ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

Dipartimento di Informatica - Scienza e Ingegneria

Corso di Laurea Magistrale in Ingegneria Informatica

TESI DI LAUREA

in

Mobile System M

Studio e implementazione di un e-service per la comunicazione tra Aree Organizzative Omogenee di Documenti Amministrativi Protocollati

CANDIDATO:

Davide Filoni

RELATORE:

Prof. Paolo Bellavista

Anno Accademico: 2022/2023

“Ai miei genitori,
le fondamenta stabili
su cui ho costruito
questa avventura”

INDICE

PREFAZIONE

Gruppo Finmatica [Gruppo_Finmatica], presente sul mercato dal 1969, si posiziona tra i principali produttori di software e servizi in Italia. Questa holding è il risultato dell'unione di tre aziende:

- ADS Automated Data Systems: specializzata nella realizzazione di soluzioni e servizi per la Pubblica Amministrazione;
- Data Processing: focalizzata sulla creazione di soluzioni e servizi dedicati al mercato della Sanità e alle Aziende Private;
- Finmatica: l'azienda Holding del gruppo che eroga servizi centralizzati.

La missione principale dell'azienda è studiare e sviluppare soluzioni d'avanguardia, semplificare i processi, condividere informazioni e introdurre innovazioni radicali che generano valore e sviluppo sociale. Per perseguire questo obiettivo, circa il 10% del fatturato viene investito nella ricerca e nello sviluppo.

Gruppo Finmatica si dedica alla progettazione e realizzazione di soluzioni software per enti pubblici, aziende sanitarie e aziende private, concentrando i propri sforzi su due principali mercati:

- Pubblica Amministrazione: La suite completa di servizi e applicazioni software denominata Smart*Gov è progettata per realizzare progetti innovativi di e-government. L'obiettivo è implementare soluzioni e servizi che sostengano la trasformazione e migliorino l'efficienza del lavoro nella Pubblica Amministrazione, con oltre 500 enti che utilizzano quotidianamente i prodotti della suite.
- Sanità: La piattaforma robusta Smart*Health, basata su standard internazionali, consente alle aziende sanitarie di ottenere risultati concreti in tempi certi. L'intento è razionalizzare e semplificare i processi delle Aziende Sanitarie, migliorando costantemente la qualità dei servizi alla persona. Oltre 25.000 utenti utilizzano quotidianamente le soluzioni offerte.

Il Gruppo Finmatica si distingue per il suo impegno nella ricerca, nello sviluppo e nell'offerta di soluzioni innovative per migliorare l'efficienza e la qualità nei settori della Pubblica Amministrazione e della Sanità.

L'obiettivo principale della tesi svolta presso questa azienda è quello di progettare e sviluppare un e-service per la comunicazione tra Aree Organizzative Omogenee di Documenti Amministrativi Protocollati utilizzando sia il protocollo SOAP che la tecnologia REST, in modo tale da offrire un'alternativa al longevo meccanismo della posta elettronica certificata (PEC). In generale con il termine e-service si indica un servizio digitale realizzato da un Erogatore, attraverso l'implementazione delle API conformi alle linee guida di

Agenzia per l'Italia Digitale (AgID) [AgID], per assicurare ai Fruitori l’accesso ai propri dati e/o l’integrazione di processi.

Per raggiungere il fine prestabilito, il lavoro è stato articolato in tre fasi distinte. Nella prima ho approfondito le mie competenze riguardo alle linee guida fornite da AgID per la realizzazione di questo e-service, studiando alcuni documenti significativi, tra cui l’Allegato 6. Nella fase successiva ho implementato il client e il server del servizio utilizzando sia la tecnologia REST che il protocollo SOAP in modo tale da poter offrire ai clienti due versioni differenti del medesimo e-service. Infine, ho condotto una serie di test per verificare il corretto funzionamento e le prestazioni dei servizi implementati.

1 ITALIA DIGITALE 2026

Il Piano Nazionale di Ripresa e Resilienza (PNRR) [PNRR] si propone di raggiungere alcuni obiettivi ambiziosi per migliorare la posizione dell'Italia in Europa entro il 2026. Questi sono:

1. Diffusione dell'identità digitale, ovvero mira a garantire che il 70% della popolazione italiana utilizzi l'identità digitale. Ciò consentirebbe ai cittadini di accedere a servizi online in modo più efficiente e sicuro.
2. Colmare il divario delle competenze digitali, l'obiettivo è che almeno il 70% della popolazione sviluppi competenze digitali per essere in grado di utilizzare le tali tecnologie in modo efficace. Questo è cruciale per garantire che tutti possano beneficiare delle opportunità offerte dalla trasformazione digitale.
3. Adozione dei servizi cloud nella Pubblica Amministrazione (PA), ossia portare circa il 75% delle PA italiane a utilizzare servizi basati su cloud. In questo modo si contribuirebbe a migliorare l'efficienza e la flessibilità delle Pubbliche Amministrazioni, consentendo loro di erogare servizi più efficacemente.
4. Erogazione online dei servizi pubblici essenziali, quindi rendere disponibili online almeno l'80% dei servizi pubblici essenziali. Semplificando notevolmente la vita dei cittadini e consentendo loro di accedere a servizi importanti comodamente da casa.
5. Banda ultra-larga, in collaborazione con il Ministero dello Sviluppo Economico (MISE), il piano mira a raggiungere il 100% delle famiglie e delle imprese italiane con reti a banda ultra-larga. Ciò fornirebbe una connettività ad alta velocità in tutto il paese, essenziale per la competitività economica e l'accesso a servizi digitali avanzati.

L'obiettivo finale è migliorare la competitività e la resilienza dell'Italia nell'era digitale.

Figura 1. Riepilogo degli obiettivi di Italia Digitale 2026

In particolare, il 27% delle risorse totali del PNRR sono dedicate alla transizione digitale e la strategia per l'Italia digitale si sviluppa su due assi: la connettività a banda ultra-larga e gli interventi volti a trasformare la PA in chiave digitale.

Per quanto riguarda la trasformazione digitale della PA, l'obiettivo è quello di rendere la Pubblica Amministrazione la "migliore alleata" di cittadini e imprese, con un'offerta di servizi sempre più efficienti e facilmente accessibili. Una parte dei fondi del PNRR, 650 milioni, stanziati per questo asse sono stati destinati a ridurre il divario digitale che attualmente caratterizza la Pubblica Amministrazione italiana che spesso si traduce in una ridotta produttività e in un peso non sopportabile per cittadini e imprese, che debbono accedere alle diverse amministrazioni come "silos verticali", non interconnessi tra loro.

L'idea è quella di avere banche dati pubbliche che comunicano tra loro in modo tale da ridurre significativamente i costi amministrativi e il tempo necessario ai cittadini per ottenere servizi. L'implementazione di un unico profilo digitale permetterebbe alle amministrazioni di accedere alle informazioni sui cittadini in modo immediato, senza la necessità di richiederle ripetutamente.

