfruttare le caratteristiche del WS-Security UsernameToken senza utilizzare SOAP ma effettuando il porting degli header SOAP all'interno degli header HTTP.

Il funzionamento è simile a quello per l'autenticazione Digest. Anche in questo caso quando il client effettua una richiesta per accedere a una risorsa protetta senza indicare le credenziali, il server risponderà in questo modo:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: WSSE realm="PrivateArea", profile="UsernameToken"
```

Il server comunica al client che il tipo di autenticazione è: WSSE e tramite "profile" specifica che la chiave hash deve essere realizzata utilizzando le norme UsernameToken. Cioè il client dovrà generare la chiave hash prendendo in considerazione: un Nonce generato in locale, la password e la data corrente.

I vantaggi principali di questo tipo di autenticazione sono: non viene mandata la password in chiaro come succedeva nell'autenticazione basic e non richiede particolari setup sul server, come invece spesso richiede il Digest. Lo svantaggio è che al contrario dell'autenticazione Digest, nel WSSE per verificare se l'impronta di autenticazione è giusta il server memorizza le password in chiaro e quindi in caso di attacco informatico si avranno a disposizione tutte le password degli utenti.

2.4 OAuth

OAuth - Open Authentication[24][25], è un protocollo di autenticazione attraverso il quale un utente può delegare applicazioni (mobile e desktop) o siti web ad accedere ai dati custoditi per suo conto presso un certo servizio web, senza però dover fornire loro le proprie credenziali di accesso a tale servizio.

Infatti attualmente sempre più siti web offrono dei servizi collegandosi ad altri siti internet. Basti pensare a Facebook che usa i contatti di Gmail per cercare gli amici, oppure un sito di stampa foto che stampa le foto caricate su Flickr.

In pratica, si tratta di un protocollo che implementa una metodologia di autenticazione basata su *API - Application programming interface* attraverso il quale un determinato servizio web può garantire, per conto di un Utente, un sorta di "accesso speciale" a differenti Applicazioni/Siti Web, permettendo loro di accedere ai dati di quell'Utente in maniera sicura e protetta senza che l'utente condivida la sua password con servizi o Applicazioni diversi da quelli dei Servizi Web che custodiscono i suoi dati privati.

Nel modello tradizionale di autenticazione client-server, il client utilizza le proprie credenziali per accedere alle risorse ospitate dal server. OAuth introduce un terzo ruolo in questo modello: il "proprietario della risorsa". Quindi nel modello OAuth il client (che non è il proprietario della risorsa) richiede l'accesso alla risorsa controllata dal "proprietario della risorsa" ma memorizzata sul server. Affinchè il client possa accedere alla risorsa, deve prima ottenere il permesso dal proprietario della risorsa. Questo permesso è espresso sotto forma di token che ha una durata limitata e permette quindi al client di non condividere le sue credenziali.

2.4.1 Terminologia

Prima di proseguire con la spiegazione del funzionamento di OAuth è importante conoscere i termini fondamentali utilizzati dal protocollo e capire il loro ruolo nella transazione.

- ***Service Provider*** - è il sito o il servizio web dove sono memorizzate le risorse protette dell'utente(*Protected Resources*).
- ***User*** - è il proprietario delle risorse che vuole condividere con un'altro sito o applicazione senza renderle pubbliche.
- ***Consumer*** - può essere un sito web, un programma, un dispositivo mobile, o qualsiasi altra cosa connessa al web che cerca di accedere alle risorse dell' user.
- ***Protected Resources*** - sono gli oggetti che OAuth tutela (risorse protette), sui quali concede accesso e permessi. Possono essere dati (foto, documenti, contatti), attività (postare un oggetto in un blog, trasferire fondi) o qualsiasi risorsa che necessita una restrizione degli accessi.
- ***Tokens*** - sono stringhe alfanumeriche casuali uniche utilizzate in alternativa alle credenziali degli utenti.

2.4.2 Workflow del protocollo

Il flusso di autenticazione del protocolo OAuth, prevede che per accedere alle risorse protette attraverso un API è necessario acquisire un token di accesso *Access Token*. Prima di ottenere questo token sono necessari diversi passaggi. In Figura 2.2 è possibile visualizzare il flusso di autenticazione OAuth tra un Consumer e un Service Provider. Nella fase A, il Consumer richiede il *Request Token* che non è ancora specifico dell'utente ma server per stabilire l'inizio della sessione di OAuth. L'autorizzazione di questo *Request Token* è uno dei punti vitali nel flusso di lavoro OAuth. Il Consumer quindi dopo aver ricevuto risposta (fase B) dal Service Provider alla sua richiesta di Request Token, reindirizza l'utente (fase C) al Service Provider utilizzando il Request Token appena ricevuto. In questo modo l'utente potrà effettuare il login e accedere in modo sicuro alle risorse protette senza cedere le sue credeziali a terze parti. Quando l'utente ha effettuato il login, il Service Provider ridireziona l'User al Consumer e restituisce al Consumer un *Request Token* autorizzato (fase D). In questo modo, attraverso il Request Token autorizzato, nella fase E il Consumer può richiedere l'Access Token che verrà utilizzato per accedere alle risorse attraverso una API OAuth protetta. Dopo che il Service Provider ha rilasciato l'Access Token, il Consumer può accedere alle risorse protette.

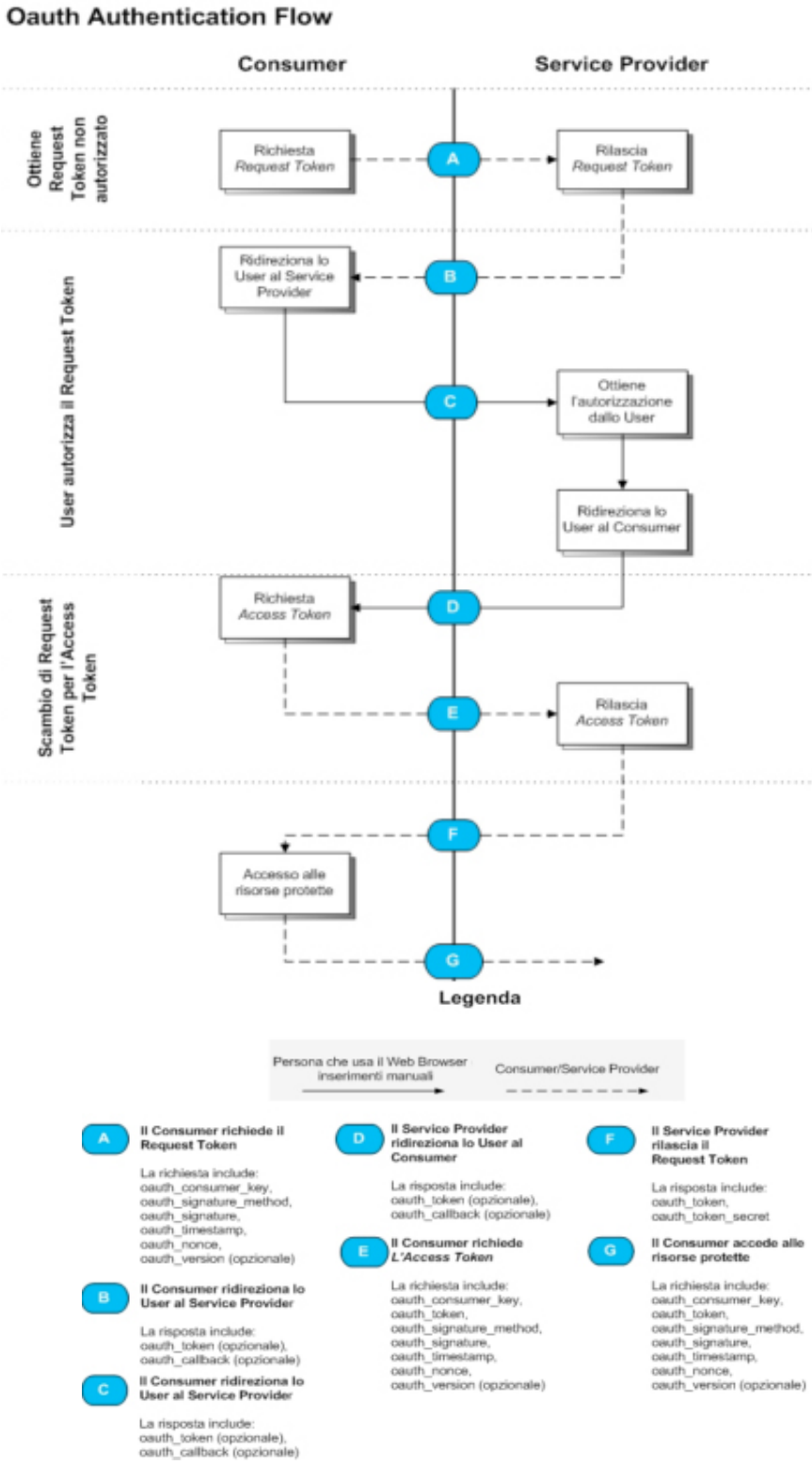

Figura 2.2: Flusso di autenticazione OAuth.

Per comprendere meglio il flusso di lavoro del protocollo è possibile fare riferimento alla Figura 2.3 dove viene riportato il diagramma di sequenza del flusso OAuth in cui viene descritto lo scenario attraverso una sequenza di azioni in cui si evidenzia la sequenza nel tempo dei messaggi scambiati.

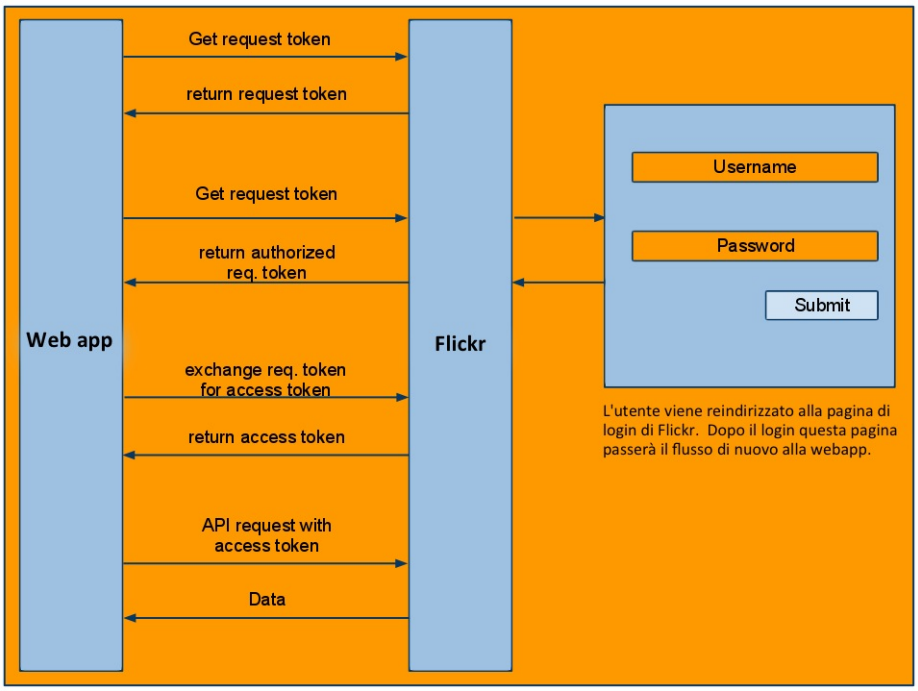

Figura 2.3: Diagramma di sequenza del flusso OAuth.

Nel dettaglio, prima che il Consumer chieda il permesso all'User di accedere alle sue risorse, si devono stabilire una serie di credenziali temporanee con il Service Provider per identificare la richiesta.

Per fare questo, il Consumer invierà al Service Provider una richiesta HTTP:

```
POST /initiate HTTP/1.1
Host: flickr.com
Authorization: OAuth realm="Photos",
        oauth_consumer_key="dpf43f3p2l4k3l03",
        oauth_signature_method="HMAC-SHA1",
        oauth_timestamp="137131200",
        oauth_nonce="wIjqoS",
        oauth_callback="http%3A%2F%2Fstampafoto.it%2Fready",
        oauth_signature="74KNZJeDHnMBp0EMJ9ZHt%2FXKycU%3D"
```

Questa richiesta iclude un oauth_consumer_key che rappresenta la API key assegnata al Consumer per poter utilizzare i servizi offerti dal Service Provider, un oauth_signature_method che indica il metodo di criptaggio utilizzato dal Consumer per firmare la richiesta. Dato che OAuth non impone un particolare metodo di firma è possibile utilizzare ad esempio HMAC-SHA1, RSA-SHA1, o PLAINTEXT. Il parametro oauth_timestamp indica il tempo espresso in secondi dal 1 Gennaio, 1970 00:00:00 GMT (epoch). Il valore del timestamp deve essere un numero intero positivo e deve essere uguale o superiore al timestamp utilizzato nelle precedenti richieste. Opzionalmente il Consumer può specificare il parametro oauth_callback in cui viene specificato l'URI che il Service Provider userà per reindirizzare l'User verso il Consumer un volta che l'autorizzazione dell'utente è completata. Infine oauth_signature è la firma generata tramite il metodo specificato in oauth_signature_method.

Il Service Provider convaliderà la richiesta e risponderà con una serie di credenziali temporanee Request Token nel body della risposta HTTP.

```
HTTP/1.1 200 OK
Content-Type: application/x-www-form-urlencoded
oauth_token=hh5s93j4hdidpola&oauth_token_secret=hdhd0244k9j7ao03&
oauth_callback_confirmed=true
```

Il Consumer, rendirizzerà l'User all'endpoint di autorizzazione per ottenere l'approvazione ad accedere alle sue risorse private sul Service Provider utilizzando il Request Token ricevuto in precedenza.

```
https://flickr.com/authorize?oauth_token=hh5s93j4hdidpola
```

Il Service Provider, chiederà all'User di accedere utilizzando il suo nome utente e la password e in caso di successo, chiederà di approvare l'accesso alle sue risorse private da parte del Consumer. Una volta che L'User ha approvato la richiesta, verrà reindirizzato alla callback URI fornita dal Consumer nella precedente richiesta.

```
http://stampafoto.it/ready?oauth_token=hh5s93j4hdidpola&oauth_verifier=hfdp7dh39dks9884
```

Questa callback, informa il Consumer che l'User ha completato il processo di autorizzazione e quindi è possibile procedere con lo scambio dell'*Access Token*.

```
POST /token HTTP/1.1
Host: flickr.com
Authorization: OAuth realm="Photos",
        oauth_consumer_key="dpf43f3p2l4k3l03",
        oauth_token="hh5s93j4hdidpola",
        oauth_signature_method="HMAC-SHA1",
        oauth_timestamp="137131201",
        oauth_nonce="walatlh",
        oauth_verifier="hfdp7dh39dks9884",
        oauth_signature="gKgrFCywp7rOOOXSjdot%2FIHF7IU%3D"
```

Il server convalida la richiesta e risponde con una serie di *Access Token* nel body della risposta HTTP:

```
HTTP/1.1 200 OK
Content-Type: application/x-www-form-urlencoded
oauth_token=nnch734d00sl2jdk&oauth_token_secret=pfkkdhi9sl3r4s00
```

In questo modo il Consumer è pronto per richiedere le risorse private.