In particolare, l'investimento prevede due misure:

1. Sviluppare una Piattaforma Digitale Nazionale Dati (PDND) [PDND] per garantire l'interoperabilità dei dati pubblici, permettendo così agli enti di erogare servizi in modo sicuro, più veloce ed efficace e ai cittadini di non fornire nuovamente informazioni che la PA già possiede.
2. Facilitare la realizzazione dello "Sportello Digitale Unico" (Single Digital Gateway), ossia supportare l'attuazione del regolamento europeo che ha l'obiettivo di uniformare l'accesso ai servizi digitali in tutto i Paesi membri dell'UE.

In dettaglio, l'obiettivo finale di questa tesi è la realizzazione di un servizio digitale pubblicato sulla Piattaforma PDND che consenta lo scambio di documenti amministrativi protocollati tra Pubbliche Amministrazioni in modo standard e rapido.

1.1 PIATTAFORMA PDND

La PDND costituisce un elemento essenziale all'interno dell'ecosistema dell'interoperabilità e funge da strumento chiave per gestire l'autenticazione, l'autorizzazione e la tracciabilità delle entità autorizzate, allo scopo di garantire la massima sicurezza delle informazioni. La sua missione principale è fornire un set di regole condivise, che facilita gli accordi di interoperabilità, semplificando le procedure di verifica e riducendo il carico di oneri e la complessità amministrativa.

In aggiunta, la piattaforma offre un Catalogo API completo, il quale espone tutti i servizi digitali resi disponibili dalle istituzioni pubbliche, chiamati e-service. Attraverso questo catalogo, è possibile richiedere l'accesso ai dati e, successivamente, integrarli nei propri servizi destinati ai cittadini.

L'obiettivo fondamentale della PDND è promuovere una completa interoperabilità tra i vari dataset e servizi essenziali sia a livello centrale che locale delle Pubbliche Amministrazioni. Questo consentirà di attuare il principio del “once-only”, garantendo che le istituzioni pubbliche non richiedano dati ai cittadini e alle imprese che già sono in loro possesso, e di valorizzare al massimo le risorse informative delle amministrazioni pubbliche.

La piattaforma presenta una serie di vantaggi chiave per diversi attori:

- Gli Erogatori di servizi, che mettono a disposizione e-service per l'accesso ai propri dati, possono contare su una sicurezza avanzata nello scambio di informazioni e su processi standardizzati.
- I Fruitori dei servizi possono accedere al catalogo degli e-service disponibili e integrare agevolmente le API nei propri servizi destinati ai cittadini e alle imprese.
- I tecnici sviluppatori, responsabili dell'implementazione e della gestione del ciclo di vita dei servizi digitali delle amministrazioni, possono sfruttare la piattaforma per effettuare integrazioni standardizzate.
- I responsabili della protezione dei dati delle entità aderenti possono accedere a documenti amministrativi standardizzati e garantire un processo uniforme per tutte le istituzioni coinvolte.
- Imprese e cittadini possono beneficiare dell'implementazione del principio del "once-only", evitando di dover fornire informazioni che hanno già comunicato in precedenza alle PA.

In sintesi, la PDND rappresenta un fondamentale strumento per promuovere l'efficienza, la sicurezza e la standardizzazione nei rapporti tra le amministrazioni pubbliche e i cittadini/imprese, contribuendo in modo significativo a semplificare l'accesso ai servizi digitali e migliorare la gestione dei dati.

1.2 E-SERVICE

Si definiscono e-service i servizi digitali offerti da un Erogatore, attraverso l’implementazione delle necessarie API conformi alle linee guida di AgID, per garantire ai Fruitori l’accesso ai propri dati e/o l’integrazione di processi.

Il Catalogo API è la componente unica e centralizzata che assicura agli Erogatori la registrazione e la pubblicazione dei propri e-service e ai Fruitori la consultazione di quelli pubblicati, integrata nell’Infrastruttura Interoperabilità PDND [Infrastruttura_ Interoperabilità_PDND].

Il Catalogo API, quindi è realizzato al fine di:

- favorire l'uso degli e-service grazie alla loro pubblicazione e la messa a disposizione della relativa documentazione tecnica;
- agevolare la gestione del ciclo di vita degli e-service;
- mitigare la creazione di interfacce ridondanti e/o con semantica sovrapposta.

Il Catalogo API permette ai Fruitori di consultare l'elenco degli e-service pubblicati dagli Erogatori al fine di conoscerne le modalità di accesso e i requisiti di fruizione, nonché verificarne l'utilità per il raggiungimento delle proprie finalità.

Ogni e-service pubblicato sul Catalogo API si compone delle seguenti parti:

- i descrittori dell'e-service, che definiscono le informazioni relative alla sua natura e le informazioni tecniche per usufruire dello stesso;
- una interfaccia per usufruire dell'e-service;
- la documentazione accessoria e manualistica per il suo utilizzo.

Gli accordi di interoperabilità rappresentano un meccanismo che consente agli Erogatori e ai Fruitori di stabilire una relazione per l'utilizzo degli e-service. Il processo di stipula di un accordo di interoperabilità avviene esclusivamente attraverso l'Infrastruttura di Interoperabilità della PDND e comporta i seguenti passaggi:

- Il Fruitore individua l'e-service di suo interesse dal Catalogo API disponibile.
- Quando il Fruitore invia la richiesta per stipulare un accordo di interoperabilità per un particolare e-service, è tenuto a dichiarare, se non lo ha già fatto in precedenza, di possedere gli attributi "Dichiarati" e/o a quelli "Verificati", se richiesti dai requisiti di fruizione stabiliti dall'Erogatore per quel determinato e-service.
- L'Erogatore, prima di confermare o rifiutare la richiesta di stipula dell'accordo di interoperabilità per l'e-service scelto, deve verificare se il Fruitore possiede gli attributi "Verificati".

Nella terminologia utilizzata, un "Attributo" è una caratteristica o informazione posseduta da un Aderente. Sono identificate tre tipologie di attributi:

- Certificati: Si riferiscono alle caratteristiche possedute dall'Aderente che sono definite sulla base delle informazioni contenute nelle banche dati di interesse nazionale e/o detenute da altre fonti autorevoli.
- Dichiarati: Si tratta delle caratteristiche il cui possesso è stato dichiarato dall'Aderente stesso sotto la sua esclusiva responsabilità.

- Verificati: Questi sono attributi il cui possesso è stato indicato dall'Aderente stesso e verificato da almeno un altro Aderente, agendo in veste di Erogatore, durante la stipula di un accordo di interoperabilità.

Di seguito una struttura generale della PDND Interoperabilità.

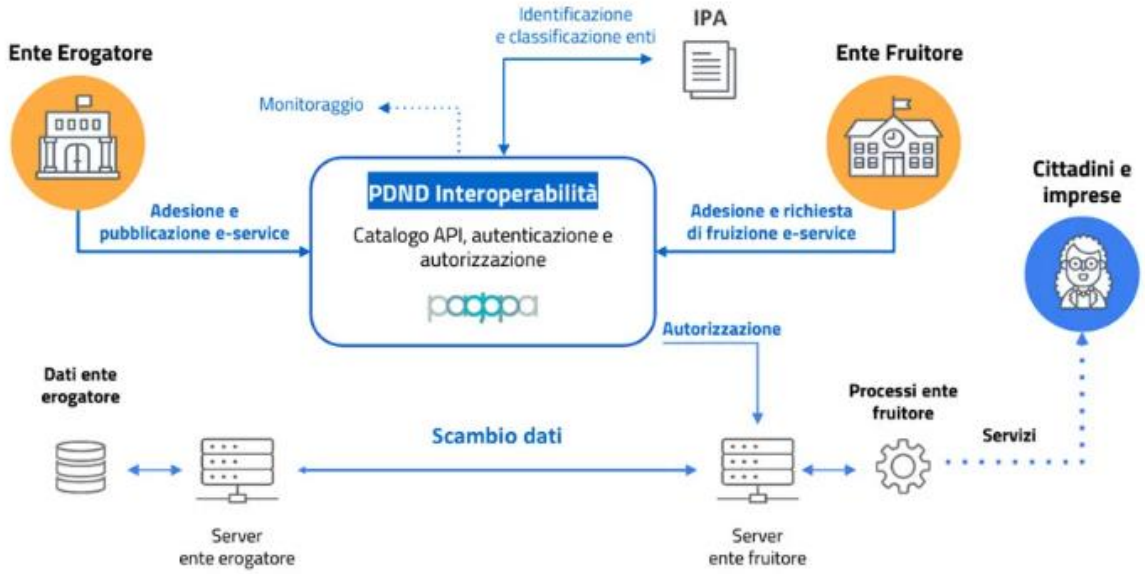


Figura 2. Struttura PDND Interoperabilità

1.2.1 Ciclo di vita degli e-service

L'Allegato 2 [Allegato_2] individua le modalità che gli Erogatori devono attuare per registrare, pubblicare e aggiornare le informazioni relative ai propri e-service sul Catalogo API e le modalità che i Fruitori devono rispettare per consultare l'elenco degli e-service pubblicati sul Catalogo API.

In particolare, per "descrittori degli e-service" si intende l'insieme di dati registrati per ciascun servizio digitale, il cui complesso fornisce informazioni dettagliate sulle caratteristiche e sul suo funzionamento. Essi sono definiti dagli Erogatori.

Gli e-service di un Erogatore sono resi disponibili ai Fruitori attraverso la pubblicazione del relativo descrittore degli e-service sul Catalogo API. Inoltre, ogni Erogatore deve definire per ciascun e-service una delle tecnologie previste, ossia REST [REST] o SOAP [SOAP], caricando il corrispondente file di specifica delle API, ovvero un file OpenAPI per i servizi REST e WSDL per i servizi SOAP.

Un descrittore rimane associato all'e-service per l'intero ciclo di vita di quest'ultimo. Esso può trovarsi in diversi stati:

- "Bozza": l'Erogatore inizia a popolare i dati e i metadati del descrittore. Questo stato consente all'Erogatore di registrare le informazioni iniziali e di completarle in seguito.
- "Attivo": Una volta che l'Erogatore ha completato le informazioni obbligatorie del descrittore, può pubblicare l'e-service. In questo stato, l'Erogatore rende l'e-service disponibile agli Aderenti per la stipula di accordi di interoperabilità. In un certo istante vi può essere un solo descrittore in stato "Attivo" per uno specifico e-service.
- "Deprecato": Se l'Erogatore decide di non accettare nuovi Fruitori per l'e-service, ma desidera comunque mantenerlo disponibile per coloro che hanno già stipulato un accordo di interoperabilità.
- "Archiviato": Nel caso in cui non vengano stipulati accordi di interoperabilità relativi a un particolare descrittore di e-service, le informazioni associate possono essere archiviate dall'Infrastruttura di Interoperabilità della PDND.

L'Erogatore può eseguire diverse azioni per modificare lo stato di un descrittore di e-service che sono riportate nel diagramma di stato sottostante.

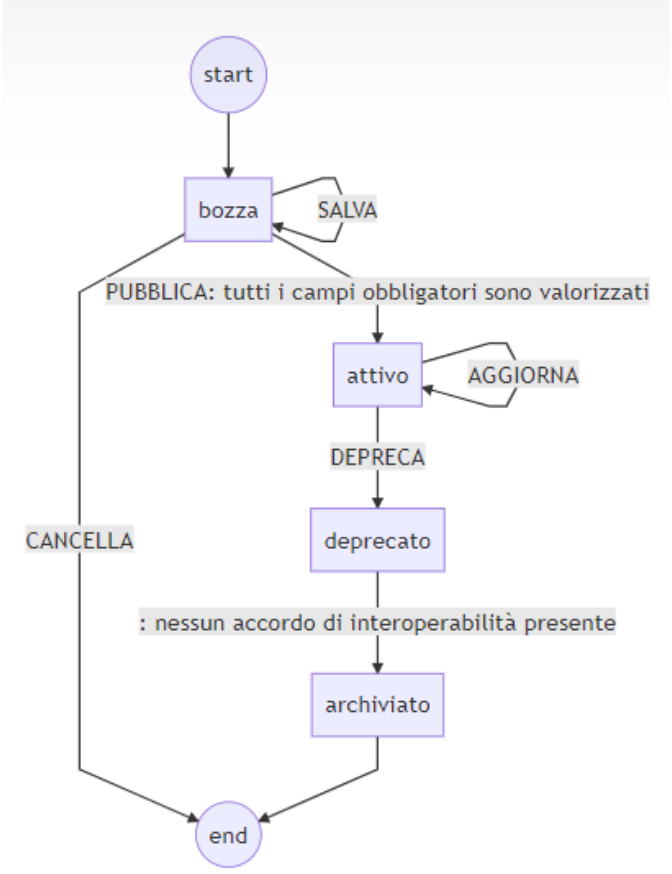


Figura 3. Diagramma di stato del ciclo di vita di un e-service

1.2.2 Sicurezza

L’Infrastruttura Interoperabilità PDND deve fornire agli Aderenti le funzionalità necessarie ad assicurare l’autorizzazione e autenticazione dei Fruitori al fine di accedere agli e-service pubblicati degli Erogatori.

Attraverso le funzionalità offerte dall'Infrastruttura, viene stabilito un trust machine-to-machine tra i seguenti attori:

- Sistemi informatici degli Aderenti, ovvero Erogatori e Fruitori, e sistemi informatici dell'Infrastruttura di Interoperabilità della PDND.
- Sistemi informatici degli Aderenti stessi.

Per realizzare tale meccanismo di sicurezza si usano dei client, ovvero dei contenitori di utenti abilitati e di chiavi pubbliche da questi caricate. Esistono due tipi di client: uno che si rivolge verso gli Erogatori degli e-service, e uno verso le API esposte da PDND Interoperabilità. Il primo tipo, il client e-service, è associabile agli e-service per i quali un Fruitore ha una richiesta di fruizione attiva, mentre il secondo tipo, il client api interop, è direttamente utilizzabile per ottenere informazioni da PDND Interoperabilità attraverso le sue API.

Le chiavi pubbliche fanno parte di un corredo crittografico di cui gli Aderenti si dotano per ottenere un voucher da PDND Interoperabilità, che potrà essere speso presso gli e-service degli Erogatori nel caso di un client e-service, o presso l'API gateway di PDND Interoperabilità nel caso di un client api interop.

L'intero ciclo di vita dei voucher implica una serie di interazioni basate su comunicazioni machine-to-machine. Il flusso varia a seconda se un Aderente sta richiedendo un voucher da utilizzare su un e-service del Catalogo come Fruitore (client e-service) o se desidera interagire con le API di PDND Interoperabilità per ottenere informazioni (client api interop).