```
GET /photos?file=erasmus.jpg&size=original HTTP/1.1
Host: flickr.com
Authorization: OAuth realm="Photos",
        oauth_consumer_key="dpf43f3p2l4k3l03",
        oauth_token="nnch734d00sl2jdk",
        oauth_signature_method="HMAC-SHA1",
        oauth_timestamp="137131202",
        oauth_nonce="chapoH",
        oauth_signature="MdpQcU8iPSUjWoN%2FUDMsK2sui9I%3D"
```

In questo caso il Service Provider convalida la richiesta e risponde con la risorsa richiesta. Il Consumer sarà in grado di continuare ad accedere alle risorse private dell'User utilizzando lo stesso set di *Access Token* per la durata dell'autorizzazione concessa dall'User, o fino a quando l'User revoca l'accesso.

Attraverso un esempio reale è possibile capire meglio il concetto di OAuth:

Supponiamo che Marco sia appena tornato dal suo erasmus a Istanbul e vuole condividere alcune foto con i suoi amici tramite Flickr. Quindi effettua il login su Flickr e carica le foto segnandole come private.

In questo modo Marco è lo User, Flickr il Service Provider e le foto caricate sono le Risorse Protette.

Dopo aver condiviso le sue foto con alcuni dei suoi amici, Marco decide di stamparne qualcuna. A questo punto Marco accede a StampaFoto.it, un servizio di stampe online che supporta l'importazione di foto da molti siti,

incluso Flickr. Marco quindi sceglie di importare le foto da Flickr e clicca su continua.

Dato che Marco ha contrassegnano le sue foto come private, StampaFoto.it che è il Consumer deve utilizzare OAuth per avere accesso alle foto di Marco su Flickr. Dopo aver cliccato il tasto Continua, StampaFoto.it richiede a Flickr un *RequestToken* non nominativo, che verrà utilizzato da StampaFoto.it per ottenere l'approvazione da Marco per accedere alle sue foto.

Appena StampaFoto.it riceve il RequestToken, a Marco viene mostrata la pagina di autorizzazione OAuth di Flickr in modo tale che Marco confermi il permesso di accedere alle sue foto. Una volta confermato, Marco viene reindirizzato su Flickr dove gli viene chiesto di accedere. Marco si assicura di essere realmente su Flickr controllando l'URL e a questo punto effettua il login. Dopo essersi loggato, Marco seleziona le foto che vuole stampare e conferma. Appena la richiesta viene confermata, Flickr contrassegnara *il RequestToken come nominativo e autorizzato da Marco che viene reindirizzato nuovamente su StampaFoto.it rimasto in attesa delle autorizzazioni per accedere alle foto.*

StampaFoto.it utilizza il *RequestToken* ricevuto per scambiarlo con un *AccessToken*. I *RequestToken* servono solamente per ottenere il permesso dell'utente, mentre l' *AccessToken* può permettere a StampaFoto.it (Consumer) di accedere alle risorse protette di Marco (in questo caso le foto). Alla fine dell'operazione, il browser di Marco si aggiorna visualizzando le foto selezionate per permettere di completare l'ordine.

Con OAuth quindi Marco può far accedere alle sue foto (ad esempio su Flickr) un altro sito come StampaFoto.it senza dover MAI rivelare a quest'ultimo le sue credenziali username e password di Flickr.

2.4.3 Architettura di sicurezza

Come abbiamo visto, HTTP definisce uno schema di autenticazione chiamato "base" che è comunemente usato da molti web services e siti per consumare le proprie API. Quando non è utilizzato su HTTPS, l'autenticazione base soffre di rischi significativi per la sicurezza. Il protocollo OAuth, non obbliga l'utente a fornire le credenziali di accesso al provider (username e password) ma utilizza un sistema di firme digitali. Questo sistema funziona tramite un algoritmo matematico che trasforma la firma in una stringa alfanumerica ad esempio utilizzando una funzione hash SHA-1. In questo modo il Service Provider può controllare che i dati inviati non vengano modificati durante il percorso ma non può accertarsi dell'identità di chi ha effettuato la richiesta. Per fare ciò è necessario inviare un elemento di riconoscimento noto solo alle due parti detto *Sheared Secret*, che può essere semplicemente allegato alla richiesta (se il canale di trasmissione è sicuro), oppure integrato nell' algoritmo matematico (in questo caso HMAC-SHA-1) cosicché siano necessari entrambi per poter tradurre la stringa ricevuta.

Il vantaggio di questo approccio rispetto all'autenticazione base HTTP è che il segreto non viene mai inviato con la richiesta. Il segreto è usato per firmare la richiesta ma non è parte di essa, né può essere estratto.

Firma digitale e Shared Secret combinati, forniscono un certo livello di sicurezza garantendo l'integrità dei dati e assicurando che chi ha completato la richiesta sia un utente autorizzato. Essi però non controllano se la richiesta è stata intercettata tramite sniffing del network da una terza parte non autorizzata la quale potrebbe riutilizzare la richiesta a proprio piacimento. (Replay Attack).

Per prevenire questo tipo di attacchi, è stato necessario aggiungere nella firma un codice univoco detto *Nonce* e un *Timestamp* (data e ora). Il termine nonce che sta per "number used once", rappresenta una stringa casuale sempre diversa utilizzata per identificare univocamente ogni richiesta effettuata. Essendo integrato nella firma non può essere letto senza conoscere lo Sheared Secret, garantendo così la sicurezza che la stessa richiesta non ven-

ga utilizzata più volte. Per controllare ciò il Service Provider dovrà salvare tutti i nonces ricevuti facendo insorgere il problema del costo di dover tenere archiviati tutti questi dati. OAuth aggiunge un valore di Timestamp per ogni richiesta, che consente al server di conservare i nonce solo per un tempo limitato, eliminando i nonces che superano il suo range di datazione (implementato dal Service Provider) e di rifiutare le richieste con un Timestamp non valido (probabili Replay Attack).

Solamente la combinazione di Sheared Secret, nonce e Timestamp integrati nella firma può garantire una protezione efficace contro futuri attacchi.

OAuth utilizza tre metodi per firmare le richieste: PLAINTEXT, HMAC-SHA1 e RSA-SHA1. PLAINTEXT è destinato a lavorare con HTTPS e come per l'autenticazione base, trasmette le credenziali in chiaro. Gli altri due metodi utilizzano l'algoritmo di firma RSA HMAC, combinato con la funzione hash SHA-1. Quando le richieste vengono firmate, è necessario specificare quale metodo è stato utilizzato per consentire al destinatario di riprodurre la firma per la verifica. La decisione su quale metodo utilizzare dipende dalle esigenze di sicurezza di ogni applicazione.

Capitolo 3

Analisi dello scenario di riferimento

Per poter spiegare e capire al meglio tutte le informazioni riportate in questo capitolo, è opportuno descrivere l'ambito di sviluppo di questo lavoro di tesi. In particolare verranno descritti due concetti fondamentali: *DD - Document Delivery* (fornitura di articoli su richiesta) e *ILL - Inter Library Loan* (prestito interbibliotecario). Con il Document Delivery, i documenti (articoli scientifici) vengono richiesti a un'altra biblioteca e forniti in copia riprodotta, poi consegnata all'utente, mentre il prestito interbibliotecario, è un prestito vero e proprio di libri tra biblioteche, che permette al lettore di poter consultare libri non posseduti dalla sua biblioteca. Quindi è possibile affermare che il *Document Delivery* può essere considerato come un caso particolare dell'*Inter Library Loan.*

DD e *ILL* sono basati sul reciproco scambio di documenti fra le biblioteche e si configurano come servizi bidirezionali; la stessa biblioteca può infatti essere:

- **Fornitrice** - nel caso in cui fornisca su richiesta il materiale a un'altra biblioteca;
- **Richiedente** - nel caso in cui sia essa stessa a richiedere e ricevere il materiale.

La legge italiana sul diritto d'autore prevede eccezioni e limitazioni che rendono possibile il prestito e il Document Delivery eseguiti dalle biblioteche. I servizi sono disciplinati dal regolamento della biblioteca e vengono attivati per soddisfare gli interessi di studio e di ricerca dei propri utenti.

Il servizio di Document Delivery svolge un ruolo importante nella ricerca pubblica italiana e nelle biblioteche universitarie. I principali cataloghi in Italia sono: **ACNP - Archivio Collettivo Nazionale dei Periodici**[26] che include circa 2.000 biblioteche attive, principalmente provenienti da università e centri di ricerca e **SBN - Sistema Bibliotecario Nazionale**[27] in cui partecipano circa 4900 biblioteche statali, enti locali, universitarie, istituzioni pubbliche e private, operanti in diversi settori disciplinari. ACNP contiene le descrizioni bibliografiche (dati catalografici) dei periodici posseduti da biblioteche dislocate su tutto il territorio nazionale e copre tutti i settori disciplinari, essendo un catalogo specializzato, viene considerato la risorsa italiana più autorevole per il Document Delivery. Mentre SBN comprende diverse tipologie di documenti: materiale antico (monografie a stampa dal XV secolo fino al 1830), materiale moderno (monografie, registrazioni audio e video, archivi elettronici, periodici e altri materiali a partire dal 1831), musica manoscritta, musica a stampa e libretti, materiale grafico e cartografico.

3.0.4 Applicazioni per ILL

La maggiore visibilità del patrimonio documentale delle biblioteche, grazie alla diffusione dei cataloghi nazionali (ACNP, SBN), ha permesso un notevole incremento delle transazioni di Document Delivery e prestiti interbibliotecari, pertanto si è resa necessaria una riorganizzazione del suddetto servizio sia in termini di costi, sia in termini di rilevazione statistica dei dati. Negli ultimi anni è aumentato quindi l'interesse tra le biblioteche italiane e non, di avviare una rete di cooperazione per la condivisione di risorse in particolare per lo scambio di documenti scientifici. Per questo sono stati sviluppati applicativi software per il Document Delivery e l'Inter Library Loan per permette alle biblioteche di richiedere e di fornire documenti in maniera reciproca, mediante ad esempio applicativi web.

ILLiad

ILLiad - InterLibrary Loan Internet Accessible Database[28], è uno strumento per il Document Delivery e l'Inter Library Loan, che consente al personale di una biblioteca di gestire i borrowing (prestiti) e i lending (consegne) di documenti attraverso una interfaccia Windows-based. Questo sistema sfrutta una rete di biblioteche, centri di ricerca e altre istituzioni in tutto il mondo per soddisfare le richieste di materiale come libri, capitoli di libri e articoli scientifici. ILLiad si integra perfettamente con i principali servizi *OCLC - Online Computer Library Center*[1] come ad esempio WorldCat che è il catalogo collettivo del più grande sistema bibliotecario mondiale che registra le collezioni delle 72.000 biblioteche che, da oltre 170 nazioni partecipano alla cooperazione bibliotecaria *OCLC*.

ILLiad permette di ricevere gli articoli richiesti tramite Ariel. Tramite Ariel[29] infatti è possibile inviare e ricevere attraverso Internet un file di immagine (contenente un documento acquisito con scanner più una descri-

[1]OCLC è un'organizzazione mondiale a servizio delle biblioteche il cui scopo è far accedere tutti al mondo della cultura. Più di 60.000 biblioteche nel mondo utilizzano i servizi dell'OCLC alla fine di trovare, di catalogare o di conservare i loro libri.

zione bibliografica) da un computer all'altro. Il vantaggio è che permette di eseguire con un unico prodotto le stesse operazioni che, con la sola posta elettronica, si devono svolgere in più passaggi e, soprattutto, per mezzo di più programmi. Infatti una volta importato il documento in formato immagine (.TIFF) nel software ILLiad, viene converito in PDF e inviato al server web. L'utente che ha fatto la richiesta viene avvertito tramite email che il suo articolo richiesto è disponibile per cui potrà visionarlo e stamparlo. Nel 2003 venne introdotto un nuovo protocollo per la consegna elettronica degli articoli chiamato Odyssey al quale venne aggiunta l'opzione Trusted Sender per indicare l'attendibilità dell'articolo ricevuto. In questo modo solo gli articoli ricevuti da una biblioteca designata come "attendibile" saranno inviati a un server web senza l'intervento del personale.

Questo software è stato sviluppato presso la Virginia Polytechnic Institute and State University con lo scopo di automatizzare le funzioni di prestito interbibliotecario, dando la possibilità agli utenti di avviare e monitorare le loro richieste ILL attraverso una semplice interfaccia web. ILLiad mantiene delle statistiche in tempo reale per tenere traccia del flusso di lavoro e quantificare le prestazioni. Inoltre l'interfaccia web ottimizzata per la visualizzazione su dispositivi mobili consente agli utenti di accedere ai loro account tramite smartphone o tablet.

Relais

Relais[30] permette alle biblioteche di automatizzare completamente i loro processi di prestito interbibliotecario e di Document Delivery (ILL/DD). E' stato implementato presso la Biblioteca Nazionale di Medicina degli Stati Uniti ed offre un modello innovativo per la condivisione delle risorse tra le biblioteche. Il sistema permette alle biblioteche di passare da un processo di prestito interbibliotecario basato sulle fotocopie a un processo basato sulle immagini digitali. Attraverso l'automazione del processo di prestito l'operatore addetto alla digitaizzazione del documento può concentrarsi esclusi-

vamente sulla qualità del documento in quanto è il software che pensa alla gestione dei dati e delle informazioni relative al prestito.

Relais supporta una vasta gamma di metodi di consegna compresa la stampa locale del documento, consegna tramite posta tradizionale, tramite Fax, Ariel e consegna tramite e-mail. Relais riesce a comunicare con sistemi esterni compresi OCLC, DOCLINE e sistemi che utilizzano il protocollo ISO ILL[31]. In particolare, Relais è composto da una gamma di prodotti come supporto alla fornitura di soluzioni sofisticate per automatizzare e snellire le operazioni tradizionali di ILL e Document Delivery.

- **Relais D2D - Discovery to Delivery** - D2D consente di ottimizzare la capacità dell'utente di scoprire gli articoli di suo interesse e ottenere accesso immediato in modo tale che la richiesta e la consegna degli articoli desiderati avvengono senza alcun intervento del personale. Con Relais D2D, gli utenti saranno in grado di effettuare ricerche specifiche utilizzando un unico sito di riferimento.

- **Relais ILL - end to end Request Management** - Permette di inviare e ricevere richieste da e verso altri applicativi ILL compreso *WCRS - WorldCat Resource Sharing*. Utilizza OpenURL per semplificare il modo in cui gli utenti inviano richieste e verifica automaticamente i cataloghi utilizzando il protocollo Z39.50.

- **Relais Express / Express PLUS** - Permette di convertire i documenti cartacei in formato elettronico, o utilizzare i documenti elettronici esistenti, e consegnarli ai propri utenti o ad altre biblioteche, riducendo il tempo, la complessita del lavoro e gli errori.

RapidILL

RapidILL[32] è un sistema di condivisione delle risorse che è stato progettato dal personale che si occupa del prestito interbibliotecario presso la Colorado State University Libraries. A seguito di una devastante alluvione nel 1997, RapidILL fu sviluppato per fornire un sistema che permettesse in

modo veloce e a basso costo la richiesta e la consegna di documenti e articoli tramite prestito interbibliotecario. E' compatibile con qualsiasi sistema di ILL tra cui Clio, ILLiad, e Relais e utilizza i dati da qualsiasi OPAC[2]. Inoltre, RapidILL fornisce la trasmissione dei documenti via ILLiad, Odissey e RapidX. RapidX è stato sviluppato dal team Rapid ed elimina l'obbligo per le biblioteche di utilizzare Ariel o Odyssey per la consegna elettronica o la ricezione degli articoli.

RapidILL conta circa 300 biblioteche con oltre 50 milioni di volumi e utilizza un unico database che conta oltre 3 milioni di articoli a libero accesso. In questo modo, quando viene trovata una corrispondenza per un articolo richiesto, viene subito inviato automaticamente alla biblioteca richiedente senza stampare nessuna richiesta, senza nessuna scansione e quindi senza intervento del personale. Le biblioteche vengono raggruppate in gruppi denominati "pod". In questo modo una biblioteca può entrare a far parte di un gruppo e può partecipare a tutti i "pod" per le quali soddisfa i requisiti di adesione. Non è previsto alcun costo aggiuntivo per la partecipazione a più pod. Partecipare a più pod è un modo semplice e gratuito per espandere la rete di condivisione delle risorse in RapidILL.

Una delle caratteristiche che rende RapidILL un software a basso costo è che non richiede praticamente alcun supporto tecnico. La maggior parte dei risparmi sui costi di RapidILL sono attribuiti al risparmio di tempo del personale. Ad esempio, RapidILL automatizza molte operazioni di prestito, come la selezione e la verifica degli istituti che ne fanno parte.

GTBib

GTBib[33] è il più grande gestore di documenti che fornisce un servizio di prestito interbibliotecario tra le biblioteche universitarie, la biblioteca nazionale e molte biblioteche ospedaliere in Spagna. Esso comprende tutti gli aspetti relativi alla gestione delle richieste, e utilizza il Web come strumento

[2]On-line Public Access Catalogue ovvero Catalogo in rete ad accesso pubblico ed è il catalogo informatizzato delle biblioteche.

di base per la comunicazione tra utenti e fornitori di servizi. In particolare GTBib è basato su un'architettura client-server, è scritto in PHP, utilizza MySQL come database manager e i dati scambiati sono strutturati utilizzando il linguaggio XML e trasmessi tra client e server attraverso il protocollo SOAP. L'intera gestione dei servizi (Lending e Borrowing) vengono eseguiti da un browser web. Il sistema è distribuito, ogni biblioteca ha un suo applicativo client e invia i suoi record bibliografici ai cataloghi regionali e in particolare al catalogo REBIUN[3].

Inizialmente la gestione delle richieste veniva gestita attraverso le email che però risultò essere un metodo poco efficiente in quanto molte volte le email venivano contrassegnate come spam e gli allegati non potevano superare una certa dimensione. Per questo venne introdotto SOAP per la gestione del servizio di Document Delivery.

Nel 2004 venne implementato nel software lo standard NCIP Z39.83 che definisce un protocollo di comunicazione per l'interoperabilità tra sistemi bibliotecari integrati (vedi capitolo 3.0.5) per esempio per il borrowing degli articoli richiesti da un'altra biblioteca. Inoltre GTBib utilizza il protocollo Z39.50 per verificare i posseduti delle biblioteche nei cataloghi spagnoli come il C17, REBIUN, CCUC ed è inoltre compatibile con i principali servizi di document delivery spagnoli e internazionali (British Library-Artemail NLM).

ILL SBN

ILL SBN[34] è un servizio nazionale di prestito interbibliotecario e fornitura documenti accessibile gratuitamente a tutti su Internet, rivolto ai bibliotecari e agli utenti finali. Il servizio gestisce il prestito interbibliotecario, tramite l'intermediazione di una biblioteca richiedente, la fornitura di un documento in fotocopia o in formato elettronico, la richiesta di preventivi di spesa e le statistiche sui servizi effettuati. La partecipazione è aperta a

[3]REBIUN è il catalogo collettivo delle università spagnole e comprende il patrimonio bibliografico di 74 atenei spagnoli sia pubblici sia privati, oltre che quello della Biblioteca nacional de España e di alcune importanti biblioteche specializzate e di ricerca spagnole.

tutte le biblioteche, anche non SBN. ILL SBN è un sistema aperto in grado di integrarsi con i cataloghi italiani ed esteri e di interoperare, attraverso un protocollo standard, con i sistemi di gestione di prestito locale e con altri servizi nazionali di prestito interbibliotecario. Al servizio si accede, dopo aver localizzato una pubblicazione dal catalogo **SBN - Servizio Bibliotecario Nazionale** oppure dal catalogo **ACNP - Archivio Collettivo Nazionale dei Periodici** o anche dal catalogo di spoglio di periodici **ESSPER - Associazione periodici italiani di economia, scienze sociali e storia**.

I bibliotecari attraverso un'interfaccia web, inviano le richieste per conto dei loro utenti e gestiscono le transazioni con le biblioteche partner attraverso messaggi definiti dal sistema in accordo allo standard ISO/ILL. Gli utenti finali, in possesso di un indirizzo di posta elettronica e iscritti a una biblioteca partner possono inviare direttamente una richiesta di prestito interbibliotecario e fornitura documenti a un'altra biblioteca iscritta al servizio. In questo caso non è necessaria una password, ma la richiesta deve essere validata dalla biblioteca di riferimento. I bibliotecari e gli utenti possono seguire in modo trasparente l'iter delle richieste fino al ricevimento dei documenti.

Le funzioni previste da ILL SBN sono quelle definite dallo standard ISO ILL[31] per il prestito interbibliotecario e contenute nel documento Service definition (ISO 10160).

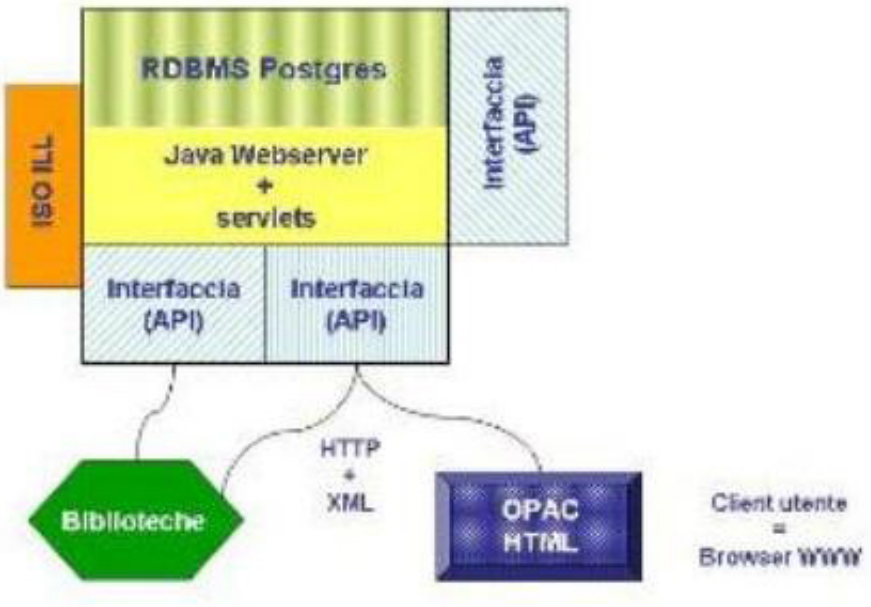

Figura 3.1: Architettura e tecnologia ILL SBN.

Come definito in Figura 3.1, l'architettura di ILL SBN presenta un server centrale, realizzato con tecnologie Java, su cui sono implementate le funzioni di prestito interbibliotecario attraverso l'utilizzo di API e una interfaccia utente gestita tramite browser web.

3.0.5 Protocolli di comunicazione e Standard di riferimento

Tutti i software appena descritti hanno bisogno di protocolli e standard di comunicazione per interagire tra di loro e con i vari cataloghi (es. ACNP, SBN) in modo da poter scambiarsi le informazioni necessarie per le operazioni di Document Delivery. Per questo sono stati sviluppati protocolli e standard di comunicazione specifici.

Z39.50

Lo Z39.50[35] è un protocollo di comunicazione client-server per ricercare attraverso una rete informatica informazioni in un database (es. un catalogo bibliografico). Supporta una serie di azioni tra cui la ricerca e recupero di informazioni, e la sua sintassi permette di effettuare query molto complesse. Lo Z39.50 permette di interrogare cataloghi di tipo diverso o basati su software differenti. Attraverso questo protocollo, un utente remoto può ricercare e recuperare in modo trasparente le informazioni disponibili su un altro sistema informatico. Inoltre i record di catalogazione provenienti da una interrogazione Z39.50 sono spesso in formato *MARC - MAchine Readable Cataloguing*[4] e possono essere uniti a record MARC di altri cataloghi per formare bibliografie, per catalogazione derivata, esempio utilizzando interfacce che consentono di consultare più OPAC simultaneamente.

Standard di riferimento sono la norma ISO 23950 e, per gli Stati Uniti, la norma ANSI/ISO Z39.50. La sua evoluzione è coordinata dalla Biblioteca del Congresso degli Stati Uniti.

OpenURL

OpenURL[36] è uno standard che fornisce la sintassi per il trasporto fra i diversi servizi informativi dei metadati bibliografici. I metadati sono 'dati sui dati', cioé informazioni, di solito strutturate in campi, che permettono una piu' efficiente organizzazione, gestione e recupero delle informazioni in rete. OpenURL consiste nello scambio di metadati relativi a una risorsa (solitamente un libro, un periodico, un articolo) tra un utente che attiva un collegamento e un sistema informativo ovvero information provider, che rappresenta la fonte dei metadati. Di solito si riferisce a una banca dati, ma può anche essere un servizio di full text, un archivio di e-print, di pre-print o un open-archive, un database di tesi, oppure un OPAC (catalogo in linea).

[4] E' una specifica per la rappresentazione dell'informazione bibliografica, confluita nello standard ISO 2709.

OpenURL può essere definito anche un protocollo generale che permette a una risorsa informativa, (come la banca dati Web of Science), di passare i metadati inerenti una citazione (come ad esempio l'ISSN, il titolo, l'autore, il volume, il fascicolo, il numero di pagina, etc..) al server configurato per la ricezione di OpenURL (OpenURL Resolver, vedi Figura 3.2) di una istituzione il quale crea dinamicamente i collegamenti ai servizi legati a quella particolare risorsa, per esempio:

- accesso libero ai dati bibliografici e all'abstract dell'articolo;
- accesso libero al full text dell'articolo dal sito della rivista, cui la biblioteca è abbonata;
- indicazione della collocazione della versione cartacea dell'articolo all'interno della biblioteca;
- attivazione dell'Interlibrary Loan e Document Delivery;

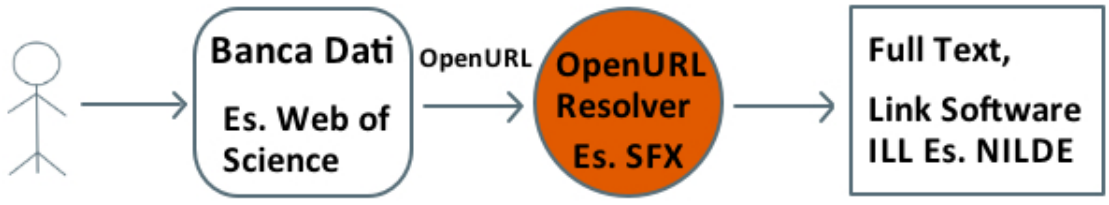

Figura 3.2: Esempio utilizzo OpenURL.

Una OpenURL consiste di un URL di base, che contiene l'indirizzo del OpenURL Resolver, seguito da una query string che contiene i dati sotto forma di coppie chiave-valore. I dati di solito sono informazioni bibliografiche ma dalla versione 1.0 di OpenURL è possibile includere anche informazioni sul richiedente, il tipo di servizio richiesto ecc. Un esempio di OpenURL 1.0 è:

```
https://nilde.bo.cnr.it/openurlresolver.php?url_ver=Z39.88-2004
&url_ctx_fmt=info:ofi/fmt:kev:mtx:ctx
&rft_val_fmt=info:ofi/fmt:kev:mtx:journal
&rft.atitle=NOETHER-LEFSCHETZ THEORY AND THE YAU-ZASLOW CONJECTURE
&rft.auinit=A
&rft.aulast=Klemm
&rft.date=2010
&rft.epage=1040
&rft.genre=article
&rft.issn=0894-0347
&rft.issue=4
&rft.place=PROVIDENCE
&rft.pub=AMER MATHEMATICAL SOC
&rft.spage=1013
&rft.stitle=J AM MATH SOC
&rft.title=JOURNAL OF THE AMERICAN MATHEMATICAL SOCIETY
&rft.volume=23
&rfr_id=info:sid/www.isinet.com:WoK:WOS
&rft.au=Maulik
&rft.au=Pandharipande
&rft.au=Scheidegger
```

Per facilitare la lettura dell'URL i parametri sono stati indicati su ogni riga. Come è possibile vedere lo standard prevede nomi specifici per i campi del metadato come ad esempio l'ISSN (&rft.issn=0894-0347), oppure il genere o il nome della rivista presente nel campo rft.title.

OpenURL è uno standard ANSI/NISO Z.39.88-2004, supporta diversi schemi di metadati (tra cui XML e Dublin Core) ed è stato allargato alla descrizione di una maggiore varietà di formati.

ISO-ILL

ISO-ILL[31][37] è un protocollo che è stato sviluppato come supporto per le biblioteche che offrono servizi di prestito interbibliotecario (DD/ILL) per migliorare il controllo e la gestione delle transazioni ILL sia per le attività di Lending che di Borrowing. Ci sono tre standard nella suite del protoccolo: ISO 10160:1997, ISO 10161-1:1997 e ISO 10161-2 che definiscono il protocollo a livello applicazione per la comunicazione tra i vari sistemi di scambio di documenti interbibliotecari. ISO-ILL consente a sistemi ILL di diverse biblioteche che risiedono su diverse piattaforme hardware di comunicare

e ricevere documenti elettronici anche utilizzando diversi sistemi software. Questo protocollo definisce il contenuto e il formato dei messaggi che possono essere trasmessi sia come messaggi di posta elettronica o direttamente tramite una connessione di rete tra la biblioteca richiedente e la biblioteca fornitrice, utilizzando una connessione X.25 o una connessione internet TCP / IP. Una transazione ILL inizia quando un messaggio di richiesta-ILL è inviato. Questo messaggio include un identificativo univoco della transazione appena iniziata e tutti i messaggi associati a questa transazione includeranno questo ID.

In generale, una transazione passa attraverso diversi stadi: richiesta, fallimento o notifica di spedizione dell'articolo, avviso di ricevimento dell'articolo, avviso di restituzione dell'articolo e avviso di ricevimento dell'articolo restituito. Lo standard divide questi stadi in due fasi principali: *processing phase* che inizia quando avviene una richiesta e termina quando la richiesta viene spedita oppure viene notificata come non disponibile; seguita dalla fase *tracking phase* in cui vengono monitorati gli stadi intermedi della richiesta.

Lo standard definisce inoltre tre ruoli che un sistema deve eseguire:

- **Richiedente** - In questo caso il sistema richiederà una copia o il prestito di un articolo per esempio e si preoccuperà di restituirlo qualora fosse richiesto.

- **Fornitore** - Il sistema riceverà la richiesta di prestito. In questo caso il sistema può elaborare la richiesta oppure può rispondere con un messaggio in cui viene indicato che la richiesta non può essere soddisfatta.

- **Intermediario** - E' un sistema contattato da un Fornitore il quale non è in grado di soddisfare una certa richiesta per cui la inoltrerà a un'altro sistema che avrà il ruolo di un nuovo Fornitore.

Il protocollo è anche estensibile, ogni messaggio contiene un campo di estensione che può trasportare più informazioni il cui formato e uso può essere definito al di fuori dello standard.

NCIP - NISO Circulation Interchange Protocol

NCIP[38] conosciuto anche come Z39.83 è un protocollo di comunicazione connection-oriented (orientato alla connessione) e sessionless (senza sessione) e utilizza HTTP per la trasmissione delle informazioni e SOAP per lo scambio di messaggi tra componenti software. Un protocollo orientato alla connessione facilita l'interazione tra le applicazioni e consente all'applicazione che richiede un servizio di sapere con sicurezza che un messaggio è stato ricevuto dall'applicazione partner. Esso definisce un insieme di messaggi e regole di sintassi e semantica per l'utilizzo da parte di applicativi software per l'Inter Library Loan e Document Delivery, fornisce un accesso controllato alle risorse elettroniche e facilita la cooperazione nella gestione di queste funzioni.

Nel contesto di condivisione delle risorse NCIP può essere utilizzato per eseguire una query sul sistema remoto per determinare se la risorsa desiderata (es. un articolo) è disponibile e, se lo è, chiedere al sistema remoto di inviarla.

NCIP presenta una struttura estensibile che include il protocollo e due tipi di profili.

- Il protocollo - Questo standard descrive i servizi, gli oggetti dei dati e gli elementi dei dati a un livello astratto. Il protocollo può essere implementato utilizzando differenti codifiche e differenti metodi di trasporto.

- Profili di implementazione - Ciascuno di questi profili specifica come i messaggi vengono scambiati. Le specifiche comprendono ad esempio il carattere e la codifica dei dati, componenti necessari, informazioni sul trasporto di rete e sulla sicurezza.

- Profili di applicazione - Descrivono come il protocollo deve essere utilizzato per supportare una specifica applicazione con un dato insieme di specifiche. Un profilo applicazione, include una descrizione dei servizi che devono essere supportati.

Questa struttura suddivisa in profili, permette al protocollo di essere supportato e rimanere flessibile ai possibili cambiamenti tecnologici, permettendo il suo utilizzo anche in più applicazioni.

E' possibile distinguere due gruppi principali di servizi offerti dal protocollo "resource sharing core" e "self-service core" ciascuno composto da 9 servizi di cui 5 sono condivisi da entrambi i gruppi e quattro sono unici per ogni campo di applicazione. I cinque servizi comuni sono: *Check In Item, Check Out Item, Lookup Item, Lookup User, Renew Item.* Mentre per il primo gruppo di servizi "Resource Sharing Core" i servizi sono: *Accept Item, Cancel Request Item, Item Recall, Item Request.* I quattro servizi base per il "Self-Service Core" sono: *Create User, Create User Fiscal Transaction, Undo Checkout Item, Update User.* La spiegazione dettagliata di questi servizi è esentata da questa trattazione, per un maggior approfondimento si rimanda alla descrizione del protocollo.

3.0.6 Funzionamento e utilizzo dei protocolli

Nella Figura 3.3 è possibile capire meglio il flusso della richiesta di un articolo da parte di un utente e l'ambito di utilizzo di ciascuno dei protocolli precedentemente descritti.

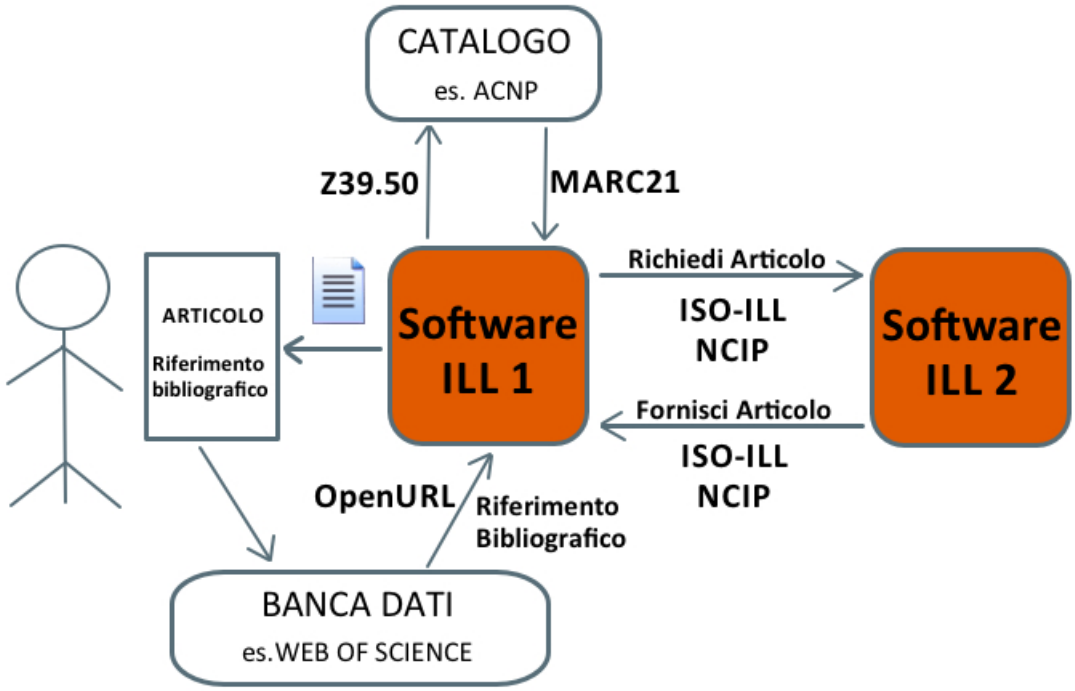

Figura 3.3: Schema generale della richiesta di un articolo.

Come si può vedere l'utente che desidera ricercare un articolo, può effettuare una ricerca su una banca dati (es. Web of Science). Una volta trovato l'articolo desiderato l'utente può procedere alla richiesta di tale articolo tramite un software di ILL (es. NILDE) il quale tramite un link OpenURL può leggere il riferimento bibliografico dell'articolo che verrà interpretato da un OpenURL Resolver all'interno del software di ILL. In questo modo, la biblioteca dell'utente, tramite il software di ILL può fornire l'articolo richiesto (se posseduto), oppure inoltrare la richiesta a un'altra biblioteca (che utilizza un'altro software di ILL) identificata tramite l'interrogazione dei catologhi (attraverso l'uso di protocolli standard es. Z39.50 e la codifica in MARC21

XML delle informazioni sui posseduti). I due software di ILL comunicano tra loro attraverso protocolli specifici (es. ISO-ILL o NCIP) per scambiarsi messaggi e informazioni per il Document Delivery. Il secondo software di ILL che riceve la richiesta dell'articolo può procedere all'evasione dell'articolo verso il primo software se possiede l'articolo richiesto oppure può effettuare un'ulteriore richiesta verso un'altro software di ILL.

3.1 NILDE

NILDE - Network Inter Library Document Exchange[39][40][41] è un servizio di Document Delivery basato su un software (Web Based) attorno al quale si è costituita una comunità di biblioteche disposte a condividere le loro risorse bibliografiche in spirito di collaborazione reciproca, prevalentemente, in maniera gratuita. E' quindi, sia un'applicazione web-based che un network di biblioteche che condividono una precisa e innovativa idea di servizio. NILDE è stato sviluppato nel 2001 dalla Biblioteca d'Area del CNR di Bologna, nell'ambito del progetto BiblioMIME, rivolto alla realizzazione di servizi avanzati di DD tra le biblioteche che ne hanno stimolato il continuo sviluppo e l'accrescimento di funzionalità innovative.

NILDE consente alla biblioteca di:

- Gestire in modo automatizzato le procedure connesse al Document Delivery
- Gestire lo scambio dei documenti tra biblioteche in modo elettronico sicuro attraverso un'interfaccia web con server dedicato, oppure tramite fax e posta ordinaria
- Offrire agli utenti della biblioteca un servizio personalizzato di Document Delivery
- Avere un bilancio dettagliato e sempre aggiornato degli scambi effettuati tra biblioteche, con la possibilità di misurare e confrontare le performance delle varie biblioteche, o della stessa negli anni

- Rilevare automaticamente gli indicatori di qualità "tasso di successo" e "tempo di fornitura"

- Far parte di un network fondato sulla condivisione di buone pratiche, di strumenti e di politiche per il miglioramento continuo dei servizi di Document Delivery.

NILDE come RapidILL ha un'anagrafica delle biblioteche centralizzata cioè viene mantenuto in un unico database l'intera anagrafica delle biblioteche aderenti a NILDE. Mentre GTBib e ILLiad essendo dei software con un'architettura distribuita mantengono una anagrafica locale delle biblioteche.

Attualmente NILDE è utilizzato da piu di 830 biblioteche e 15.000 utenti finali delle università e delle istituzioni di ricerca pubblica di tutta Italia. Attraverso NILDE è possibile effettuare ricerche nei cataloghi nazionali ACNP e SBN e tutti quelli che utilizzano il protocollo Z39.50. Nei suoi primi dieci anni di attività, NILDE ha elaborato circa 1 milione di richieste di documenti. I fattori chiave del successo di NILDE sono:

- La rete NILDE è costituita da una struttura organizzativa analoga a quella della comunità dei Software Open Source, in cui la qualità del sistema è garantita da un continuo feedback da parte della comunità stessa e in cui gli utenti del sistema sono anche i suoi produttori. Grazie al continuo feedback da parte della comunità, sono state implementate molte nuove funzionalità nel corso degli anni di attività.

- Reciprocità e gratuità, impegno a rendere visibile il proprio catalogo, rapidità di evasione delle richieste (mediamente 2 giorni lavorativi), equilibrio nella distribuzione delle richieste su tutte le biblioteche; la scelta della modalità con cui fornire i documenti è a discrezione della singola biblioteca: invio di una copia cartacea per posta o via fax oppure invio di una copia digitale temporanea attraverso un sistema di Internet delivery (Ariel, Prospero, NILDE).

3.1.1 Architettura di riferimento

In Figura 3.4 è mostrata l'architettura software di NILDE che nel corso degli anni ha subito notevoli cambiamenti e aggiornamenti. Essa è basata sulla ormai nota LAMP (Linux Apache MySQL PHP) su server Debian a 64bit. Nell'ultima versione di NILDE 4.0, sono state aggiunte nuove funzionalità che garantiscono alte prestazioni e scalabilità. Il codice sorgente è in PHP 5 che dispone di un solido modello *OOP - Object Oriented Programming* che permette di sfruttare tutti i vantaggi che questo stile di programmazione offre. L'interfaccia grafica è stata realizzata completamente in XHTML e attraverso l'impiego dei CSS; l'interazione con l'utente è stata gestita attraverso l'uso di Javascript e in particolar modo del framework open source AJAX Prototype. Per la realizzazione dei grafici viene utilizzata la libreria JpGraph. Le funzionalità esistenti dell'architettura sono:

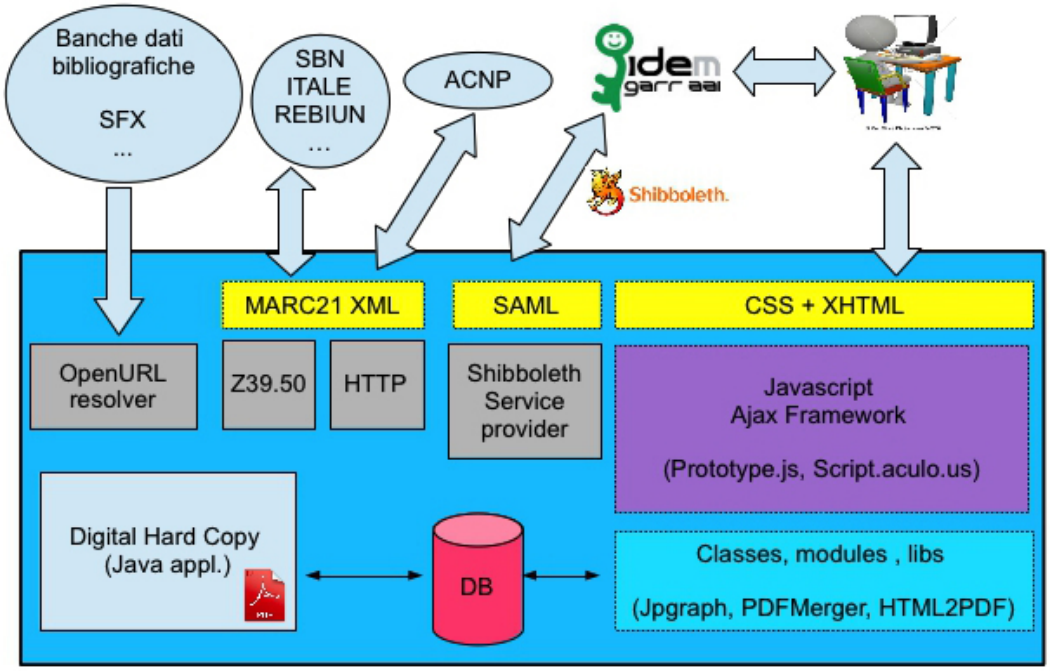

Figura 3.4: Architettura software NILDE 4.0.

- **Interfaccia grafica multilingue** - Grazie alla codifica dei dati di UTF-8, l'interfaccia utente NILDE 4.0 fornisce il supporto per più lingue e può essere tradotta facilmente in qualsiasi lingua. Attualmente NILDE è disponibile in italiano, inglese e francese, e gli utenti possono specificare la lingua preferita o usare la lingua di default del browser.

- **Integrazione con i processi di autenticazione degli utenti della federazione IDEM-GARR** (per l'accesso a servizi/risorse tramite il framework Shibboleth) - NILDE 4.0 oltre all'autenticazione standard basata su username e password, implementa l'autenticazione federata tramite IDEM-GARR in quanto Service Provider della federazione IDEM, concedendo l'accesso a tutti gli utenti della federazione IDEM-GARR. *IDEM - IDEntity Management* per l'accesso federato offre ai ricercatori degli istituti di ricerca, ai docenti e agli studenti delle università e delle scuole l'opportunità di poter usare una sola identità e una sola password per accedere a diversi servizi in rete (*SSO - Single sign-on*[5]). L'identità unica viene fornita dalla propria università, dalla propria scuola, in generale dalla propria organizzazione di appartenenza chiamata *Identity Provider*. In questo modo, i fornitori dei servizi in rete chiamati *Service provider* (es. NILDE) non avranno più bisogno di gestire onerose procedure di accreditamento e di amministrazione degli utenti. Infatti il processo di autenticazione è effettuato dall'istituzione di appartenenza (es l'Università), mentre l'autorizzazione è effettuata dal servizio esterno (es. NILDE). Tutti i membri della federazione IDEM devono implementare i propri servizi con software compatibili con lo standard SAML 2 e in particolare con l'implementazione di Shibboleth, un middleware standard open source per l'autenticazione e l'autorizzazione interistituzionale single sign-on sviluppato

[5]SSO, traducibile come autenticazione unica o identificazione unica è la proprietà di un sistema di controllo d'accesso che consente ad un utente di effettuare un'unica autenticazione valida per più sistemi software o risorse informatiche alle quali è abilitato.

dall'Associazione Internet2 e basato su SAML 2.0[6].

- **SEDD Secure Electronic DD** - NILDE permette a una biblioteca di inviare un documento richiesto da un'altra biblioteca in modo sicuro via Internet (utilizzando il modulo built-in NILDE SEDD), oppure via fax o per posta tradizionale, o utilizzando un altro sistema SEDD (ad esempio, Ariel). In questi ultimi casi (fax, posta ordinaria, Ariel) NILDE mantiene solo lo stato del database aggiornato e invia una e-mail informativa per la biblioteca richiedente, senza eseguire alcuna altra azione e nel caso in cui la biblioteca che effettua il Lending della richiesta sceglie la consegna elettronica, NILDE esegue SEDD tramite l'upload del file PDF. NILDE salva il file sul web-server, lo elabora attraverso la procedura digitale HardCopy (che viene eseguita su tutti i file PDF), quindi rende disponibile il documento digitale alla biblioteca richiedente per mezzo dell' interfaccia utente NILDE. La biblioteca richiedente dovrà autenticarsi in NILDE per visualizzare il documento ricevuto e stamparlo. Il documento verrà immediatamente distrutto dal server NILDE, dopo averlo stampato. In ogni caso, esso verrà rimosso dal server NILDE dopo 7 giorni.

- **Modulo Digital Hard Copy** - è un modulo software che fa parte del modulo SEDD per l'elaborazione dei file PDF. Infatti molti editori non consentono l'invio del file PDF originale ma solo una copia stampata. Per questo il modulo hard-copy aggiunto al processo di trasferimento dei file verso il web server emula le operazioni manuali, come la stampa del pdf e digitalizzazione attraverso uno scanner in modo da trasformare in PDF immagine il documento originale.

[6]E' uno standard informatico basato su XML per lo scambio di dati di autenticazione e autorizzazione (dette asserzioni) tra domini di sicurezza distinti, tipicamente un identity provider (entità che fornisce informazioni di identita) e un service provider (entità che fornisce servizi).

- **Utilizzo del protocollo OpenURL** - OpenUrl è un protocollo generale che permette a una Banca Dati, come Web of Science, di passare i metadati inerenti una citazione (come ad esempio l'ISSN, il titolo, l'autore, il volume, il fascicolo, il numero di pagina, etc..). Questo protocollo è implementato in NILDE in modo da garantire la comunicazione con le banche dati bibliografiche.

- **Cataloghi ACNP e SBN** - Una delle risorse principali di NILDE è il collegamento con i catalogghi nazionali ed in particolare con ACNP e SBN per la ricerca dinamica del posseduto delle biblioteche. L'implementazione di protocolli di comunicazione con questi cataloghi (HTTP + MARC21 XML per ACNP e Z39.50 + MARC21 XML per SBN) permette alla biblioteca richiedente di ottenere direttamente un elenco di possibili fornitori utilizzando il parametro ISSN nel caso la ricerca venga effettuata nel catalogo ACNP oppure usando il parametro ISBN nel caso in cui si stia interrogando il catalogo SBN. Questo porta chiaramente una notevole ottimizzazione della procedura di richiesta interbibliotecaria: la ricerca di una biblioteca fornitrice, diventa automatico e totalmente integrato in NILDE. In entrambi i casi le ricerche effettuate su questi cataloghi producono in output messaggi basati sugli standards XML e MARC che vengono poi interpretati da NILDE e presentati all'utente.