I voucher, in realtà, sono rappresentati come token JWT (JSON Web Tokens) [RFC_7519]. Il flusso di autenticazione implementato segue il protocollo OAuth 2.0 e fa riferimento all'RFC 6750 per l'uso del token Bearer [RFC_6750] e all'RFC 7521 per l'autorizzazione del client tramite client assertion [RFC_7521].

Il paradigma utilizzato per l'implementazione di tutte le API esposte da PDND Interoperabilità è il modello architetturale REST.

L'aderente costruisce una client assertion e la firma con una chiave privata la cui omologa pubblica è stata depositata sulla piattaforma PDND Interoperabilità. Poi, chiama il server autorizzativo di PDND Interoperabilità richiedendo un voucher valido per quell'asserzione firmata. In caso tutti i controlli diano esito positivo, PDND Interoperabilità restituisce un voucher all'aderente, che può essere spendibile presso:

- Le API di Interoperabilità: una volta ottenuto il voucher, esso viene inserito come header autorizzativo (Authorization) nelle successive chiamate verso le API di PDND Interoperabilità.
- Un e-service: una volta ottenuto il voucher, esso viene inserito come header autorizzativo nelle chiamate verso l'e-service di un Erogatore. L'Erogatore, ricevuta la richiesta, effettua i debiti controlli. È possibile che alcuni di questi controlli richiedano il recupero di informazioni detenute da PDND Interoperabilità, come ad esempio il controllo della firma mediante la chiave pubblica. In quel caso, l'Erogatore stesso dovrà richiedere un voucher spendibile presso le API di PDND Interoperabilità. Una volta terminati i controlli, l'Erogatore autorizza il Fruitore all'accesso al servizio per la specifica finalità richiesta.

In alternativa all'utilizzo del client api interop, per evitare che l'Erogatore debba richiedere un voucher a PDND per ottenere la chiave pubblica che verrà poi utilizzata per autenticare il token JWT ricevuto dal Fruitore in precedenza, esite un ulteriore meccanismo che semplifica l'operazione senza dover "staccare" un secondo token. Nel dettaglio, l'Erogatore ottiene la lista di chiavi in uso da un file JSON esposto nella cartella ".well-known" di PDND Interoperabilità, utilizzando l'URL indicato all'interno della scheda di ogni singolo e-service. Con l'ausilio del kid e di tale lista riesce a estrapolare la chiave pubblica necessaria per la validazione del token JWT ricevuto.

Riassumendo il flusso della sicurezza si sviluppa nei seguenti passaggi:

1. Il Fruitore genera e inoltra a PDND Interoperabilità la client assertion, firmata con la chiave privata corrispondente alla chiave pubblica depositata sulla piattaforma PDND, per ottenere un voucher (token JWT);
2. PDND Interoperabilità rilascia al Fruitore un voucher;
3. Il Fruitore invia quindi una richiesta di utilizzo di un e-service all'Erogatore, inserendo il voucher nell'header Authorization;

A questo punto l'Erogatore ha la necessità di ottenere la chiave pubblica associata al Fruitore per validare la firma del token JWT appena ricevuto, per fare ciò ha due alternative:

Prima alternativa: Richiesta di un secondo voucher e chiamata REST a PDND Interoperabilità

4. L'Erogatore genera e invia una client assertion (client api interop) a PDND Interoperabilità;
5. PDND Interoperabilità rilascia all'Erogatore un voucher;
6. Successivamente l'Erogatore estrae il kid contenuto nell'header del token JWT ricevuto dal Fruitore per effettuare una richiesta a PDND Interoperabilità per

ottenere la chiave pubblica corrispondente. In tale richiesta inserisce il voucher appena restituito da PDND Interoperabilità nell'header Authorization;

7. PDND Interoperabilità restituisce all'Erogatore la chiave pubblica corrispondente al kid inserito nella richiesta REST;

Seconda alternativa: Ottenimento della chiave pubblica tramite cartella ".well-know".

4. L'Erogatore effettua una richiesta HTTP GET, utilizzando l'URL indicato nella scheda dell'e-service;
5. Riceve un file JSON contenete la lista di chiavi pubbliche in uso;
6. All'interno del file, l'Erogatore cerca l'oggetto che ha lo stesso kid presente nell'header del voucher ricevuto dal Fruitore. In quello stesso oggetto troverà la chiave pubblica in corrispondenza del valore del parametro "n".

In entrambi i casi, dopo aver ottenuto la chiave pubblica, l'Erogatore effettua tutti i controlli necessari per la validazione del voucher ricevuto dal Fruitore e, in caso di esito positivo, risponde alla richiesta del Fruitore.

Per maggiore chiarezza di seguito vengono riportati i diagrammi di interazione tra gli attori partecipanti, rappresentanti le due possibili alternative:

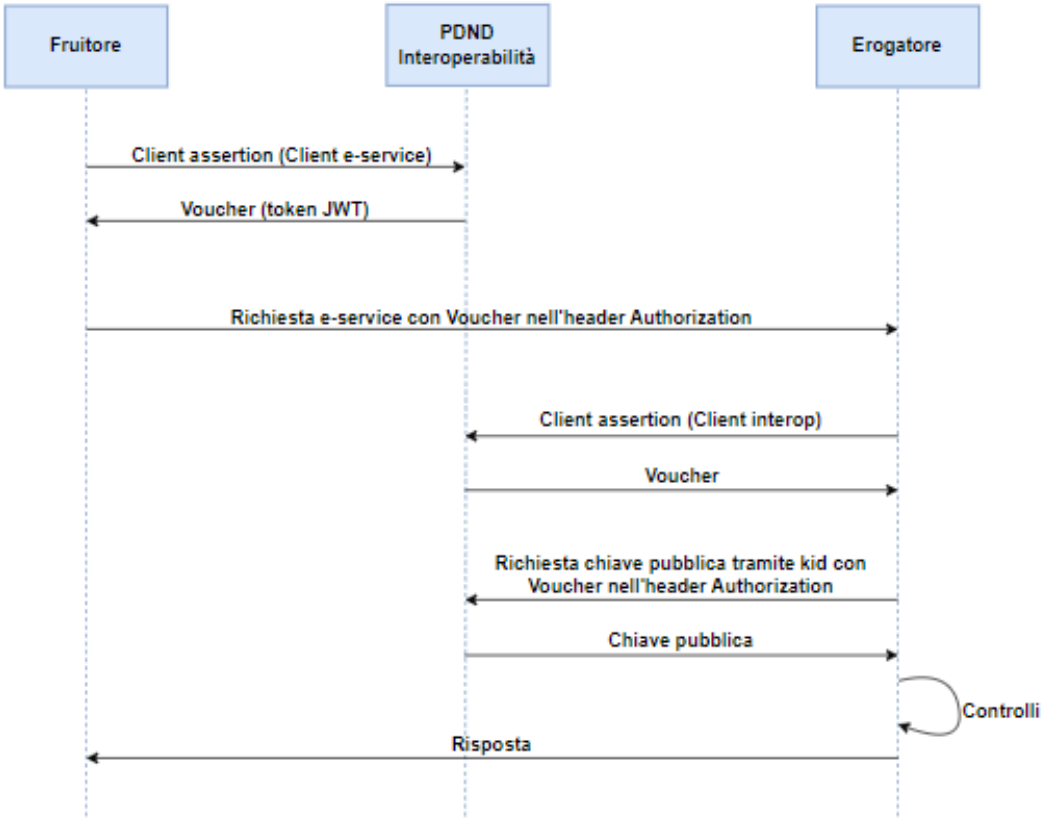


Figura 4. Prima alternativa: Richiesta di un secondo voucher e chiamata REST a PDND Interoperabilità

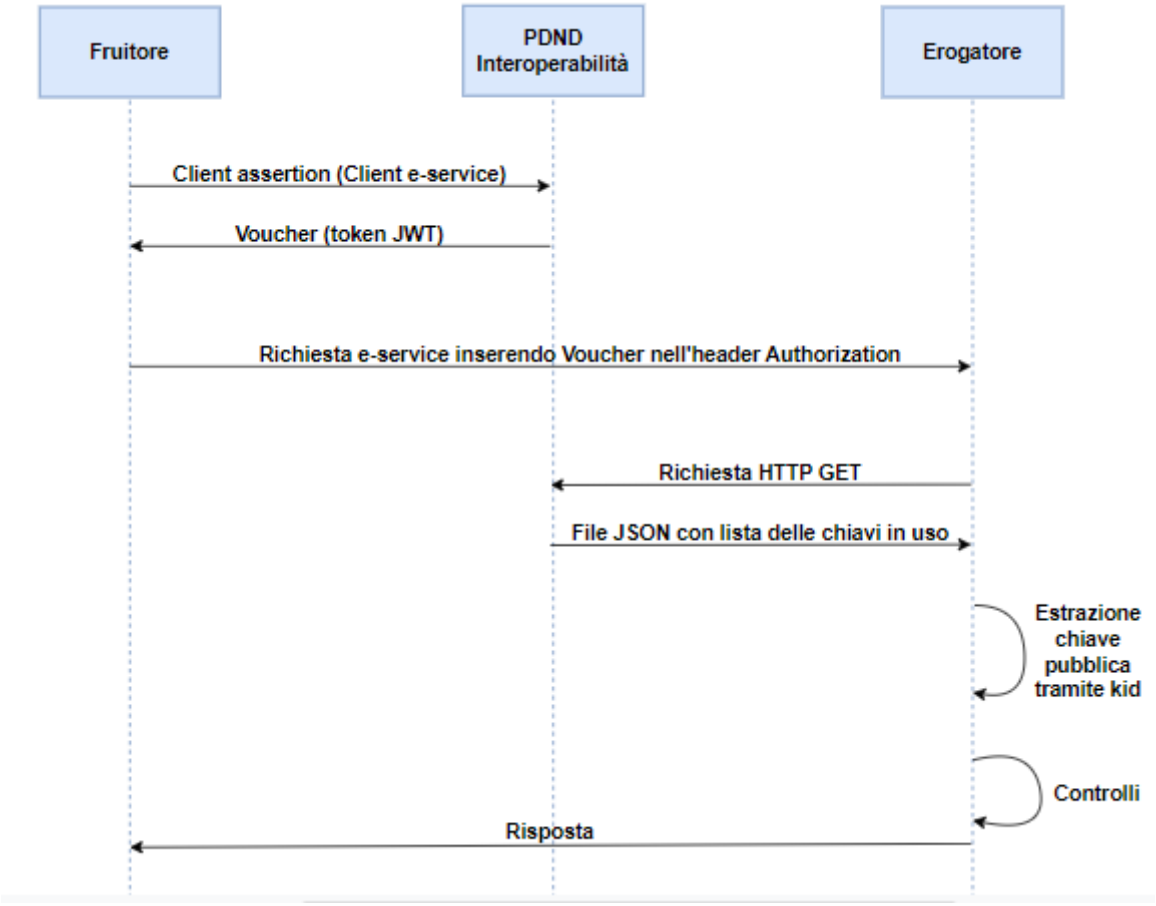


Figura 5. Seconda alternativa: Ottenimento della chiave pubblica tramite cartella ".well-know".

2 ALLEGATO 6: COMUNICAZIONE TRA AOO DI DOCUMENTI AMMINISTRATIVI PROTOCOLLATI

L'Allegato 6 [Allegato_6] stabilisce le modalità tecniche per assicurare il trasferimento di documenti amministrativi informatici tra Aree Organizzative Omogenee (AOO) della Pubblica Amministrazione.

Il "Protocollo Informatico" è la componente software che garantisce la gestione della registrazione di protocollo e della segnatura di protocollo, in particolare assicura le seguenti azioni:

- produzione della segnatura di protocollo, cioè l'apposizione o l'associazione al documento originale in forma permanente e non modificabile dei metadati funzionali alla ricezione o spedizione dalle Pubbliche Amministrazioni;
- registrazione di protocollo, cioè l'attività di memorizzazione dei dati necessari a conservare le informazioni per ogni documento ricevuto o spedito dalle PA.

Di seguito tutte le specifiche necessarie per l'implementazione dell'e-service che consente l'invio e la ricezione di documenti amministrativi protocollati tra AOO.

2.1 MESSAGGIO DI PROTOCOLLO

Lo scambio di documenti amministrativi protocollati tra AOO vede coinvolti:

- il mittente, ovvero l'AOO che invia i documenti;
- il destinatario, ossia l'AOO che li riceve.

Un messaggio di protocollo è l'elemento atomico di interesse per consentire lo scambio di documenti amministrativi protocollati tra AOO. Esso è una struttura logica che:

- contiene il documento amministrativo informatico principale;
- può contenere un numero qualsiasi di documenti amministrativi informatici allegati;
- include la segnatura di protocollo del messaggio protocollato.

Figura 6. Struttura logica del messaggio di protocollo

La segnatura di protocollo deve essere associata in forma permanente al documento principale e agli allegati che con esso formano il messaggio di protocollo. Per fare ciò il sistema informatico dell'AOO mittente:

- riporta nella segnatura di protocollo l'impronta del documento principale e, se presenti, degli allegati;
- assicura l'autenticità e integrità della segnatura di protocollo.

Il controllo della validità amministrativa del documento principale, degli allegati e dei dati riportati nella segnatura di protocollo è a carico della AOO mittente e dev'essere effettuato prima della composizione del messaggio di protocollo.

Inoltre, per assicurare la non ripudiabilità dello scambio tra AOO, le informazioni della segnatura di protocollo devono essere memorizzate sia nel sistema di gestione dei documenti della AOO mittente che in quello delle AOO destinataria.

2.1.1 Struttura della segnatura di protocollo

La segnatura di protocollo prevede le seguenti sezioni:

- Intestazione: contiene i dati identificativi e le informazioni fondamentali del messaggio;
- Riferimento (opzionale): include le informazioni relative al messaggio di protocollo ricevuto;
- Descrizione: comprende le informazioni descrittive inerenti al contenuto del messaggio;
- Signature: per consentire la firma della segnatura di protocollo.

Le AOO mittenti che generano la segnatura di protocollo devono assicurare la conformità rispetto allo schema XML [Segnatura_protocollo].