3.1.2 Funzionamento di NILDE

In figura 3.5 viene mostrato un'esempio di funzionamento del software NILDE. In partcolare un utente registrato a NILDE può effettuare una richiesta di un riferimento bibliografico alla sua biblioteca di appartenza. Ogni utente infatti in fase di registrazione dovrà indicare la sua biblioteca di riferimento alla quale potrà effettuare le richieste. NILDE mette a disposizione per i suoi utenti un reference manager in cui un utente puo tenere traccia di tutti i suoi riferimenti inseriti (ed eventualmente richiesti).

Figura 3.5: Funzionamento di NILDE.

Una volta che la biblioteca ha ricevuto la richiesta di un riferimento da parte di un suo utente, potrà evaderla direttamente se l'articolo richiesto

è posseduto dalla biblioteca oppure, tramite il software stesso potrà individuare una possibile biblioteca fornitrice tra quelle del circuito NILDE (attraverso l'interrogazione dei catologhi) alla quale potrà inoltrare la richiesta dell'articolo desiderato (operazione di Borrowing).

La biblioteca fornitrice tramite l'interfaccia di NILDE potrà procedere con l'operazione di Lending effettuando il caricamento del file sul server NILDE che lo renderà quindi disponibile per la biblioteca richiedente. Infine la biblioteca richiedente una volta ottenuto l'articolo, potrà darne comunicazione al suo utente inviandogli una copia stampata dell'articolo richiesto.

3.1.3 Analisi dei bisogni

L'analisi dei bisogni permette di individuare i requisiti che il sistema dovrà avere, al fine di raggiungere gli obiettivi prefissati. L'obiettivo è appunto quello di estendere l'architettura esistente di NILDE rendendo disponibili i servizi di NILDE accessibili dall'esterno ad esempio da altri programmi di ILL, in modo tale da sviluppare e fornire un insieme di API pubbliche e private con le quali accedere a tutte le funzionalità del sistema. In particolare si è pensato di sviluppare dei servizi che permettano ad applicazioni esterne (es. GTBib) di poter interagire con NILDE in modo tale da mantenere gli stati delle richieste di Borrowing e Lending sincronizzati tra i vari software di Document Delivery.

Un'altro interessante caso potrebbe essere quello di sviluppare un App mobile per smartphone e tablet la quale potrebbe sfruttare i servizi offerti da NILDE attraverso le chiamate API di NILDE.

L'obiettivo di questo lavoro di tesi è infatti la progettazione e l'implementazione di servizi conformi ai principi dell'architettura REST in cui la comunicazione dovrà avvenire attraverso semplici richieste HTTP, il cui output sarà formattato in XML oppure JSON. Attraverso un'architettura di questo tipo qualsiasi applicazione esterna potrà comunicare con NILDE in modo autonomo sfruttando le API messe a disposizione.

3.1.4 Analisi dei requisiti

Questa è un'attività preliminare allo sviluppo o alla modifica di un sistema software, il cui scopo è quello di definire le funzionalità che il nuovo prodotto o il prodotto modificato deve offrire, ovvero i requisiti che devono essere soddisfatti dal software sviluppato. I requisiti si possono dividere in requisiti funzionali che descrivono le funzionalità ed i servizi forniti dal sistema e i requisiti non funzionali che non sono collegati direttamente con le funzioni implementate dal sistema, ma piuttosto alle modalità operative e di gestione definendo ad esempio vincoli sullo sviluppo del sistema.

Requisiti funzionali

- Estendere le funzionalità principali di NILDE per essere accessibili e fruibili da altri sistemi come ad esempio software di Document Delivery o applicazioni Mobile.
- Possibilità di utilizzare le API REST di NILDE tramite chiamate AJAX in Cros Domain.
- Per utilizzare le API NILDE è obbligatorio richiedere una chiave di applicazione (API Key e Secret).
- Per i servizi pubblici non è richiesta l'autenticazione, ma viene richiesta l'API Key e il Secret per consentire l'accesso a tali servizi solamente ad alcuni sistemi.
- Per i servizi privati è necessaria l'autenticazione con il proprio account NILDE, (utenti e biblioteche); è inoltre necessaria una chiave di accesso APY Key e Secret da richiedere al gestore. Per questi servizi l'autenticazione verrà implementata utilizzando il protocollo OAuth.
- Le API Key e il Secret sono diverse per ogni sistema che vorrà utilizzare le API NILDE e andranno indicate come parametri in tutte le chiamate REST a NILDE.

- API Key e Secret possono essere richieste al gestore di NILDE compilando un apposito form.
- Il formato della risposta (anche in caso di errore) di una chiamata REST verso NILDE sarà XML contenente un messaggio .

Requisiti non funzionali

- Le API NILDE utilizzate per implementare i servizi offerti, verranno esposte publicamente sul sito di NILDE in modo tale che chiunque potrà documentarsi e utilizzarle previa la richiesta di un API key e Secret che verrà fornita solo previa autorizzazione.
- Il tempo di risposta a una chiamata REST utilizzando le API messe a disposizione non dovrà superare l'ordine dei 5 secondi.
- Le API sviluppate dovranno rispettare i vincoli e i principi dei servizi REST.
- Essendo NILDE un' applicazione web sviluppata utilizzando il linguaggio di programmazione PHP, tutti i metodi che si andranno a sviluppare dovranno essere programmati utilizzando lo stesso linguaggio di programmazione.
- Garantire l'autenticità dell'applicazione e/o dell'utente che effettua la chiamata ai servizi utilizzando le API NILDE.
- Il sistema deve essere facilmente espandibile, e adattabile alle future esigenze.

Capitolo 4

Progettazione e sviluppo servizi REST in NILDE

In questo capitolo verrà esposta la soluzione software sviluppata per implementare i servizi di NILDE che vengono resi fruibili attraverso un'insieme di API pubbliche e private con le quali è possibile accedere alle funzionalità del sistema. Come è possibile vedere dalla Figura 4.1, l'architettura software di NILDE è stata estesa in riferimento alla figura 3.4 in modo da aggiungere queste nuove funzionalità che garantiscono un'aumento della scalabilità dell'intero sistema e assicurano un elevata interoperabilità con altri sistemi.

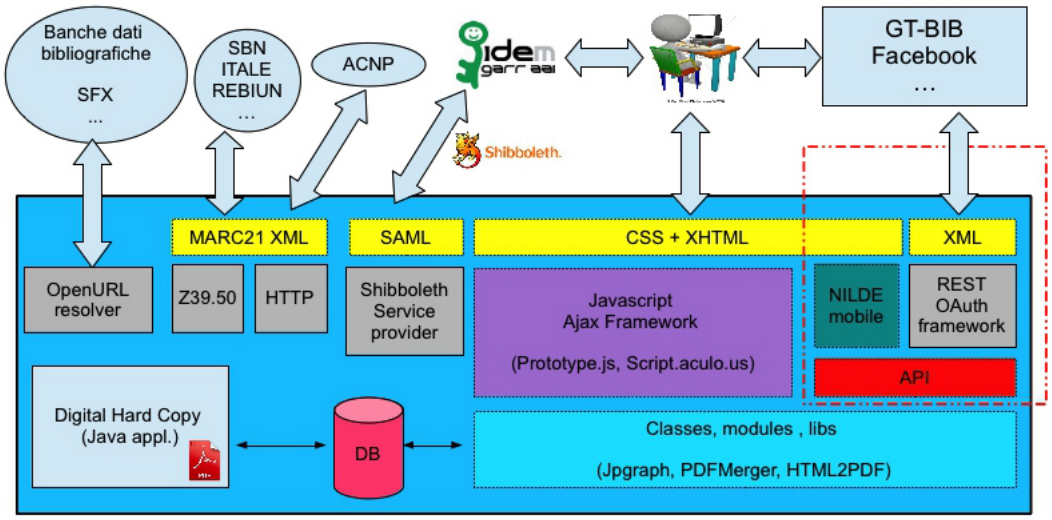

Figura 4.1: Architettura software NILDE + API.

Nel riquadro tratteggiato, vengono indicati appunto gli strati software che sono stati inseriti nel sistema per permettere lo sviluppo delle API REST. Il blocco API indica appunto l'insieme dei servizi (vedi paragrafo 4.5) messi a disposizione da NILDE, mentre il blocco grigio sta ad indicare i framework utilizzati per la creazione dei servizi REST e per la gestione dell'autenticazione / autorizzazione realizzata tramite l'utilizzo di OAuth. Il riquadro NILDE Mobile indica lo sviluppo futuro di un Applicazione Mobile o di un sito Mobile che sfrutterebbe le API REST per fornire i servizi all'utente che desidera interagire con l'applicazione o con il sito Mobile. Infine il riquadro XML indica che i dati scambiati (esempio le risposte alle chiamate API) vengono rappresentati attraverso l'utilizzo del linguaggio XML (Vedi paragrafo 1.2.3).

Nel corso di questo capitolo, verranno descritti i principali framework esistenti utilizzati per facilitare lo sviluppo delle API REST e infine verranno trattati nel dettaglio i servizi sviluppati focalizzando lo studio sulla definizione delle risorse, dei parametri previsti nelle chiamate API e sulla progettazione delle risposte del server in caso di chiamate con esito positivo o negativo.

4.1 Descrizione della migrazione

In questo paragrafo viene descritto cosa è stato realizzato per questo lavoro di tesi, analizzando le scelte implementative e le tecnice utilizzate. Come è stato già detto, lo scopo di questo lavoro è quello di rendere NILDE accessibile dall'esterno in modo da permettere a utenti di interagire con le funzionalità del sistema. Si è pensato quindi di estendere l'architettura software esistente basandosi sul paradigma architetturale REST i cui principi hanno permesso l'implementazione di un set di servizi pubblici e privati:

- le API pubbliche di NILDE, permettono di interagire con il sistema attraverso le chiamate a metodi che non richiedono autenticazione ma che necessitano l'utilizzo di una API key e un Secret che dovranno essere richiesti al gestore e rilasciati previa autorizzazione.

- le API private invece oltre a richiedere l'utilizzo della API key e del Secret, necessitano dell'autenticazione con il proprio account NILDE per interagire con i servizi.

API key e Secret, possono essere richieste al gestore di NILDE compilando un apposito form (URL), sono diversi per ogni sistema / utente che vorrà utilizzare le API di NILDE e andranno indicati come parametri obbligatori in tutte le chiamate REST a NILDE.

Per entrambi i servizi (Pubblici e Privati), la gestione dell'autenticazione e autorizzazione attraverso API key e Secret per e API pubbliche e API key, secret + username e password per le API private, verrà implementata utilizzando il protocollo OAuth[24] (vedi paragrafo 4.3). In particolare per le API private per le quali è richiesta l'autenticazione, verrà richiesto di effettuare il login tramite un form.

Si è reso necessario però apportare una modifica al metodo di gestione dell'autenticazione in quanto per alcuni sistemi che dovranno interagire con NILDE risulta scomodo effettuare il login tramite l'inserimento dei dati nel form. Per cui è stata sviluppata la funzionalità di Direct Login con la

quale è possibile specificare le credenziali di accesso (username e password) direttamente nel'URL della chiamata API. La password in questo caso, verrà codificata utilizzando l'algoritmo MD5[1].

Le chiamate alle API di NILDE potranno essere effettuate attraverso l'utilizzo di AJAX in Cros Domain. Questo è stato implementato grazie all'utilizzo della specifica CORS (vedi paragrafo 4.4.1.3 e 4.6) che permette a un server di distinguere la provenienza di una richiesta e decidere se servirla.

Nei prossimi paragrafi verrà spiegato con maggior dettaglio tecnico come queste funzionalità sono state implementate in NILDE.

4.2 Framework PHP per servizi web REST

Come affermato nel paragrafo 1.4 REST è un tipo di architettura piuttosto che un protocollo, per questo motivo sono indispensabili framework e librerie che favoriscono e facilitano lo sviluppo delle API. Infatti in rete molti siti web si sforzano di dare indicazioni su quali framework devono essere utilizzati per lo sviluppo di API in REST. In realtà non esiste un framework che vada bene per tutti gli scopi, dipende sempre dallo scenario e dalla tecnologia che si vuole utilizzare. Per quanto riguarda la programmazione in PHP, esistono molti framework Open Source che consentono lo sviluppo di API REST. I fattori da considerare nella scelta di un framework sono essenzialmente tre:

- deve avere l'architettura REST integrata nel framework;
- deve essere stabile e semplice da utilizzare;
- deve fornire una documentazione dettagliata delle funzionalità;

Di seguito vengono elencati i principali framework per lo sviluppo di API REST in PHP, con particolare attenzione a quello utilizzato in questo lavoro di tesi.

[1]Indica un algoritmo crittografico di hashing che prende in input una stringa di lunghezza arbitraria e ne produce in output un'altra a 128 bit.

Recess!

Recess[42] è un framework PHP Open Source rilasciato sotto licenza MIT, utilizza una architettura *MVC - Model-View-Controller*[2] che permette di essere più flessibile, manutenibile, aggiornabile nel tempo e facilita la riusabilità del codice. E' stato progettato e ottimizzato specificamente per lo sviluppo di API RESTful utilizzando il protocollo HTTP. Recess! si basa su PHP5 e include un database *ORM - Object-Relational Mapping*[3] in modo da evitare query SQL complesse. Inoltre prevede un'interfaccia grafica e un set di strumenti per accelerare la fase di sviluppo.

Zend Framework

Zend[43] è un framework open source rilasciato con licenza BSD per lo sviluppo di servizi e applicazioni web basato su PHP 5. Per questo motivo Zend utilizza la maggior parte delle nuove caratteristiche di PHP 5 come ad esempio gli oggetti. Zend è stato progettato con lo scopo di semplificare l'attività di sviluppo web ed agevolare la produttività mettendo a disposizione una serie di librerie e componenti. Ciò che differenzia questo framework da altri framework PHP è la sua particolare struttura a componenti indipendenti. Infatti, l'indipendenza delle sue componenti permette di utilizzare solo il necessario per il proprio progetto, ma allo stesso tempo, se utilizzate insieme, esse si integrano indirizzando il lavoro dello sviluppatore verso soluzioni basate sui principi di riutilizzo del codice, di estendibilità, leggibilità e manutenzione.

[2]E' un pattern architetturale molto diffuso nello sviluppo di sistemi software, in particolare nell'ambito della programmazione orientata agli oggetti, in grado di separare la logica di presentazione dei dati dalla logica di business.

[3]E' una tecnica di programmazione che favorisce l'integrazione di sistemi software aderenti al paradigma della programmazione orientata agli oggetti con sistemi RDBMS. Un database ORM fornisce, mediante un'interfaccia orientata agli oggetti, tutti i servizi inerenti alla persistenza dei dati, astraendo nel contempo le caratteristiche implementative dello specifico RDBMS utilizzato.

In particolare Zend dispone della libreria Zend_REST che offre agli sviluppatori PHP un approccio incredibilmente semplice per creare servizi Web RESTful. Questa libreria, fornisce un'insieme di componenti Zend_Rest_Server e Zend_Rest_Client che offrono funzionalità aggiuntive sia lato Client che Server.

Slim

Slim[44] è un micro framework PHP rilasciato sotto licenza MIT che offre funzionalità di routing per creare rapidamente applicazioni e servizi web RESTful semplici ma potenti. Una interessante e potente, caratteristica è il suo concetto di middleware. Slim implementa il protocollo Rack, un'architettura pipeline comune a molti framework Ruby. Grazie ai middleware è possibile modificare l'esecuzione dei metodi standard, le richieste e le risposte prima e / o dopo che l'applicazione Slim viene richiamata. Si pensi all'applicazione Slim come il nucleo di una cipolla (vedi Figura 4.2), in cui ogni strato è un middleware. Quando si richiama il metodo run() dell'applicazione Slim, lo strato più esterno (Authentication) viene richiamato per primo ed è responsabile dell'invocazione del successivo strato middleware che lo circonda fino a quando viene richiamata l'applicazione nucleo Slim.

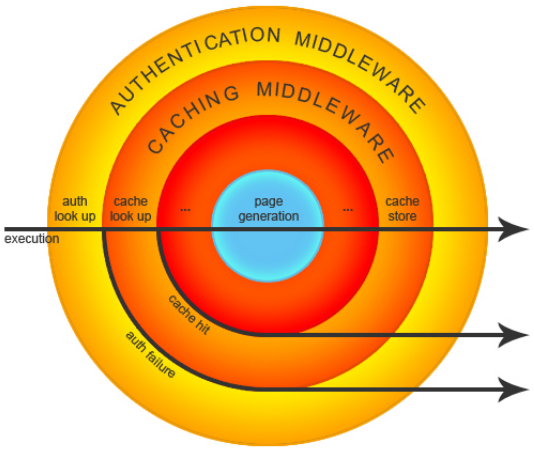

Figura 4.2: Slim Framewok: Middleware.

Altre caratteristiche di questo framework sono:

- RESTful routing;
- HTTP caching;
- possibiltà di personalizzare i metodi HTTP;
- possibilità di personalizzare gli errori;
- gestione dei log.

4.3 Sviluppo di un OAuth provider

Un' applicazione web che mette a disposizione degli utenti alcuni o tutti i suoi servizi attraverso l'utilizzo di API, richiede un meccanismo di sicurezza per garantire l'autenticazione e l'autorizzazione di chi effettua le chiamate ai servizi tramite le API. A tale scopo è stato utilizzato OAuth che come spiegato nel paragrafo 2.4 è un protocollo che implementa una metodologia di autenticazione basata su API e l'utilizzo di token attraverso il quale un determinato servizio web può garantire, per conto di un Utente, un sorta di "accesso speciale" a differenti Applicazioni/Siti Web permettendo loro di accedere ai dati di quell'Utente in maniera sicura e protetta senza che l'utente condivida la sua password con servizi o Applicazioni diversi da quelli dei Servizi Web che custodiscono i suoi dati privati.

Per lo sviluppo dell' OAuth provider è stato utilizzato un framework PHP[45] sviluppato da Freek Lijten sotto licenza BSD e reso disponibile sul sito GitHub[46]. Il codice utilizzata le librerie PECL[4] OAuth per implementare il flusso di lavoro di OAuth. Al framework OAuth utilizzato sono state

[4]PECL - *PHP Extension Community Library* è un insieme di estensioni PHP, include moduli per il parsing XML, supporto per database non contenuti nella libreria standard e inclusione di programmi Perl o Python all'interno di script PHP.E' accessibile tramite il sistema di packaging PEAR per questo è necessario che entrambi siano installati sul sistema sul quale si vuole installare OAuth.

apportate una serie di modifiche per realizzare funzionalità aggiuntive come ad esempio quella del direct login (in cui si evita di mostrare il form di login passando direttamente le credenziali (username e password) dell'User direttamente nella richiesta HTTP GET) oppure il salvataggio dell'Access Token in sessione in modo da non richiedere il login se si effettua una seconda chiamata API in un certo arco di tempo.

OAuth è disponibile per molti linguaggi di programmazione tra cui Java, Python, DotNET e PHP.

4.3.1 Creazione API Key e Secret

Il primo aspetto da considerare una volta installata la libreria OAuth 1.0, è quello di sviluppare il codice per la generazione di API Key e Secret per un Consumer. In particolare è stata sviluppata una pagina web apposita in cui viene mostrato un form da compilare con i dati del Consumer che vuole richiede l'API Key e il Secret. Una volta compilato il form l'utente che ha effettuato la richiesta riceverà tramite email dopo l'approvazione dell'amministratore l'API Key e il Secret che potrà utilizzare per effettuare le chiamate API REST ai servizi di NILDE.

Nel dettaglio il codice per la generazione dell'API Key e del Secret è il seguente:

```
$Consumer = new OAuthConsumerModel(Configuration::getDataStore());
$Consumer->setConsumerCreateDate(time());
$Consumer->setConsumerKey(OAuthProviderWrapper::generateToken());
$Consumer->setConsumerSecret(OAuthProviderWrapper::generateToken());

$Consumer->setConsumerName($nome);
$Consumer->setConsumerPassword($pass_md5);
$Consumer->setConsumerEmail($email);
        try {
                $Consumer->save();
        } catch (DataStoreCreateException $Exception) {
```

```
                echo $Exception->getMessage();
                exit;
        }
```

OAuthConsumerModel è la classe in cui sono state implementate le funzioni per le operazioni *CRUD - Create Read Update Delete*, ovvero le quattro operazioni base di un database. Configuration::getDataStore() invece rappresenta la soluzione per definire in un unico punto la connessione al database (MySQL in questo caso).

```
public static function getDataStore() {
static $DataStore;
global $DBHOST,$DBLOGIN,$DBNAME,$DBPASS,$DBPORT;

$DataStore=new mysqli();
$DataStore->init();
$DataStore->real_connect($DBHOST,$DBLOGIN,$DBPASS,
                         $DBNAME,$DBPORT);
```

Infatti nel file Configuration.php viene definito il metodo statico getDataStore() in cui viene istanziato un nuovo oggetto mysqli $DataStore sul quale si esegue la funzione real_connect che apre una connessione con il server mysql. Il metodo statico generateToken() di ProviderWrapper genera un token alfanumerico di 40 caratteri criptato attraverso la funzione PHP sha1(). Una volta settate le informazioni del Consumer attraverso i metodi setter, viene richiamato il metodo save() il quale provvedere a scrivere sul database i dati del Consumer.

4.3.2 Configurare gli Endpoints

Come è stato spiegato nel paragrafo 2.4.2 il flusso di autenticazione di OAuth prevede la definizione di quattro endpoints cioè le URI che permettono la richiesta di autenticazione con OAuth. In particolare nella versione 1.0 di OAuth vengono definiti i seguenti endpoints:

- Request Token Endpoint - L'endpoint utilizzato dal Consumer per ottenere un insieme di credenziali temporanee (request token).

- User Authorization Endpoint - L'authorization endpoint è l'endpoint sul Service Provider richiamato quando il proprietario della risora (User) effettua l'accesso presso il Service Provider e concede l'autorizzazione all'applicazione client (Consumer).

- Access Token Endpoint - E' l'endpoint presente sul Service Provider usato dal Consumer il quale scambia il request token ricevuto in precedenza per un access token.

- Protected Resource Endpoint - E' l'endpoint in cui il proprietario della risorsa viene reindirizzato in modo da ottenere le risorse private (Protected resources) dopo aver concesso l'autorizzazione nel User Authorization Endpoint.

4.3.2.1 Request Token Endpoint

Il primo endpoint ad essere implementato è appunto quello per la richiesta delle credenziali temporanee (request token) dal parte del Consumer. Questo endpoint presente sul Service Provider viene chiamato tramite una richiesta HTTP GET passando nella URI i parametri API Key e Secret + parametri opzionali del servizio oppure API Key, Secret, Username e Password (in caso di direct login[5]) + parametri opzionali del servizio (per la descrizione dettagliata dei servizi e dei relativi parametri si veda il capitolo).

Il primo file PHP ad essere richiamato è quindi get_request_token.php:

```
$OAuth = new OAuth($consumerKey, $consumerSecret);
$tokenInfo = $OAuth->getRequestToken($requestURL.
        '?oauth_callback='.$callbackURL.'&scope=all');
```

[5]Con il direct login si evita di mostrare il form di login generato nell'User Authorization Endpoint passando direttamente le credenziali (username e password) dell'User direttamente nella richiesta HTTP GET.

Come prima cosa viene istanziato un oggetto OAuth indicando la API Key e il secret del Consumer forniti nella richiesta HTTP. Attraverso il metodo getRequestToken() verrà generato un Request Token che verrà salvato nella variabile $tokenInfo.

Il parametro oauth_callback rappresenta l'URI utilizzata dal server per reindirizzare il proprietario della risorsa (User) quando la fase di User Authorization Endpoint è completata e quindi bisognerà procedere con Access Token Endpoint.

Mentre la variabile $requestURL, indica l'indirizzo fisico del file request_token.php in cui vi è il codice che si occupa di generare i request token:

```
$Provider = new OAuthProviderWrapper
        (OAuthProviderWrapper::TOKEN_REQUEST);
$response = $Provider->checkOAuthRequest();
if ($response !== true) {
        echo $response;
        exit;
}
```

Tralasciamo per ora la spiegazione del metodo checkOAuthRequest() il suo scopo e il suo funzionamento saranno chiariti più avanti, e analizziamo il costruttore della classe OAuthProviderWrapper con il quale viene istanziato l'oggetto $Provider:

```
public function __construct($mode) {
        $this->Provider = new OAuthProvider();
        $this->Provider->consumerHandler(array
                        ($this,'consumerHandler'));
        $this->Provider->timestampNonceHandler(array
                        ($this,'timestampNonceHandler'));
        if ($mode == self::TOKEN_REQUEST) {
        $this->Provider->isRequestTokenEndpoint(true);
        //enforce the presence of these parameters
        $this->Provider->addRequiredParameter("oauth_callback");
```

```
            $this->Provider->addRequiredParameter("scope");
            }
            ...
}
```

Come prima cosa viene istanziato un oggetto OAuthProvider e vengono definiti i due metodi di callback consumerHandler() e timestampNonceHandler(). Dopodichè, dato che è stato richiesto un request token (OAuthProviderWrapper::TOKEN_REQUEST) verrà eseguito il codice all'interno dell'if dove vengono definite le chiamate alle funzioni addRequiredParameter() che indicano all'oggetto OAuthProvider la definizione dei parametri scope e oauth_callback. Se questi due paramentri non sono presenti, qualsiasi richiesta al Request Token Endpoint deve essere considerata invalida.

Consumer Handler

La prima delle due funzioni di callback specificate nel listato precedente è il consumerHandler. Questa funzione, verifica se le informazioni del Consumer (API Key e Secret) fornite nella richiesta sono corrette. Questa funzione deve restituire una delle seguenti costanti PECL OAuth: OAUTH_CONSUMER_KEY_UNKNOWN, OAUTH_CONSUMER_KEY_REFUSED o OAUTH_OK.

```
public static function consumerHandler($Provider) {
try {
$OAuthConsumer = OAuthConsumerModel::loadFromConsumerKey
        ($Provider->consumer_key, Configuration::getDataStore());
} catch (DataStoreReadException $Exception) {
        return OAUTH_CONSUMER_KEY_UNKNOWN;
}
$Provider->consumer_secret = $OAuthConsumer->getConsumerSecret();
return OAUTH_OK;
}
```

Come si può notare questa funzione non fa altro che verificare all'interno del database se la API Key fornita come parametro della chiamata API al servizio è associata a un Consumer oppure no. In caso negativo verrà generata un'accezione e sarà restituito il valore della costante OAUTH_CONSUMER_KEY_UNKNOWN; altrimenti se la API Key corrisponde a un Consumer, verrà salvato anche il relativo Secret all'interno di una variabile e sarà restituito il valore della costante OAUTH_OK.

La costante OAUTH_CONSUMER_KEY_REFUSED indica che l' API Key e il Secret sono temporaneamente invalide ad esempio nel caso in cui il Service Provider supporta una blacklist oppure stabilisce un limite alle richieste che un determinato Consumer può effettuare utilizzando una determinata API Key.

Timestamp e Nonce Handler

La seconda funzione di callback definita precedentemente nel costruttore OAuthProviderWrapper controlla due cose: il timestamp e il nonce entrambi richiesti dalla specifica OAuth. Nonce (*number used once*) indica un numero, generalmente casuale o pseudo-casuale, che ha un utilizzo unico. Per cui tramite l'utilizzo dei nonce e del timestamp viene assicurato che i dati scambiati nelle vecchie comunicazioni / richieste non possano essere riutilizzati in attacchi di tipo replay attack.

```
public static function timestampNonceHandler($Provider) {
if (time() - $Provider->timestamp > 300)
        return OAUTH_BAD_TIMESTAMP;
if (OAuthNonceModel::nonceExists($Provider->nonce,
                        Configuration::getDataStore()))
        return OAUTH_BAD_NONCE;

$OAuthNonce = new OAuthNonceModel(Configuration::getDataStore());
$OAuthNonce->setId($Provider->nonce);
$OAuthNonce->setNonceConsumerKey($Provider->consumer_key);
$OAuthNonce->setNonceDate(time());
$OAuthNonce->save();

return OAUTH_OK;
}
```

Quindi a ogni richiesta (request token) verranno memorizzati sul database i nonce e il timestamp corrente relativi al Cosumer che ha effettuato la richiesta e saranno ignorare tutte le richieste vecchie più di 5 minuti. Se il controllo timestamp effettuato nel primo if va a buon fine, verranno controllati attraverso il secondo if se i nonce sono già presenti nel database. Se sono già presenti, la richiesta sarà invalidata. Anche in questo caso i valori di ritorno della funzione sono delle costanti predefinite in OAuth.

Una volta che il Service Provider ha verificato la richiesta, risponderà al Consumer con un set di credenziali temporanee incluse nel body della risposta HTTP:

```
HTTP/1.1 200 OK
Content-Type: application/x-www-form-urlencoded

oauth_token=hdk48Djdsa&oauth_token_secret=xyz4992k83j47x0b&
oauth_callback_confirmed=true
```

4.3.2.2 User Authorization Endpoint

Quando il Consumer ha ricevuto il request token, dovrà rendirizzare lo User all' authorization endpoint cioè alla pagina di autenticazione dove il proprietario della risora (User) effettua l'accesso presso il Service Provider e concede l'autorizzazione all'applicazione client (Consumer). Il reindirizzamento avverrà utilizzando il parametro oauth_token=hdk48Djdsa ricevuto in precedenza e in aggiunta altri parametri opzionali. Un esempio di reindirizzamento è questo:

```
GET /authorize.php?oauth_token=hdk48Djdsa HTTP/1.1
Host: nilde.bo.cnr.it
```

Quando si chiede al proprietario della risorsa di autorizzare l'accesso alle sue risorse private da parte di un Consumer, il server dovrebbe presentare allo User le informazioni dell'applicazione client che richiede l'accesso temporaneo alle risorse private dell'utente che sta effettuando il login.

Una volta che l'User ha effettuato il login e ha concesso il permesso al Consumer di accedere alle sue risorse private, l'ID dell'utente verrà memorizzato sul database associandolo al request token e a un verification token che verrà inviato al Consumer per assicurare che il proprietario della risorsa che sta concedendo l'accesso alle risorse private è lo stesso utente che ritornerà nella pagina del Consumer.

```
//Verifica username e password dell'utente
if ($row['ut_pwd'] != md5($_POST['user_password'])) {
        XMLCreator::sendXMLError(403, "Wrong authentication.");
        exit;
}

$verificationCode = OAuthProviderWrapper::generateToken();
$RequestToken->setTokenVerificationCode($verificationCode);
$RequestToken->setTokenUserId($row['ut_id']);
    try {
```

```
                $RequestToken->save();
        } catch (DataStoreUpdateException $Exception) {
                echo $Exception->getMessage();
                exit;
        }
header( 'location: ' . $RequestToken->getTokenCallback() .
'?oauth_token='.$RequestToken->getToken().'&oauth_verifier='.
$verificationCode.'&consumerKey='.$consumerKey.
'&consumerSecret='.$consumerSecret.'&par='.$parameters);
```

Come si può notare dal listato precedente una volta che l'utente ha effettuato il login in modo corretto viene generato un verification token tramite il metodo statico generateToken() della classe OAuthProviderWrapper e viene salvato insieme all'ID dell'utente sul database attraverso il metodo save(). Dopodichè l'utente sarà reindirizzato verso la URI di callback specificando obbligatoriamente i parametri oauth_token e oauth_verifier. In questo caso sono stati aggiunti ulteriori parametri come l'API Key e il Secret del Consumer e altri parametri aggiuntivi utilizzati in fase di esecuzione dei servizi implementati. Un'esempio del metodo GET HTTP per il reindirizzamento verso la URI di callback potrebbe essere questo:

```
GET /get_access_token.php?oauth_token=hdk48Djdsa&oauth_verifier=473f82d3 HTTP/1.1
Host: client.example.net
```

4.3.2.3 Access Token Endpoint

Ora che è stato emesso il request token ed è stato autorizzato dall'User c'è ancora un ulteriore passo da completare prima che il codice dell'API chiamata possa essere eseguito. Bisogna generare un access token e scambiarlo con il request token autorizzato del Consumer. In questo caso il costruttore di OAuthProviderWrapper viene richiamato con una costante differente (TOKEN_ACCESS):

```
$Provider = new OAuthProviderWrapper
        (OAuthProviderWrapper::TOKEN_ACCESS);
$response = $Provider->checkOAuthRequest();
if ($response !== true) {
        echo $response;
        exit;
}
```

Il costruttore di OAuthProviderWrapper eseguirà il codice presente nell'else if:

```
public function __construct($mode) {
$this->Provider = new OAuthProvider();
$this->Provider->consumerHandler(array($this,'consumerHandler'));
$this->Provider->timestampNonceHandler(array
                        ($this,'timestampNonceHandler'));
if ($mode == self::TOKEN_REQUEST) {
        ...
} else if ($mode == self::TOKEN_ACCESS) {
        $this->Provider->tokenHandler(array
                        ($this,'checkRequestToken'));
} ...
}
```

In particolare quando si vuole generare un access token vengono effettuati dei controlli sul request token e il verification token tramite il metodo checkRequestToken():

```
public static function checkRequestToken($Provider) {
$DataStore = Configuration::getDataStore();

//Token can not be loaded, reject it.
try {
        $RequestToken = OAuthRequestTokenModel::loadFromToken
```

```
                        ($Provider->token, $DataStore);
} catch (DataStoreReadException $Exception) {
        return OAUTH_TOKEN_REJECTED;     }

//The consumer must be the same as the one this
//request token was originally issued for
if ($RequestToken->getTokenConsumerKey() !=
                        $Provider->consumer_key) {
        return OAUTH_TOKEN_REJECTED;
}

//Check if the verification code is correct.
if ($_GET['oauth_verifier'] != $RequestToken->
                        getTokenVerificationCode()) {
        return OAUTH_VERIFIER_INVALID;
}
$Provider->token_secret = $RequestToken->getTokenSecret();
return OAUTH_OK; }
```

In particolare viene effettuato un primo controllo sull'esistenza del token, viene verificato che il Consumer sia lo stesso per il quale è stato rilasciato il request token e infine viene verifacata se il verification code è corretto. Se questi controlli passano, viene generato un access token e il request token viene eliminato dal database perchè non più necessario.

A questo punto il Consumer dispone di tutti i dati necessari per effettuare una chiamata autorizzata alla API e ottenere le risorse private dell'User.

4.3.2.4 Protected Resource Endpoint

Le API di NILDE sono state sviluppate nel file api.php il quale prima di eseguire il codice relativo alla API richiesta effettua dei controlli sulla chiamata in particolare sull'access token fornito tramite la funzione OAuthProviderWrapper::TOKEN_VERIFY.

```
$Provider = new OAuthProviderWrapper
                (OAuthProviderWrapper::TOKEN_VERIFY);
$response = $Provider->checkOAuthRequest();
if ($response !== true) {
        echo $response;
        exit;
}
```

I controlli effettuati sull'access token sono gli stessi di quelli effettuati per il request token. Se questi controlli vanno a buon fine è possibile eseguire il codice relativo all'API richiesta e il tal caso fornire l'output previsto.