Nel dettaglio:

- L'Intestazione deve contenere gli elementi essenziali per l'identificazione e la caratterizzazione amministrativa del messaggio. Nello specifico, la sezione comprende l'Identificatore della registrazione relativa al messaggio protocollato in uscita. Esso riporta i seguenti dati:
    a. indicazione dell'amministrazione mittente (codice IPA);
    b. indicazione dell'AOO mittente (codice AOO);
    c. indicazione del registro nell'ambito del quale è stata effettuata la registrazione;
    d. numero progressivo di protocollo;
    e. data di registrazione;
    f. l'oggetto del messaggio;
    g. la classificazione del messaggio;
    h. il fascicolo del messaggio.
- Il Riferimento può includere i dati per consentire al mittente di indicare il messaggio di protocollo ricevuto che ha prodotto la presente segnatura.
- La Descrizione contiene le informazioni che descrivono i corrispondenti (mittente e destinatario) interessati nello scambio e i riferimenti al documento principale e agli eventuali allegati. In particolare, comprende l'impronta informatica del documento principale e degli eventuali allegati necessaria per associarli in forma permanente alla segnatura di protocollo.
- Infine, la Signature mantiene le informazioni per garantire la firma della segnatura di protocollo da parte della AOO mittente per garantire l'autenticità e integrità.

2.2 REGOLE DI PROCESSAMENTO

Il flusso di processamento che le AOO mittenti devono seguire per assicurare la corretta formazione del messaggio di protocollo è il seguente:

1. Formazione del documento principale (document) e degli eventuali allegati (attachment).
2. Calcolo dell'impronta del documento principale e degli eventuali allegati.
3. Generazione del numero di protocollo da assegnare al messaggio.
4. Formazione della segnatura di protocollo che deve rispettare lo schema XML [Segnatura_protocollo], utilizzando le impronte create al passo precedente.

5. Apposizione di un “sigillo elettronico qualificato” alla segnatura di protocollo per garantire l’integrità e autenticità, ovvero la Firma.

L’AOO mittente assicura l'atomicità delle ultime tre operazioni.

2.3 FLUSSI DI COMUNICAZIONE

Le esigenze di comunicazione tra AOO mittente e AOO destinataria per garantire l’inoltro di un messaggio protocollato possono richiedere le seguenti operazioni:

- Inoltro di un messaggio protocollato (MessaggioInoltro) da una AOO mittente ad una AOO destinataria e la relativa conferma (ConfermaMessaggioInoltro) se richiesta dal mittente (settando l’attributo “confermaRicezione” a true);
- Annullamento protocollazione mittente (AnnullamentoInoltroMittente), nel caso in cui successivamente all’inoltro l’AOO mittente adotti un provvedimento per il suo annullamento;
- Annullamento protocollazione destinatario (AnnullamentoInoltroDestinatario), nel caso in cui successivamente alla conferma di ricezione l’AOO destinataria adotti un provvedimento per il suo annullamento.

Di seguito viene riportato il diagramma UML che riassume schematicamente le possibili interazioni tra mittente e destinatario:

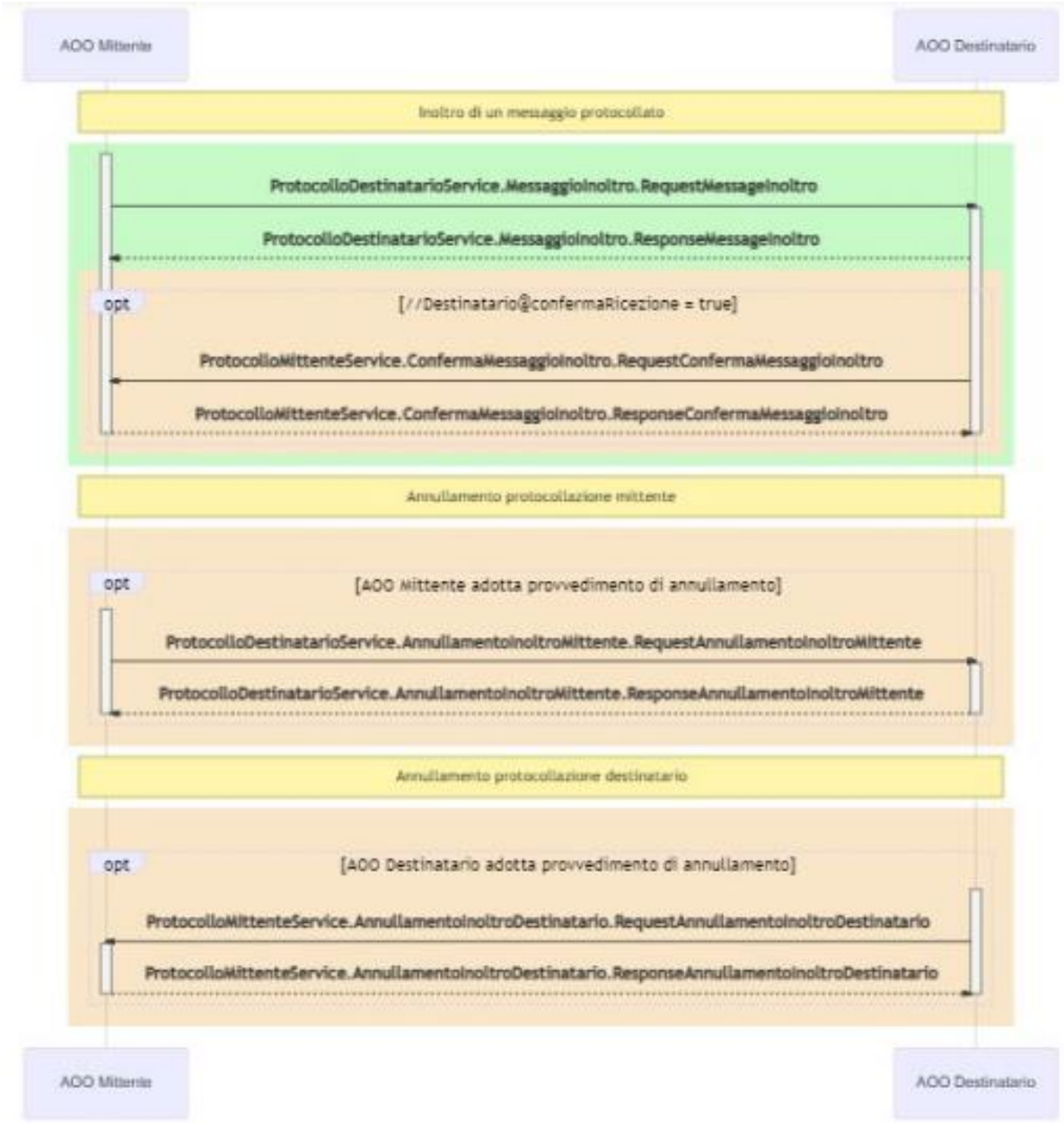


Figura 7. Interazioni tra AOO mittente e AOO destinatario

Per garantire la comunicazione tra AOO le Amministrazioni Pubbliche devono registrare e mantenere aggiornato, per ogni AOO individuata nella propria organizzazione, l'indice dei domicili digitali delle PA e dei gestori di pubblici servizi (IPA) [IPA] con il prefisso condiviso dagli endpoint di esposizione dei servizi.

2.3.1 Inoltro di un messaggio protocollato (MessaggioInoltro)

Per effettuare l'inoltro di un messaggio protocollato i passi da eseguire sono i seguenti:

1. L'AOO mittente genera il messaggio di protocollo;
2. L'AOO destinatario, ricevuto il messaggio di protocollo e verifica la segnatura, in particolare:
    a) Controlla la correttezza della firma;
    b) Esamina la corrispondenza dell'impronta del documento principale presente nella segnatura di protocollo e il documento principale ricevuto;
    c) se presenti allegati, per ognuno di essi verifica la corrispondenza dell'impronta dell'allegato presente nella segnatura e l'allegato ricevuto.

    Se l'AOO destinataria è riuscita a verificare il messaggio di protocollo ricevuto risponde indicando l'Identificatore associato dalla AOO mittente.

    Se invece non è riuscita a controllare il messaggio di protocollo ricevuto segnala all'AOO mittente l'anomalia riscontrata, nel dettaglio:

    a) 001_ValidazioneFirma: se la firma della segnatura di protocollo non è verificata;
    b) 002_AnomaliaImpronta: se almeno una delle impronte riportate nella segnatura di protocollo non è verificata.

I successivi step sono effettuati se l'attributo "confermaRicezione" è settato a true:

3. L'AOO destinataria inoltra la conferma di protocollazione del messaggio all'AOO mittente a conclusione del processo di protocollazione in ingresso.

    Successivamente può effettuare la verifica dei file ricevuti ed in caso di anomalie, le segnala alla AOO mittente, nello specifico restituisce:

    a) 003_DocumentoAllegatiNonLeggibili: se almeno uno dei file ricevuti risulta non leggibile.
    b) 004_DocumentoAllegatiErrorreValidazioneFirma: se almeno uno dei file ricevuti risulta firmato e la validazione della stessa fallisce.
    c) 005_DocumentoAllegatiErroreValidazioneMarcaTemporale: se almeno uno dei file ricevuti risulta con marca temporale e la validazione di quest'ultima fallisce.
    d) 006_DocumentoAllegatiErroreValidazioneSigillo: se almeno uno dei file ricevuti possiede un sigillo elettronico e la validazione dello stesso fallisce.

L'AOO destinataria controlla la ricevibilità del messaggio di protocollo ricevuto, ossia esamina se il messaggio è conforme alle specifiche, ed in caso negativo restituisce l'anomalia 000_Irricevibile e l'indicazione della motivazione di irricevibilità. Inoltre, deve generare il numero di protocollo per messaggio di protocollo ricevuto e memorizzarlo nel registro di protocollo in ingresso.

Infine, inoltra la conferma di protocollazione del messaggio ricevuto includendo l'Identificatore associato a quest'ultimo dall'AOO mittente e l'Identificatore da essa associato.

4. Nel caso in cui l'AOO destinatario non segnali anomalie, l'AOO mittente memorizza la conferma di protocollazione nel registro di protocollo per assicurarne la persistenza. Viceversa, l'AOO mittente ritiene la transazione non conclusa.

2.3.2 Annullamento protocollazione mittente (AnnullamentoMittente)

Per consentire l'annullamento di un messaggio, precedentemente inoltrato, da parte dell'AOO mittente, i passi da seguire sono i sottostanti:

1. L'AOO mittente inoltra la richiesta di annullamento di un messaggio di protocollo precedentemente inviato, indicando l'Identificatore associato a quest'ultimo da essa al momento dell'inoltro, l'Identificatore associato dal destinatario indicato nella ricevuta di ricezione e il riferimento al provvedimento che ne determina l'annullamento.

2. L'AOO destinatario invia la ricevuta di annullamento di un messaggio di protocollo precedentemente ricevuto. Vi possono essere vari casi:

    a. Se l'operazione è stata completata con successo, la ricevuta di annullamento deve contenere l'Identificatore associato al messaggio appena annullato dall'AOO mittente al momento dell'inoltro e l'Identificatore associato dall'AOO destinatario indicato, precedentemente, nella ricevuta di ricezione inoltrata al mittente;

    b. In caso di irricevibilità viene restituita l'anomalia 000_Irricevibile indicando il motivo di irricevibilità;

    c. Nel caso in cui non venga trovato il messaggio di protocollo viene restituita l'anomalia 007_ErroreIdentificatoreNonTrovato.

2.3.3 Annullamento protocollazione destinatario (AnnullamentoDestinatario)

I passi necessari per permettere l'annullamento di un messaggio protocollato precedentemente ricevuto da parte dell'AOO destinatario sono i seguenti:

1. L'AOO destinatario inoltra la richiesta di annullamento di un messaggio di protocollo precedentemente ricevuto, indicando l'Identificatore associato a quest'ultimo dall'AOO mittente al momento dell'inoltro, l'Identificatore associato

dall'AOO destinatario indicato nella ricevuta di ricezione e il riferimento al provvedimento che ne determina l'annullamento.

3. L'AOO mittente inoltra la ricevuta di annullamento di un messaggio di protocollo precedentemente inviato. Vi possono essere vari casi:

    a. Se l'operazione è stata completata con successo, la ricevuta di annullamento deve contenere l'Identificatore associato al messaggio dal mittente al momento dell'inoltro e l'Identificatore associato dal destinatario indicato nella ricevuta di ricezione.

    b. In caso di irricevibilità viene restituita l'anomalia 000_Irricevibile indicando il motivo di irricevibilità;

    c. Nel caso in cui non il messaggio di protocollo non venga trovato viene restituita l'anomalia 007_ErroreIdentificatoreNonTrovato.

2.4 LIVELLI DI SERVIZIO E POLITICHE DI RITRASMISSIONE

2.4.1 Definizioni

I seguenti concetti sono utilizzati per calcolare il livello di servizio

Finestra temporale di erogazione (FdM)	Intervallo temporale utilizzato per la misurazione dei livelli di servizio.
Classificazione dei disservizi (Severity)	I disservizi sono classificati in base al seguente criterio: • Bloccante: l'amministrazione non è in grado di usufruire del servizio; • Non bloccante: l'amministrazione è in grado di usufruire del servizio, ma con prestazioni degradate.
Finestra temporale di osservazione (T_{oss})	Arco temporale di osservazione delle misurazioni del livello di servizio ai fini del calcolo degli indicatori.

2.4.2 Indicatori livelli di servizio

Disponibilità (D)	Percentuale di tempo durante il quale il servizio è funzionante (ovvero non si verifica su di esso un disservizio di tipo bloccante) all'interno della FdM e rispetto al T_{oss}, meglio definita come: $$D = \frac{\sum_{j=1}^{M} d_j}{T}$$ dove: • M rappresenta il numero totale di disservizi bloccanti; • d_j è la durata, espressa in minuti, del disservizio bloccante j-esimo nell'ambito della FdM; • T è la finestra temporale di misurazione della disponibilità, pari al tempo totale espresso in minuti della FdM del servizio nel T_{oss}
Round Trip Delay (RTD), SOAP Service Invocation and Processing	Tempo medio in secondi che intercorre tra l'invio della Request SOAP da parte del client all'endpoint del servizio e la ricezione della Response SOAP. $$RTD = T_{req} - T_{resp}$$ Considerando una banda minima garantita sul canale di trasporto di 64 kbps e la dimensione delle coppie di messaggi request-response SOAP con media di 50 KB e deviazione standard di 10 KB.