```
//Per ogni servizio vengono eseguite delle funzioni diverse
switch ($arrayParameters['serviceName']) {
        case 'datasospensionedd':
                Classe::datasospensionedd();
        break;
        case 'esempioServizio':
                ...
        break;
                ...
        default:
                XMLCreator::sendXMLError(501, "Method $arrayParameters['serviceName']
                        not found.");
                exit;
        break;
}
```

In particolare verrà eseguito uno switch case che in base al nome del servizio e ai parametri passati eseguirà la funzione appropriata per restituire o modicare i dati desiderati.

4.4 Progettazione dei servizi REST

Per lo sviluppo delle API REST di NILDE è stato scelto Slim framework per vari motivi. In prima analisi, il framewrok Recess! è stato escluso perchè non è risultato molto robusto rispestto agli altri due framework e inoltre molti sviluppatori si sono lamentati della difficoltà nello sviluppare servizi web utilizzando questo framework. Zend framework invece che risulta essere uno dei migliori framework PHP Open Source il quale è anche supportato da Flickr, Amazon Yahoo e Google, presenta alcuni svantaggi. In particolare uno sviluppatore che utilizza Zend, ha bisogno di scrivere molte righe di codice PHP anche solo per fare semplici operazioni risultando così molto lento nei casi in cui c'e bisogno di sviluppare servizi web semplici e veloci. Inoltre la sua curva di apprendimento è enorme se confrontata agli altri due framework. Slim Framework a primo impatto invece è risultato molto flessibile e di facile utilizzo grazie alla sua documentazione dettagiata che in breve tempo ha permesso lo sviluppo delle API REST per NILDE.

Come spiegato nel paragrafo 1.4 quando si parla di REST non si fa più riferimento all'architettura *SOA - Services Oriented Architecture* dove l'interazione tra client e server è effettuata tramite delle invocazioni a procedure remote ma ci si riferisce a un nuovo paradigma architetturale che pone al centro dell'attenzione non più i "Servizi" ma le "Risorse" dando vita così a un'architettura *ROA - Resource Oriented Architecture*, ovvero un'architettura orientata alle risorse.

La risorsa in effetti, "è qualcosa", quindi l'approccio utilizzato per la sua realizzazione sarà simile a quello relativo al mondo object-oriented in cui il sistema viene diviso in entità che vengono individuate attraverso la presenza di un sostantivo. Per esempio nella programmazione OOP, per ogni uno di questi ("libro", "biblioteca") vengono create delle classi e i rispettivi comportamenti per l'interazione tra le altre classi. A differenza della progettazione di una classe (che può esporre un certo numero di metodi e attribuire a essi dei nomi), una risorsa deve attenersi ad una ben definita interfaccia che fa riferimento ai quattro metodi più importanti di http (GET, PUT, POST,

DELETE). Su di essa quindi saranno possibili solo operazioni di lettura, di creazione, di modifica e di eliminazione.

Come indicato in figura 4.3, per la progettazione di una risorsa è possibile individuare alcuni passi da seguire che si adattano ad ogni frameworks o linguaggio di programmazione:

- Individuare il servizio;
- Identificare le risorse;
- Per ogni risorsa:
  - Assegnare un URI alla risorsa;
  - Definire i parametri della risorsa;
  - Progettare il tipo di rappresentazione accettato dal client;
  - Progettare i tipi di rappresentazioni in risposta al client;
  - Progettare la risposta in caso di esito positivo o negativo;

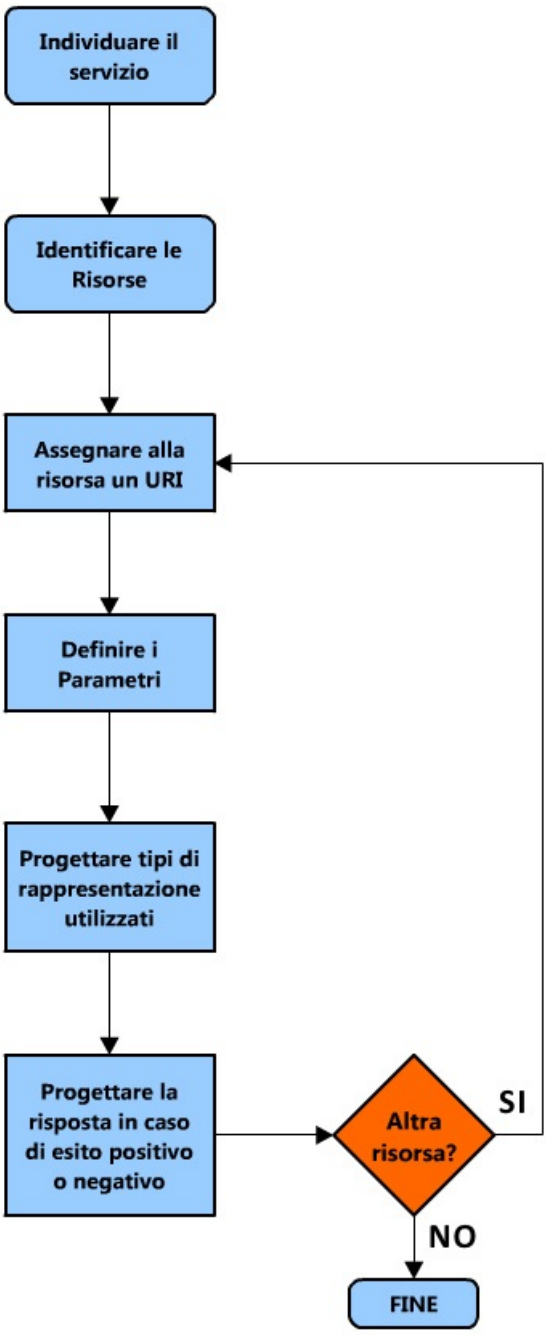

Figura 4.3: Processo di creazione risorsa RESTful.

Individuare il servizio

Questa prima fase si base sull'analisi dei dati con cui l'applicazione dovrà lavorare. Si deve estrarre dai documenti di specifica tutti i dati che l'applicazione dovrà utilizzare e il loro significato nel contesto in modo da individuare i servizi che potranno essere offerti.

Identificare le risorse

In questa fase si devono identificare le risorse da mostrare come servizi (ad esempio "elencoRiferimenti").

Assegnare un URI alla risorsa

Una volta identificate le risorse da implementare all'interno di un servizio, si deve associare a loro un nome. Questo in REST si traduce (in virtù del principio di indicizzazione), nell'assegnare a loro una URI che contenga le informazioni di scope. Esistono diversi pattern per incapsulare le informazioni di scope all'interno di una URI, in particolare:

- Codificando le informazioni di scope all'interno di una forma gerarchica - Questo è il modo migliore di organizzare i dati qualora siano compatibili con un tipo di organizzazione gerarchica. Gli elementi possono essere raggruppati in base alla profondità all'interno dell'ordine di gerarchia, e sono separati dal carattere "/". Esempio: services/{foglia1}/{foglia2}.

- Utilizzando delle forme di punteggiatura - questo per evitare l'utilizzo di gerarchie quando non esistono. Quando i dati di scope non siano compatibili con un'organizzazione gerarchica, allora si può pensare di inserire dei simboli di punteggiatura per separare le variabili. Si pensi ad esempio, un servizio che richiede in ingresso le coordinate di un piano di riferimento. Non ha senso mettere queste due informazioni in forma gerarchica, perché hanno lo stesso livello. Per separare le variabili si

possono utilizzare caratteri come la virgola o il punto e virgola. In particolare, viene consigliato l'utilizzo della virgola nel caso in cui la posizione delle due variabili è influente e il punto e virgola nel caso contrario.

- Inserire le informazioni all'interno della querystring - questa soluzione viene adottata frequentemente nel caso in cui la risorsa a cui si passano i dati, li utilizza per eseguire un algoritmo. In questo caso l'URL risulta essere: http://nilde.bo.cnr.it/services/elencoriferimenti?parm1=valore1&parm2=parm2

Definire i parametri della risorsa

In questa fase si deve identificare i parametri da definire per una risorsa in base a quali operazioni concedere sulla risorsa stessa. Bisogna inoltre specificare quali saranno i parametri obbligatori e quali quelli opzionali.

Progettare il tipo di rappresentazione accettato dal client

A questo punto si deve decidere quali dati mandare a un client che richiede la risorsa e quale formato utilizzare. Si potrebbe pensare di costruirne una rappresentazione, magari elencando gli elementi in plain text, che comporterebbe però alla realizzazione di un parser personalizzato ed inoltre è impensabile da realizzare anche solo per strutture poco complesse.

Un altro modo di rappresentare le risorse è attraverso XML che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo, assicurando piena indipendenza dal tipo di ambiente di sviluppo che si utilizza. L'utilizzo di questo linguaggio per la rappresentazione di risorse, è reso comodo dall'ottimo supporto presente praticamente per ogni linguaggio di programmazione.

Progettare i tipi di rappresentazioni in risposta al client

Per le risorse di tipo read/write, si rende necessario inviare lo stato di una risorsa dal client al server per aggiornare lo stato della stessa. Tal-

volta il problema può essere evitato perché ciò che occorre è già all'interno dell'URL. Ad esempio se si vuole esporre l'oggetto "libro" e si vuole memorizzare solo l'attributo nome si può pensare di creare la nuova risorsa semplicemente effettuando una richiesta di tipo POST all'indirizzo http://nilde.bo.cnr.it/services/rest/libro/{nome_autore}. Quando la risorsa però, deve essere rappresentata mediante più di un elemento, si rende necessario l'utilizzo del body entity al fine di trasmettere gli elementi che la identificano. A questo punto sorge il problema di definire in che modo i dati devono essere formattati affinché il server possa interpretarli nel modo migliore. Nel caso in cui la rappresentazione sia complessa, potrebbe essere utile l'impiego dello stesso formato utilizzato dal server per inviare dati al client, poiché il cliente potrebbe richiedere la rappresentazione mediante una richiesta GET, manipolarla al fine di effettuare su di essa le modifiche necessarie e infine rispedirla al server per l'aggiornamento.

Un'altra tecnica potrebbe essere quella del Form-encoding come formato di rappresentazione che viene largamente utilizzata per l'invio dei dati di form nel mondo delle web application. Un esempio di form che utilizza questo metodo potrebbe essere:

```
<form action="http://www.nilde.bo.cnr.it/risorsaEsempio"
method="post"
enctype="application/x-www-form-urlencoded">
<input type="text" name="nome" value="valore1">
<input type="text" name="autore" value="valore1">
</form>
```

In questo caso, il form, è composto da due caselle di testo "nome" e "autore", che hanno valori di default rispettivamente "valore1" e "valore2". Se l'utente sottomettesse il form con i valori di default, il browser farà partire una richiesta all'indirizzo http://www.nilde.bo.cnr.it/risorsaEsempio con questo entity body: nome=valore1&autore=valore2.

Progettare la risposta in caso di esito positivo o negativo

In questa fase bisogna definire in che modo i client si accorgono che la loro richiesta è stata accolta positivamente oppure no. HTTP mette a disposizione un modo per rispondere alle richieste del client in maniera sintetica ma molto precisa. Infatti definisce una vasta gamma di codici di risposta che sono utilizzabili dal server per rendere noto al client lo stato HTTP. Questo codice viene trasmesso nell'header della risposta. Ad esempio, quando si richiede una pagina attraverso un browser, se esiste e non si verificano errori il server invierà un codice di risposta 200 (OK). Il browser capirà che la sua richiesta è stata accolta e che nell'entity body ci sarà il contenuto della pagina da interpretare. L'utilizzo di REST non pone alcun vincolo sul tipo ti risposta da utilizzare sia in caso di successo della richiesta sia in caso di fallimento. Infatti un servizio web che si basa su un'architettura REST ha la massima libertà per impiegare qualunque codice che HTTP mette a disposizione per rispondere al client nella maniera più idonea.

4.4.1 Best Practices

In questo paragrafo verranno esposti una serie di consigli utili da considerare in fase di progettazione e sviluppo di servizi web REST.

4.4.1.1 Design delle URI

In fase di progettazione delle URI è necessario seguire alcune linee guida:

- usare nomi (cosa sono) e non verbi (cosa fanno) - Cattivo esempio: http://nilde.bo.cnr.it/scegliriferimento; Buon esempio: http://nilde.bo.cnr.it/riferimento;
- non utilizzare le URI per passare i parametri attraverso una GET: in un'architettura RESTful le GET dovrebbero essere usate esclusivamente per operazioni SAFE (vedi paragrafo 4.4.1.2);
- usare solo lettere minuscole;

- le risorse dovrebbero avere un nome plurale, gli oggetti singolare;
- scegliere una struttura chiara, intuitiva e facilmente comprensibile;
- quando si progetta la struttura URI bisogna tenere conto che l'obiettivo è non cambiarla più. REST infatti introduce un accoppiamento molto forte tra client e server (ad esempio incoraggia l'utilizzo di Segnalibri).

4.4.1.2 Operazioni SAFE e UNSAFE

Le operazione SAFE, sono quelle operazioni definite idempotenti cioè che non modificano lo stato del server come ad esempio il metodo HTTP GET; mentre le UNSAFE come il metodo HTTP POST sono l'esatto contrario. Infatti il metodo POST può essere usato in scrittura, quindi va trattata con attenzione perché potrebbe cambiare lo stato della risorsa con effetti indesiderati.

Ad esempio, se vogliamo incrementare il saldo di un conto corrente di 200€ utilizzando una POST e qualcosa va storto nella comunicazione (ad esempio il server non risponde), ripetere l'operazione potrebbe depositare 400€ invece della somma corretta. Utilizzando operazioni Safe non si correrebbe questo rischio perché la loro ripetizione non ha effetti collaterali sulla risorsa:

- la GET è un'operazione di sola lettura, quindi ripetendola più volte si ottiene una lettura ripetuta della risorsa;
- la PUT è un'operazione di aggiornamento, quindi ripetendola più volte si otterrà una sovrascrittura dello stato della risorsa sempre con lo stesso valore;
- la DELETE è un'operazione di eliminazione, quindi se la risorsa è già stata cancellata non avrà alcun effetto, altrimenti la eliminerà.

Quindi per limitare i potenziali danni dell'esecuzione di un'operazione UNSAFE è opportuno riformularla sotto forma di una serie di operazioni SAFE.

Ad esempio, si potrebbe trasformare l'operazione Deposita 200€ in questo modo:

1. Si usa una GET (operazione SAFE) per leggere il saldo attuale s = GetSaldo();

2. Si esegue la somma in locale s = s + 200€;

3. Si usa PUT (operazione idempotente) per aggiornare il saldo corrente SetSaldo(s);

4.4.1.3 CORS - Cross Origin Resource Sharing

Uno scenario ricorrente nei servizi web è quello in cui un Consumer, mediante JavaScript e AJAX (Asynchronous Javascript and XML), accede a dei dati consumando un servizio REST. Nella programmazione Javascript utilizzando la tecnica AJAX esiste un limite che riguarda l'accesso ai dati esterni che non fanno parte dello stesso dominio. Per questo motivo, per attenuare i rischi derivanti da attacchi di tipo "Cross Site Scripting" o "Man in the middle", i browser applicano una restrizione, detta *same-origin policy*, per assicurare un certo livello di sicurezza ovvero nella negazione di specificare URL esterni cioè nell'impossibilità di accesso ai dati esposti da un servizio REST da parte di un'applicazione web esposta su un dominio diverso. La soluzione definitiva a questo problema è il *Cross Origin Resource Sharing* (CORS), e si tratta di un meccanismo che permette al server di distinguere la provenienza di una richiesta e decidere se servirla. In pratica, permette di far funzionare la chiamata AJAX per qualsiasi dominio che si vuole specificare come affidabile.

CORS è una specifica W3C ed è supportato dai seguenti browser: Chrome 3+, Firefox 3.5+, Opera 12+, Safari 4+ e Internet Explorer 8+.

Nel paragrafo 4.6 verrà spiegato nel dettaglio come è stato implementato CORS in NILDE.

4.5 Progettazione servizi web in NILDE

Come specificiato nel paragrafo 4.1, le API di NILDE sono state suddivise in due categorie API Pubbliche e API Private. Per entrambi è obbligatorio specificare l'API key e il Secret e in aggiunta, per le API private è richiesta obbligatoriamente l'autenticazione che può avvenire o tramite form di login (con OAuth) oppure tramite la funzionalità Direct Login (vedi paragrafo 4.1). La funzionalità di Direct Login quindi può essere applicata a tutte le API private specificando username e password al momento della chiamata API:

Esempio richiesta con form di login:

http://nilde.bo.cnr.it/rest/nomeservizio/APIKEY/SECRET/

Esempio richiesta directlogin:

http://nilde.bo.cnr.it/rest/nomeservizio/directlogin/APIKEY/SECRET/username/passwd/

Per le API pubbliche invece non è richiesta l'autenticazione quindi basterà specificare l'API key e il Secret:

Esempio richiesta API pubblica:

http://nilde.bo.cnr.it/rest/nomeservizio/APIKEY/SECRET/

Le risposte del server verso il client vengono gestite tramite la classe XMLCreator, la quale si occupa della creazione dell'XML e degli header HTTP in caso di esito positivo o in caso di esito negativo:

```
//CREAZIONE XML IN CASO DI ERRORE.
public static function xmlError($errorCode, $responseError) {
        $xml_error  = "<?xml version=\"1.0\" encoding=\"utf-8\"?>";
        $xml_error .= "<response status=\"error\">";
        $xml_error .= "<error code=\"$errorCode\">$responseError</error>";
        $xml_error .= "</response>";
        return $xml_error;
}
//CREAZIONE XML IN CASO DI RISPOSTA OK.
public static function xmlOk($responseXML) {
        $xml_ok  = "<?xml version=\"1.0\" encoding=\"utf-8\"?>";
        $xml_ok .= "<response status=\"ok\">";
        $xml_ok .= "<content>$responseXML</content>";
        $xml_ok .= "</response>";
    return $xml_ok;
}
//INVIO E CREAZIONE HEADER HTTP IN CASO DI ERRORE
public static function sendXMLError($status, $response) {
```

```
        header('HTTP/1.1 '.XMLCreator::getMessageForCode($status));
        header('Content-Type: text/xml');
        echo XMLCreator::xmlError($status, $response);
}
//INVIO E CREAZIONE HEADER HTTP IN CASO OK
public static function sendXMLOk($status, $response) {
        header('HTTP/1.1 '.XMLCreator::getMessageForCode($status));
        header('Content-Type: text/xml');
        echo XMLCreator::xmlOk($response);
}
//FUNZIONE CHE RESTITUISCE IL MESSAGGIO HTTP CORRISPONDENTE
//AL CODICE INDICATO.
public static function getMessageForCode($status)     {
        if (isset(self::$messages[$status])) {
                return self::$messages[$status];
        } else {
                return null;
        }
}
```

Da come si può intuire il codice è abbastanza semplice. In caso di errore verrà richiamata la funzione xmlError() alla quale verranno passati come parametri lo stato HTTP e il relativo messaggio. Questa funzione si occupa quindi di generare l'XML che sarà mostrato a video al client in caso di errore. La funzione xmlOk() invece si occupa della creazione dell'XML in caso di esito positivo della chiamata e in questo caso l'unico parametro passato è il contenuto della risposta che sarà mostrato a video al client.

Le altre due funzioni sendXMLError() e sendXMLOk() vengono utilizzate rispettivamente per la creazione degli header HTTP in caso di errore o in caso di esito positivo e della stampa del corrispondente messaggio XML. Infine il metodo statico getMessageForCode() si occupa di restituire il messaggio HTTP corrispondente al codice di stato indicato come parametro.

Esempio risposta in caso di esito positivo con stato **201**: *Created*:

```
<?xml version="1.0" encoding="utf-8"?>
<response status="created">
        <content>Resource created</content>
</response>
```

Header HTTP ricevuto in risposta dal server alla chiamata API:

```
Date: Mon, 28 Oct 2013 13:26:28 GMT
Server: Apache/2.2.16 (Debian)
X-Powered-By: PHP/5.3.3-7+squeeze15
Access-Control-Allow-Methods: POST, GET, OPTIONS, HEAD, PUT
Access-Control-Max-Age: 1000
Access-Control-Allow-Headers: Content-Type
Access-Control-Allow-Credentials: true
Content-Length: 522
Keep-Alive: timeout=15, max=95
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8
201 Created
```

Esempio risposta in caso di esito negativo con stato **403**: *Forbidden*:

```
<?xml version="1.0" encoding="utf-8"?>
<response status="error">
        <error code=403>
                Wrong combination API Key - Secret.
        </error>
</response>
```

Header HTTP ricevuto in risposta dal server alla chiamata API:

```
Date: Mon, 28 Oct 2013 13:27:50 GMT
Server: Apache/2.2.16 (Debian)
X-Powered-By: PHP/5.3.3-7+squeeze15
Access-Control-Allow-Methods: POST, GET, OPTIONS, HEAD, PUT
Access-Control-Max-Age: 1000
Access-Control-Allow-Headers: Content-Type
Access-Control-Allow-Credentials: true
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Pragma: no-cache
Content-Length: 135
Keep-Alive: timeout=15, max=98
Connection: Keep-Alive
Content-Type: text/xml
403 Forbidden
```

4.5.1 Progettazione di servizi web per l'Utente

In questo paragrafo verranno elencati e definiti tutti i servizi individuati per l'interazione dell'utente con il sistema NILDE. In questo caso, i dati e le risorse con cui si interagisce tramite le API, fanno riferimento a informazioni relative all'utente.

4.5.1.1 Elenco dei riferimenti / richieste

Descrizione: Ritorna la lista dei riferimenti di un utente (eventualmente richiesti alla sua biblioteca).

Risorsa: *ElencoRiferimenti*: L'elenco dei riferimenti viene rappresentato da una risorsa che mostra tutti i riferimenti di un utente.

URI: http://nilde.bo.cnr.it/rest/elencoriferimenti/

Autenticazione: Questo metodo necessita di autenticazione tramite username e password (implementato con OAuth).

Autorizzazione (di lettura): attraverso API Key e Secret.

Tipo di rappresentazione: Documento XML.

Risposta in caso di esito positivo: Essendo un servizio di sola lettura, la risposta al client avrà codice di stato **200**: *OK* indipendentemente dal numero di risultati della ricerca. Il contenuto della risposta (Body) conterrà l'XML contenente l'output della richiesta.

Risposta in caso di esito negativo:

- **403**: *Wrong combination API Key - Secret.* La API Key e il Secret forniti non sono corretti.

- **403**: *Wrong authentication.* Username e Password forniti non sono corretti.

- **404**: *Required arguments missing.* Non sono stati inseriti tutti i parametri obbligatori per effettuare la richiesta.
- **501**: *Method "xxx" not found.* Il metodo richiesto non è stato trovato.
- **503**: *Service currently unavailable.* Il servizio richiesto non è temporaneamente disponibile.

4.5.1.2 Inserimento nuovo riferimento

Descrizione: L'utente attraverso una chiamata API REST potrà inserire all'interno del suo account un nuovo riferimento la cui rappresentazione è data dai dati bibliografici.

Nota: Questo metodo necessita di una richiesta HTTP POST.

Risorsa: *Riferimento*: rappresenta il riferimento bibliografico (articolo, libro) che l'utente intende inserire nel proprio account.

URI: http://nilde.bo.cnr.it/rest/riferimento/

Autenticazione: Questo metodo necessita di autenticazione tramite username e password (implementato con OAuth).

Autorizzazione (di scrittura): attraverso API Key e Secret.

Tipo di rappresentazione: Documento XML e HTML con form di input.

Risposta in caso di esito positivo: la risposta al client avrà codice di stato **200**: *OK* e verrà mostrato il form per inserire i dati relativi al nuovo riferimento.

Esempio:

```
<form action="http://nilde.bo.cnr.it/rest/inserisciriferimento/"
method="post" enctype="application/x-www-form-urlencoded">
        <input type="text" name="titolo" value="">
        <input type="text" name="autore" value="">
        <input name='invia' type='submit' value='invia'>
        <input name='annulla' type='submit' value='annulla'>
</form>
```

Una volta compilato il form e cliccato su invia, dato che si tratta della creazione di una nuova risorsa, il server risponderà al client con il codice di stato **201**: *Created* contenente nel body della risposta l'XML relativo.

Risposta in caso di esito negativo:

- **403**: *Wrong combination API Key - Secret.* La API Key e il Secret forniti non sono corretti.

- **403**: *Wrong authentication.* Username e Password forniti non sono corretti.

- **404**: *Required arguments missing.* Non sono stati inseriti tutti i parametri obbligatori per effettuare la richiesta.

- **501**: *Method "xxx" not found.* Il metodo richiesto non è stato trovato.

- **503**: *Service currently unavailable.* Il servizio richiesto non è temporaneamente disponibile.

4.5.1.3 Richiedi riferimento

Descrizione: Questo servizio permette all'utente di effettuare la richiesta presso una biblioteca di un riferimento (articolo, libro) precedentemente inserito nel suo account.

Nota: Questo metodo necessita di una richiesta **HTTP POST**.

Risorsa: *richiestaRiferimento*: rappresenta il riferimento bibliografico (articolo, libro) che l'utente intende richiedere a una biblioteca.

URI: http://nilde.bo.cnr.it/rest/richiestariferimento/

Autenticazione: Questo metodo necessita di autenticazione tramite username e password (implementato con OAuth).

Autorizzazione (di scrittura): attraverso API Key e Secret.

Parametri:

- *id_riferimento* (Obbligatorio) - ID del riferimento da richiedere.

Tipo di rappresentazione: Documento XML.

Risposta in caso di esito positivo: la risposta al client avrà codice di stato **201**: *Created* contenente nel body della risposta l'XML relativo.

Risposta in caso di esito negativo:

- **403**: *Wrong combination API Key - Secret.* La API Key e il Secret forniti non sono corretti.
- **403**: *Wrong authentication.* Username e Password forniti non sono corretti.
- **404**: *Required arguments missing.* Non sono stati inseriti tutti i parametri obbligatori per effettuare la richiesta.
- **404**: *Reference not found.* Il riferimento indicato nella richiesta non esiste.

- **501**: *Method "xxx" not found.* Il metodo richiesto non è stato trovato.
- **503**: *Service currently unavailable.* Il servizio richiesto non è temporaneamente disponibile.

4.5.1.4 Visualizzazione riferimento / richiesta

Descrizione: Questo servizio permette all'utente di visualizzare le informazioni di un riferimento / richiesta la cui rappresentazione è un riferimento bibliografico.

Risorsa: *VisualizzaRiferimento*: rappresenta il riferimento bibliografico che l'utente intende visualizzare.

URI: http://nilde.bo.cnr.it/rest/visualizzariferimento/

Autenticazione: Questo metodo necessita di autenticazione tramite username e password (implementato con OAuth).

Autorizzazione (di lettura): attraverso API Key e Secret.

Parametri:

- *id_riferimento* (Obbligatorio) - ID del riferimento da richiedere.

Tipo di rappresentazione: Documento XML.

Risposta in caso di esito positivo: Essendo un servizio di sola lettura, la risposta al client avrà codice di stato **200**: *OK*. Il contenuto della risposta (Body) conterrà l'XML contenente l'output della richiesta.

Risposta in caso di esito negativo:

- **403**: *Wrong combination API Key - Secret.* La API Key e il Secret forniti non sono corretti.
- **403**: *Wrong authentication.* Username e Password forniti non sono corretti.

- **404**: *Required arguments missing.* Non sono stati inseriti tutti i parametri obbligatori per effettuare la richiesta.

- **404**: *Reference not found.* Il riferimento o la richiesta indicata non esiste.

- **501**: *Method "xxx" not found.* Il metodo richiesto non è stato trovato.

- **503**: *Service currently unavailable.* Il servizio richiesto non è temporaneamente disponibile.

4.5.1.5 Visualizzazione profilo utente

Descrizione: Questo servizio permette all'utente di visualizzare le informazioni del suo profilo la cui rappresentazione sono i dati anagrafici dell'utante.

Risorsa: *ProfiloUtente*: rappresenta l'insieme delle informazioni anagrafiche di un utente che si intende visualizzare.

URI: http://nilde.bo.cnr.it/rest/profiloutente/

Autenticazione: Questo metodo necessita di autenticazione tramite username e password (implementato con OAuth).

Autorizzazione (di lettura): attraverso API Key e Secret.

Parametri:

- *id_profilo* (Obbligatorio) - ID del profilo da visualizzare.

Tipo di rappresentazione: Documento XML.

Risposta in caso di esito positivo: Essendo un servizio di sola lettura, la risposta al client avrà codice di stato **200**: Ok. Il contenuto della risposta (Body) conterrà l'XML contenente l'output della richiesta (XML con i dati).

Risposta in caso di esito negativo:

- **403**: *Wrong combination API Key - Secret.* La API Key e il Secret forniti non sono corretti.

- **403**: *Wrong authentication.* Username e Password forniti non sono corretti.

- **404**: *Required arguments missing.* Non sono stati inseriti tutti i parametri obbligatori per effettuare la richiesta.

- **404**: *Profile not found.* Il profilo richiesto non esiste.

- **501**: *Method "xxx" not found.* Il metodo richiesto non è stato trovato.

- **503**: *Service currently unavailable.* Il servizio richiesto non è temporaneamente disponibile.

4.5.1.6 Modifica profilo

Descrizione: Questo servizio permette all'utente di modificare le informazioni del suo profilo. In particolare tramite una chiamata API REST sarà possibile modificare gli elementi come ad esempio: nome, cognome, indirizzo, ecc..

Nota: Questo metodo necessita di una richiesta HTTP PUT.

Risorsa: *ProfiloUtente*: rappresenta l'insieme delle informazioni anagrafiche di un utente che si intende modificare.

URI: http://nilde.bo.cnr.it/rest/modificaprofilo/

Autenticazione: Questo metodo necessita di autenticazione tramite username e password (implementato con OAuth).

Autorizzazione (di scrittura): attraverso API Key e Secret.

Tipo di rappresentazione: Documento XML e HTML con form di input.

Risposta in caso di esito positivo: la risposta al client avrà codice di stato **200**: *OK* e verrà mostrato il form per modificare i dati relativi al profilo richiesto.

Esempio:

```
<form action="http://nilde.bo.cnr.it/rest/modificaprofilo/"
method="put" enctype="application/x-www-form-urlencoded">
        <input type="text" name="Nome" value="Vincenzo">
        <input type="text" name="Cognome" value="Alaia">
        <input name='invia' type='submit' value='invia'>
        <input name='annulla' type='submit' value='annulla'>
</form>
```

Una volta modificati i campi del form e cliccato su invia, dato che si tratta della modifica di una risorsa, il server risponderà al client con il codice di stato **205**: Reset Content contenente nel body della risposta l'XML relativo.

```
<?xml version="1.0" encoding="utf-8"?>
<response status="Reset Content">
        <content>
                User profile modified
        </content>
</response>
```

Risposta in caso di esito negativo:

- **403**: *Wrong combination API Key - Secret.* La API Key e il Secret forniti non sono corretti.
- **403**: *Wrong authentication.* Username e Password forniti non sono corretti.
- **404**: *Required arguments missing.* Non sono stati inseriti tutti i parametri obbligatori per effettuare la richiesta.
- **404**: *Profile not found.* Il profilo richiesto non esiste.
- **501**: *Method "xxx" not found.* Il metodo richiesto non è stato trovato.

- **503**: *Service currently unavailable.* Il servizio richiesto non è temporaneamente disponibile.

4.5.2 Progettazione di servizi web per la Biblioteca

In questo paragrafo verranno elencati e definiti tutti i servizi individuati per l'interazione di un Utente con il sistema NILDE. In questo caso, i dati e le risorse con cui si interagisce tramite le API, fanno riferimento a informazioni relative alla Biblioteca.

4.5.2.1 Anagrafica biblioteche

Descrizione: Questo servizio permette all'utente che effettua la richiesta di visualizzare i dati anagrafici delle biblioteche registrate su NILDE. In questo caso bisogna rappresentare il tipo di dato "anagrafica" la cui rappresentazione è data dagli elementi: nome biblioteca, indirizzo, città, ecc..

Risorsa: *AnagraficaBiblio*: rappresenta l'insieme delle informazioni anagrafiche delle biblioteche che si intende visualizzare.

URI: http://nilde.bo.cnr.it/rest/anagraficabiblio/

Autenticazione: Questo metodo non necessita di autenticazione.

Autorizzazione (di lettura): attraverso API Key e Secret.

Tipo di rappresentazione: Documento XML.

Risposta in caso di esito positivo: Essendo un servizio di sola lettura, la risposta al client avrà codice di stato **200**: *OK*. Il contenuto della risposta (Body) conterrà l'XML contenente l'output della richiesta.

Risposta in caso di esito negativo:

- **403**: *Wrong combination API Key - Secret.* La API Key e il Secret forniti non sono corretti.

- **404**: *Required arguments missing.* Non sono stati inseriti tutti i parametri obbligatori per effettuare la richiesta.
- **501**: *Method "xxx" not found.* Il metodo richiesto non è stato trovato.
- **503**: *Service currently unavailable.* Il servizio richiesto non è temporaneamente disponibile.

4.5.2.2 Dati anagrafici di una biblioteca

Descrizione: Questo servizio permette all'utente che effettua la richiesta di visualizzare i dati anagrafici di una biblioteca specifica registrata su NILDE.

Risorsa: *AnagraficaBiblio*: rappresenta l'insieme delle informazioni anagrafiche della biblioteca che si intende visualizzare.

URI: http://nilde.bo.cnr.it/rest/anagraficabiblio/IDbiblioteca

Autenticazione: Questo metodo non necessita di autenticazione.

Autorizzazione (di lettura): attraverso API Key e Secret.

Parametri:

- ID_biblioteca (Obbligatorio) - ID della biblioteca per la quale si intende visualizzare i dati anagrafici.

Tipo di rappresentazione: Documento XML.

Risposta in caso di esito positivo: Essendo un servizio di sola lettura, la risposta al client avrà codice di stato **200**: *OK*. Il contenuto della risposta (Body) conterrà l'XML contenente l'output della richiesta.

Risposta in caso di esito negativo:

- **403**: *Wrong combination API Key - Secret.* La API Key e il Secret forniti non sono corretti.

- **404**: *Required arguments missing.* Non sono stati inseriti tutti i parametri obbligatori per effettuare la richiesta.
- **404**: *Library not found.* La biblioteca richiesta non esiste.
- **501**: *Method "xxx" not found.* Il metodo richiesto non è stato trovato.
- **503**: *Service currently unavailable.* Il servizio richiesto non è temporaneamente disponibile.

4.5.2.3 Aggiornamento data sospensione servizio DD tramite codice ACNP

Descrizione: Questo servizio permette al server ACNP di modificare / aggiornare la data di sospensione del servizio di Document Delivery di una biblioteca tramite il codice ACNP mantenendo sincronizzati i dati relativi alla sospensione del servizio tra NILDE e ACNP.

Nota: Questo metodo necessita di una richiesta **HTTP PUT**.

Risorsa: *DataSospensioneDD*: rappresenta la data di sospensione del servizio di DD che una biblioteca vuole aggiornare.

URI: http://nilde.bo.cnr.it/rest/datasopensionedd/

Autenticazione: Questo metodo non necessita di autenticazione.

Autorizzazione (di scrittura): attraverso API Key e Secret.

Parametri:

- *from_data* (Obbligatorio) - data inizio sospensione servizio ILL / DD in formato "GG-MM-AA".
- *to_data* (Obbligatorio) - data fine sospensione servizio ILL / DD in formato "GG-MM-AA".
- *cod_ACNP* (Obbligatorio) - codice ACNP della biblioteca per la quale si intende sospendere il servizio di ILL / DD.

Tipo di rappresentazione: Documento XML.

Risposta in caso di esito positivo: Essendo un servizio di aggiornamento di un'informazione, il server risponderà al client con il codice di stato **205**: *Reset Content* contenente nel body della risposta l'XML relativo.

Risposta in caso di esito negativo:

- **400**: *Wrong date format.* Il formato della data è errato.

- **403**: *Wrong combination API Key - Secret.* La API Key e il Secret forniti non sono corretti.

- **404**: *Required arguments missing.* Non sono stati inseriti tutti i parametri obbligatori per effettuare la richiesta.

- **404**: *ACNP code not found.* Il codice ACNP della biblioteca richiesta non esiste.

- **501**: *Method "xxx" not found.* Il metodo richiesto non è stato trovato.

- **503**: *Service currently unavailable.* Il servizio richiesto non è temporaneamente disponibile.

4.5.2.4 Visualizzazione profilo biblioteca

Descrizione: Questo servizio permette all'utente di richiedere e visualizzare le informazioni del profilo della biblioteca la cui rappresentazione è data dagli elementi: nome, p.iva, indirizzo, ecc..

Risorsa: *ProfiloBiblio*: rappresenta l'insieme delle informazioni anagrafiche della biblioteca che si intende visualizzare.

URI: http://nilde.bo.cnr.it/rest/profilobiblio/

Autenticazione: Questo metodo necessita di autenticazione tramite username e password (implementato con OAuth).

Autorizzazione (di lettura): attraverso API Key e Secret.

Tipo di rappresentazione: Documento XML.

Risposta in caso di esito positivo: Essendo un servizio di sola lettura, la risposta al client avrà codice di stato **200**: *OK*. Il contenuto della risposta (Body) conterrà l'XML contenente l'output della richiesta.

Risposta in caso di esito negativo:

- **403**: *Wrong combination API Key - Secret.* La API Key e il Secret forniti non sono corretti.
- **403**: *Wrong authentication.* Username e Password forniti non sono corretti.
- **404**: *Required arguments missing.* Non sono stati inseriti tutti i parametri obbligatori per effettuare la richiesta.
- **501**: *Method "xxx" not found.* Il metodo richiesto non è stato trovato.
- **503**: *Service currently unavailable.* Il servizio richiesto non è temporaneamente disponibile.

4.5.2.5 Modifica Profilo biblioteca

Descrizione: Questo servizio permette all'utente di modificare le informazioni di profilo di una biblioteca. In particolare tramite una chiamata API REST sarà possibile modificare gli elementi come ad esempio: nome, p.iva, indirizzo, ecc..

Nota: Questo metodo necessita di una richiesta **HTTP PUT**.

Risorsa: *ModificaProfiloBiblio*: rappresenta l'insieme delle informazioni anagrafiche di una biblioteca che si intende modificare.

URI: http://nilde.bo.cnr.it/rest/modificaprofilobiblio/

Autenticazione: Questo metodo necessita di autenticazione tramite username e password (implementato con OAuth).

Autorizzazione (di scrittura): attraverso API Key e Secret.

Tipo di rappresentazione: Documento XML e HTML con form di input.

Risposta in caso di esito positivo: la risposta al client avrà codice di stato **200**: *OK* e verrà mostrato il form per modificare i dati relativi alla biblioteca richiesta.

Esempio:

```
<form action="http://nilde.bo.cnr.it/rest/modificaprofilo/"
method="put" enctype="application/x-www-form-urlencoded">
        <input type="text" name="Name" value="Biblioteca 1">
        <input type="text" name="Adress" value="via scappascappa">
        <input name='invia' type='submit' value='invia'>
        <input name='annulla' type='submit' value='annulla'>
</form>
```

Una volta modificati i campi del form e cliccato su invia, dato che si tratta della modifica di una risorsa, il server risponderà al client con il codice di stato **205**: *Reset Content* contenente nel body della risposta l'XML relativo.

Risposta in caso di esito negativo:

- **403**: *Wrong combination API Key - Secret.* La API Key e il Secret forniti non sono corretti.
- **403**: *Wrong authentication.* Username e Password forniti non sono corretti.
- **404**: *Required arguments missing.* Non sono stati inseriti tutti i parametri obbligatori per effettuare la richiesta.
- **501**: *Method "xxx" not found.* Il metodo richiesto non è stato trovato.
- **503**: *Service currently unavailable.* Il servizio richiesto non è temporaneamente disponibile.

4.6 Sviluppo CORS in NILDE

Come spiegato nel paragrafo 4.4.1.3 in uno scenario come questo, si è dovuto sviluppare un meccanismo che permettesse a domini esterni di accedere ai dati presenti sul dominio nilde.bo.cnr.it effettuando chiamate Ajax. Questo è stato sviluppato implementando CORS e sfruttando il Framework Prototype JavaScript per effettuare le chiamate AJAX. PrototypeJS è un framework JavaScript che facilita lo sviluppo di applicazioni web dinamiche e in particolare offre un supporto per l'utilizzo di AJAX e della programmazione orientata agli oggetti in JavaScript.

L'implementazione di CORS consiste nell'aggiunta di Header HTTP tra le richieste e le risposte HTTP cross-domain.

In particolare sul server sono stati specificati i seguenti Header:

```
if (isset($_SERVER['HTTP_ORIGIN'])){
        $domainURL = $_SERVER['HTTP_ORIGIN'];
} else if (isset($_SERVER['HTTP_REFERER'])){
        $url = $_SERVER['HTTP_REFERER'];
        $parse = parse_url($url);
        $domainURL = $parse['scheme'].'://'.$parse['host'];
}
```

```
header('Access-Control-Allow-Origin: '.$domainURL);
header('Access-Control-Allow-Methods: POST, GET, OPTIONS, HEAD, PUT');
header('Access-Control-Allow-Headers: Content-Type');
header('Access-Control-Allow-Credentials: true');
```

Il primo header definisce da quali domini verranno accettate le richieste. In questo caso vengono utilizzate le variabili globali $_SERVER['HTTP_ORIGIN'] e $_SERVER['HTTP_REFERER'] a seconda di quale delle due è definita. In sostanza si permette a qualsiasi dominio esterno di poter effettuare chiamate REST in cross-domain utilizzando AJAX. In questo modo si abilita solo il chiamante se gli header sono passati. Il secondo header indica quali operazioni HTTP sono permesse e in questo caso verranno accettate le operazioni POST, GET, PUT, OPTIONS e HEAD. Il terzo header indica quali header HTTP possono essere utilizzati quando si effettua la richiesta mentre la definizione dell'ultimo header indica se la richiesta può essere fatta utilizzando le credenziali per l'autenticazione HTTP. Quest'ultimo header è stato specificato perchè di defaul lo standard CORS non invia i cookies. Quindi per includere i cookie come parte della richiesta è necessario impostare (secondo le specifiche CORS) il parametro withCredentials di XMLHttpRequest a true e specificare sul server header('Access-Control-Allow-Credentials: true'). In questo modo saranno inclusi i cookies nelle richieste dal dominio remoto e nelle risposta dal server.

In un'architettura REST ben progettata i cookie non dovrebbero essere né presenti, né tanto meno necessari. In primo luogo per una coerenza di design: le chiamate devono essere stateless, quindi dovrebbero contenere al loro interno tutte le informazioni necessarie perché il server le processi. Poiché i cookie per definizione sono informazioni esterne alle chiamate, rappresenterebbero un'eccezione allo stile architetturale i quali vengono usati per la gestione della sessione in cui viene passato l'ID di sessione di PHP che viene trasmesso nei cookie.

4.7 Sviluppo API REST in NILDE

Come detto all'inizio di questo capitolo, il framework utilizzato per lo sviluppo delle API in REST è Slim in quale ha permesso in poco tempo lo sviluppo dei servizi di NILDE. Una importate modifica che è stata effettuata al framework è stata quella di modificare la gestione degli errori nel caso di parametri obbligatori mancanti in fase di chiamata al servizio REST. Infatti il framework di default gestiva questo tipo di errore mostrando a video un semplice messaggio "404 Page Not Found" con codice di stato HTTP 404 Not Found. Tramite un override della funzione handleError è stato possibile gestire questo tipo di errori diversamente:

```
function handleError() {
$app = \Slim\Slim::getInstance();
$app->contentType("text/xml");
$app->halt(404, XMLCreator::xmlError
                  (404,"Required arguments missing."));
}
```

Tramite il metodo statico xmlError della classe XMLCreator (vedi paragrafo 4.5), viene generato il seguente XML che verrà mostrato a video nel caso non viene specificato un parametro obbligatorio:

```
<?xml version="1.0" encoding="utf-8"?>
<response status="error">
        <error code=404>
                Required arguments missing.
        </error>
</response>
```

Slim è stato utilizzato per la definizione delle chiamate ai servizi REST di NILDE che vengono definiti all'interno di un file php il quale è il primo file ad essere richiamato per effettuare i match tra i servizi definiti e la chiamata API effettuata. Per comprendere meglio il funzionamento e l'utilizzo del framework Slim è utile vedere come i metodi sono stati definiti.

Un'esempio potrebbe essere quello del servizio "Aggiornamento data sospensione servizio DD tramite codice ACNP" (Vedi paragrafo 4.5.2.3) la cui definizione in Slim è la seguente:

```
$app = new \Slim\Slim();

$app->put('/datasospensionedd/noauth/:key/:secret/:fromdata/:todata/:codACNP',
function ($key, $secret, $fromdata, $todata, $codACNP) use ($app){
global $requestTokenURL;
global $ajax;
$parameters=base64_encode('&serviceName='.'datasospensionedd'.
        '&fromdata='.$fromdata.'&todata='.$todata.'&codACNP='.$codACNP);
$app->redirect($requestTokenURL.'?consumerKey='.$key.'&consumerSecret='.
$secret.'&noauth=1'.'&ajax='.$ajax.'&par='.$parameters, 301);
});
```

Come prima cosa viene istaziato l'oggetto $app sul quale viene definito subito dopo il metodo HTTP PUT. Il primo argomento del metodo "datasospensionedd" indica la risorsa mentre il secondo argomento "noauth" indica che per questo metodo non è prevista l'autenticazione. Gli altri argomenti definiti con i ":" sono i parametri specifici del metodo che l'utente dovrà specificare obbligatoriamente in fase di chiamata dell'API. La variabile globale $requestTokenURL indica l'URL di riferimento al primo endpoint di OAuth (Request Token Endpoint, vedi paragrafo 4.2.2) dove verranno effettuati i controlli sulla validità della API Key e Secret forniti e verrà scaturito il flusso di autorizzazione OAuth.

La variabile globale $ajax invece indica se l'invocazione del servizio è partita da una chiamata Ajax oppure no. Questa variabile viene impostata in automatico a '1' se si effettua una chiamata AJAX. Per effettuare le chiamate REST in AJAX è stato utilizzato il framework PrototypeJS al quale è stato effettuato l'override del metodo request per abilitare il cross domain e l'utilizzo dei cookie nelle chiamate AJAX. Il codice sorgente di questa modifica si trova nel file nildeapi.js (https://nilde.bo.cnr.it/scripts/nildeapi.js).

```
//Funzione che permette di importare un ulteriore file javascript
//in questo caso viene importato il file prototype.js
function IncludeJS(jsFile) {
        document.write('<script type="text/javascript" src="'+ jsFile + '"></script>');
}
```

```
function nilde_call(mth, url){
        var AjaxCross = Class.create(Ajax.Request, {
                request: function($super, url){
                this.transport.withCredentials = true;
                $super(url);
        }
});

if (mth=='GET'){
        req=new AjaxCross(url,  {
        method:mth,
        parameters: { ajax: 1 },
        onCreate: function(response) {
        // here comes the fix
        var t = response.transport;
        t.setRequestHeader = t.setRequestHeader.wrap(function(original, k, v) {
        if (/^(accept|accept-language|content-language)$/i.test(k))
                return original(k, v);
        if (/^content-type$/i.test(k) &&
/^(application\/x-www-form-urlencoded|multipart\/form-data|text\/plain)(;.+)?$/i.test(v))
                return original(k, v);
                return;
        });                     },
        onComplete: function (transport) {
                $('nildeRest').update(transport.responseText)
        }
});
        }else {
        var param=$('nildeForm').serialize()+"&ajax=1";
        req=new AjaxCross(url,  {
                method:mth,
                onCreate: function(response) {// here comes the fix
                var t = response.transport;
                t.setRequestHeader = t.setRequestHeader.wrap(function(original, k, v) {
                if (/^(accept|accept-language|content-language)$/i.test(k))
                        return original(k, v);
                if (/^content-type$/i.test(k) &&
/^(application\/x-www-form-urlencoded|multipart\/form-data|text\/plain)(;.+)?$/i.test(v))
                return original(k, v);
                return;
        });
},
        parameters: param,
        onComplete: function (transport) {
        $('nildeRest').update(transport.responseText)
        }
```

```
        });
}
}
IncludeJS.call('prototype.js');
```

La funzione IncludeJS(), permette di importare un ulteriore file javascript in questo caso viene importato il file prototype.js attraverso la funzione document.write() di prototype. Con la funzione nilde_call() invece viene effettuato l'override del metodo Ajax.Request di prototypejs per abilitare il cross domain e l'utilizzo dei cookie nelle chiamate ajax; infatti viene specificato il parametro withCredentials = true. L'if serve a distinguere se la chiamata all'API REST è una GET oppure una POST. In caso di una GET, viene istanziato un nuovo oggetto di tipo AjaxCross e viene settato il parametro ajax=1 (parameters: { ajax: 1 }). Subito dopo la callback onCreate, è stato aggiunto un fix il quale permette il trasporto degli header Accept e Content type che altrimenti non verrebbero trasportati da PrototypeJS in Cross Domain. Nella callback onComplete invece viene specificato che dovrà essere aggiornato il contenitore "nildeRest" con il risultato della chiamata API.

Nell'else invece è presente il codice che verrà eseguito quando si effettua una chiamata tramite POST. In questo caso infatti viene serializzato l'action del form "nildeForm" e viene aggiunto il parametro ajax=1 per indicare che l'invocazione del servizio è partita da una chiamata Ajax. Anche in questo caso dopo aver istanziato un oggetto AjaxCross viene effettuato il fix per permettere il trasporto degli header Accept e Content type e nella callback onComplete viene aggiornato il contenuto del contenitore "nildeRest" tramite la funzione update() di prototype.

Qualora si volesse far uso di Ajax per utilizzare le API di NILDE sarà necessario importare il file nildeapi.js e inserire il link alla chiamata API REST all'interno di un contenitore specifico che per semplicità è stato definito attraverso un div id="nildeRest":

```
<script language="javascript"
        src="https://nilde.bo.cnr.it/scripts/nildeapi.js"> </script>

<div id="nildeRest" name="nildeRest">
<a href="javascript:nilde_call('PUT','https://nilde.bo.cnr.it/
rest/datasospensionedd/5d2d16c9415bdbbafcf2f530db1e06c29d28f5d7/
c309909d3c2af3ef06ef6edd94af7077c8c6dbb7/12-05-2013/25-05-2013/BOXXX')">
Aggiorna data sospensione in NILDE</a> </div>
```

In questo caso verrà effettuata una chiamata Ajax che andrà ad effettuare una modifica della data di sospensione della biblioteca presente in NILDE con codice ACNP BOXXX. Il risultato di questa chiamata che sarà in formato XML verrà restituito all'interno del div stesso, in questo modo si evita di effettuare un refresh della pagina dalla quale è partita la chiamata all'API.

Tutte le chiamate alle API REST di NILDE (Ajax e non), si concludo nel Protected Resource Endpoint (Vedi paragrafo 4.3.2.4) in cui è presente uno switch case che in base al nome del metodo e ai parametri passati, chiama la funzione appropriata per restituire o modificare i dati desiderati.

Conclusioni

Il principale motivo della grande diffusione, in questi ultimi anni dei servizi REST è data dalla sua semplicità di utilizzo e sul fatto che utilizza il protocollo HTTP. Per realizzare un servizio REST non c'e bisogno di sapere come costruire un file WSDL o come vengono scambiati i messaggi SOAP, bisogna solo rispettare alcuni vincoli. Infatti il lavoro svolto si è basato sopratutto sulla comprensione dei principi cardine descritti da Roy Thomas Fielding[20] nella sua tesi di dottorato dove introduce per la prima volta REST. I servizi web che si basano sui principi di design REST non sono regolamentati da rigide specifiche e questo può portare al cambiamento e aggiornamento continuo di utilizzo. Basti pensare alla continua evoluzione delle tecnologie e delle piattaforme come ad esempio le piattaforme cloud che possono far passare questo paradigma di programmazione da un semplice uso per l'integrazione e la comunicazione di più applicazioni ad uno stile in cui un applicazione è costituita interamente da più servizi cooperanti.

Si pensi ad esempio a una applicazione che risiede su un server ed espone alcune sue funzionalità come servizi attraverso i quali un client può svolgere il proprio lavoro. In questo modo il server ha il solito compito di gestire le informazioni e viene lasciato al client la possibilità di creare un proprio programma che gestirà le risorse come meglio crede. Questo da una flessibilità notevole lato client diminuendo il traffico e il lavoro lato server in quanto il cliente ad ogni richiesta non chiederà informazioni su come visualizzare i dati ma richiederà solo i dati.

L'obiettivo raggiunto con questo lavoro di tesi è stato infatti quello di estendere l'architettura del software NILDE attraverso un processo di migrazione verso servizi REST utilizzando e ampliando metodologie, best practice e frameworks che hanno permesso lo sviluppo di API utilizzabili da utenti esterni. L'architettura REST sviluppata risulta sufficientemente flessibile e i servizi sviluppati possono essere utilizzati da qualunque piattaforma.

Per quanto riguarda gli sviluppi futuri questo panorama si potrebbe estendere sviluppando un sito o un'applicazione mobile di NILDE che potrebbe fornire gli attuali servizi utilizzando appunto le API sviluppate in modo da creare un software lato client più leggero e fruibile anche da dispositivi mobili quali smartphone e tablet. Un'ulteriore implementazione potrebbe essere quella di sfruttare le API per permette l'integrazione dei servizi di NILDE con i social network.

Ringraziamenti

Desidero ringraziare tutti gli amici dal primo all'ultimo livello comprese tutte quelle persone con cui ho iniziato e trascorso questi anni a Bologna, con cui ho scambiato pensieri, idee, e giornate di sano sport; direttamente o indirettamente mi avete aiutato a credere in me stesso e a raggiungere i miei obiettivi.

Un ringraziamento particolare va al personale della biblioteca del CNR in particolare ad Alessandro Tugnoli il quale con pazienza e grande professionalità mi ha seguito e aiutato nella realizzazione di questa tesi.

Bibliografia

[1] Baldoni R., "Middleware," Http://www.dis.uniroma1.it/ baldoni/SOII-middleware.pdf, 7 Settembre 2013

[2] W3C - World Wide Web Consortium, http://www.w3.org/ - 10-28-2013

[3] Flickr, http://www.flickr.com/ - 10-28-2013

[4] Fielding R., et al., *Hypertext Transfer Protocol – HTTP/1.1, RFC 2616*, IETF - Internet Engineering Task Force, June 1999, http://tools.ietf.org/html/rfc2616

[5] OASIS - Organization for the Advancement of Structured Information Standards, https://www.oasis-open.org/ - 10-28-2013

[6] IETF - Internet Engineering Task Force, http://www.ietf.org/ - 10-28-2013

[7] Richardson L., Ruby S., *RESTful Web Services*, O'Reilly, 2007

[8] Gudgin M., et al., *SOAP, W3C Recommendation*, W3C - Word Wide Web Consortium, 27 April 2007

[9] Tugnoli A., "Progetto e realizzazione di un service provider Open Archives," Universita' di Bologna, 2002

[10] XML - Extensible Markup Language, http://www.w3.org/XML/ - 10-28-2013

[11] Clark J., *XSLT - XSL Transformations, W3C Recommendation*, W3C - Word Wide Web Consortium, 16 November 1999

[12] Clark J., DeRose S., *XPath - XML Path Language*, W3C - Word Wide Web Consortium, 16 November 1999

[13] DeRose S., et al., *XPointer - XML Pointer Language*, W3C - Word Wide Web Consortium, 16 Agosto 2002

[14] David C., et al., *XML Schema*, W3C - Word Wide Web Consortium, 28 October 2004

[15] Bray T., et al., *Namespaces in XML 1.0*, W3C - Word Wide Web Consortium, 8 December 2009

[16] Yergeau F., *UTF-8, a transformation format of ISO 10646, RFC 3629*, IETF - Internet Engineering Task Force, November 2003, http://tools.ietf.org/html/rfc3629

[17] Hoffman P., Yergeau F., *UTF-16, an encoding of ISO 10646, RFC 2781*, IETF - Internet Engineering Task Force, February 2000, http://www.ietf.org/rfc/rfc2781.txt

[18] Christensen E., et al., *WSDL - Web Services Description Language*, W3C - Word Wide Web Consortium, 15 March 2001

[19] OASIS UDDI Specifications TC - Committee Specifications, https://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm - 10-28-2013

[20] Fielding R., *Architectural Styles and the Design of Network-based Software Architectures*, Ph.D. thesis, University Of California, Irvine, 2000

[21] Franks J., et al., *HTTP Authentication: Basic and Digest Access Authentication, RFC2617*, IETF - Internet Engineering Task Force, June 1999, http://www.ietf.org/rfc/rfc2617.txt

[22] Frank J., et al., *An Extension to HTTP : Digest Access Authentication, RFC 2069*, IETF - Internet Engineering Task Force, June 1999, http://www.ietf.org/rfc/rfc2069.txt

[23] Lawrence K., Kaler C., *Web Services Security: SOAP Message Security*, OASIS Standard Specification, 1 February 2006, https://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

[24] OAuth - Open Authentication, http://oauth.net/ - 10-28-2013

[25] Hammer-Lahav E., *The OAuth 1.0 Protocol, RFC5849*, IETF - Internet Engineering Task Force, April 2010, http://tools.ietf.org/html/rfc5849

[26] ACNP - Archivio Collettivo Nazionale dei Periodici, http://acnp.unibo.it/cgi-ser/start/it/cnr/fp.html - 10-28-2013

[27] SBN - Sistema Bibliotecario Nazionale, http://www.sbn.it/opacsbn/opac/iccu/free.jsp - 10-28-2013

[28] OCLC ILLiad, http://www.oclc.org/illiad.en.html - 10-28-2013

[29] Ariel¨ - Document Delivery on the Internet, http://www.oclc.org/research/activities/ariel.html - 10-28-2013

[30] Relais, http://www.relais-intl.com/ - 10-28-2013

[31] ISO ILL Protocolo Standards, http://www.collectionscanada.gc.ca/iso/ill/standard.htm - 10-28-2013

[32] RapidILL, http://rapidill.org/Public/AboutRapid - 10-28-2013

[33] Manuel J., Gairin R., "Web services in interlibrary loan transactions: the Spanish GTBib network," *Emerald*, volume 41, no. 2, pp. 48 – 53, 2013

[34] ILL SBN, http://prestito.iccu.sbn.it/ILLWeb/servlets/ILL - 10-28-2013

[35] National Information Standards Organization, November 2002, *Information Retrieval (Z39.50): Application Service Definition and Protocol Specification*

[36] Open-URL, http://www.oclc.org/research/activities/openurl.html?urlm=159705 - 10-28-2013

[37] Moulton R., Fretwell-Downing, *ISO ILL Protocol - Contribution to eLib Study on Document Requesting Standards*, 18 November 1997

[38] National Information Standards Organization, November 2008, *NCIP - NISO Circulation Interchange Protocol*

[39] Mangiaracina S., et al., "NILDE: developing a new generation tool for document delivery in Italy," *Emerald*, volume 36, no. 3, pp. 167 – 177, 2008

[40] Mangiaracina S., Tugnoli A., "NILDE reloaded: a new system open to international interlibrary loan," *Emerald*, volume 40, no. 2, pp. 88 – 92, 2012

[41] NILDE - Network Inter-Library Document Exchange, https://nilde.bo.cnr.it/ - 10-28-2013

[42] Recess! The RESTful PHP Framework, http://www.recessframework.org/ - 10-28-2013

[43] Zend Framework, http://framework.zend.com/ - 10-28-2013

[44] Slim Framework, http://www.slimframework.com/ - 10-28-2013

[45] OAuth Provider Framework, https://github.com/flijten/OAuth-consumer - 10-28-2013

[46] GitHub - Build software better, together., https://github.com/ - 10-28-2013