I target di livello del servizio che si desidera raggiungere sono quelli di ottenere una Disponibilità del 99.9 % e il 98% delle chiamate entro 1 secondo (RTD), considerando come finestra di misurazione dal Lunedì al Venerdì dalle 07.00 alle 23.00 e Sabato dalle 07.00 alle 13.00 con festivi esclusi nell'arco di un mese (T_{oss}).

2.4.3 Politica di ritrasmissione

La presente politica di ritrasmissione a livello applicativo viene utilizzata dall'AOO mittente, se:

- L'attesa della risposta SOAP a seguito di una richiesta SOAP supera il valore soglia di timeout:

$$timeout > \frac{MSG_{dim}}{MSGRef_{dim}} * RTD_{soglia}$$

  dove:

  - MSG_{dim} è la dimensione in KB della coppia di messaggi request-response relativa al servizio invocato;
  - $MSGRef_{dim}$è la dimensione in KB della coppia di messaggi request-response utilizzata per la valutazione dei target di livello del servizio;
  - RTD_{soglia} è pari al valore soglia deciso per il RTD.

- Nel caso in cui, a seguito di una risposta SOAP, viene ricevuto uno stato malfunzionamento del servizio "Response SOAP (Soap fault) e HTTP 5xx". In particolare, l'AOO mittente adotta una politica di ritrasmissione con N tentativi e backoff incrementale del tipo 2N ore con $1 \leq N \leq 3$. Se il disservizio persiste al termine dell'N-esimo tentativo di ritrasmissione, l'AOO mittente può attivare la procedura di "Gestione dei Disservizi".

2.5 GESTIONE DEI DISSERVIZI

In caso l'AOO mittente riscontri problemi nel flusso di comunicazione che sembrano essere causati dall'AOO destinataria, deve avviare la seguente procedura per la segnalazione e la gestione delle anomalie riscontrate:

1. L'AOO mittente recupera dall'IPA i riferimenti della AOO destinataria e procede a segnalare l'anomalia riscontrata contattandola.
2. Qualora l'anomalia segnalata determini un disservizio bloccate, l'AOO destinataria deve individuare e fornire all'AOO mittente una soluzione temporanea (workaround) per garantire l'operatività nel periodo in cui il disservizio è in fase di risoluzione ed il periodo stimato di ripristino del servizio.
3. L'AOO mittente in caso di disservizio bloccante adopera il workaround operativo fornito dalla AOO destinataria per soddisfare le proprie esigenze di comunicazione, fino alla scadenza del periodo di ripristino del servizio comunicato l'AOO destinataria.

3 IMPLEMENTAZIONE DEGLI E-SERVICE

In questo capitolo sono illustrate dettagliatamente le implementazioni, sia lato server che lato client, degli e-service realizzati utilizzando sia il protocollo SOAP e che la tecnologia REST, seguendo le specifiche appena descritte nel capitolo precedente.

3.1 PROTOCOLLO SOAP

Il Protocollo SOAP (Simple Object Access Protocol) è un fondamentale standard di comunicazione utilizzato nell'ambito delle tecnologie web per consentire lo scambio di messaggi tra applicazioni software attraverso la rete. Esso si basa sul metalinguaggio XML per strutturare i dati nei messaggi e definisce regole standard per la loro trasmissione. SOAP è ampiamente utilizzato in servizi web per consentire l'integrazione tra sistemi software eterogenei, permettendo loro di comunicare in modo affidabile e interoperabile. Questo protocollo è stato uno dei precursori dei servizi web e continua a svolgere un ruolo chiave nell'architettura orientata ai servizi (Service Oriented Architecture) e nelle applicazioni distribuite.

3.1.1 Interfacce di servizio SOAP

Le AOO mittente e AOO destinataria, per dare seguito alla comunicazione di messaggi protocollati utilizzando SOAP, implementano le interfacce di servizio descritte nei seguenti Web Services Description Language (WSDL).

Ogni AOO per assicurare la funzione di AOO mittente e AOO destinataria implementa:

• protocollo-mittente.wsdl [protocollo_mittente] valorizzando l'attributo location dell'elemento wsdl:service\wsdl:port\soap:address con <endpoint>/protocollo/mittente;

• protocollo-destinatario.wsdl [protocollo_destinatario] valorizzando l'attributo location dell'elemento wsdl:service\wsdl:port\soap:address con <endpoint>/protocollo/destinatario;

Le AOO mittente e AOO destinatario per garantire il non ripudio della comunicazione, provvedono alla firma dei messaggi scambiati ed al loro trasposto su canale TLS tramite SOAP coerentemente alla specifica WS-Security.

3.1.2 Server

Per ottenere una base di partenza con tutte le classi di supporto necessarie per poter implementare l'e-service lato server è stato sfruttato un plugin dell'IDE Eclipse [Eclipse] denominato "Web Tools Platform (WTP)" [WTP]. Tale strumento include un Wizard per la generazione automatica di codice Java da WSDL e XSD. In questo modo è stato possibile autogenerare le classi fondamentali per lo sviluppo passando a WTP i due file WSDL citati nel paragrafo precedente e i due schemi XSD per la rappresentazione della segnatura [Segnatura_protocollo] e del messaggio di protocollo [Messaggio_protocollo].

Queste classi sono state poi introdotte in un contesto già presente in azienda chiamato “Si4csWeb” il quale sfrutta il framework Spring Boot [Spring_Boot]. Esso è un framework open source basato su Spring, progettato per semplificare lo sviluppo di applicazioni Java. La sua principale caratteristica è la facilità di avvio e configurazione delle applicazioni, consentendo così agli sviluppatori di concentrarsi sulla logica di business invece di gestire dettagli tecnici complessi. In particolare, i vantaggi di Spring Boot includono:

1. Semplicità di avvio di nuovi progetti.
2. Configurazione automatica, questo riduce la necessità di configurare manualmente numerosi aspetti dell'applicazione.
3. Gestione delle dipendenze: Spring Boot integra una gestione delle dipendenze avanzata, ciò significa che è possibile specificare le dipendenze in modo dichiarativo e il framework gestirà il resto, garantendo che le versioni delle librerie siano coerenti e compatibili.
4. Supporto per servizi integrati (servizi web RESTful, applicazioni web, servizi di messaggistica e molto altro).

Grazie a questi vantaggi, Spring Boot è diventato uno dei framework Java più popolari per lo sviluppo di applicazioni, riducendo notevolmente il tempo e lo sforzo necessario per creare applicativi robusti e performanti.

3.1.2.1 Si4csWeb

Dopo aver integrato le classi autogenerate all’interno del contesto Si4csWeb già esistente è stato necessario implementare la logica di business legata all’e-service da sviluppare. Grazie all’utilizzo di Spring Boot, ai metodi realizzati viene passato come parametro d’ingresso una rappresentazione sottoforma di oggetto Java della busta SOAP ricevuta, in modo tale da poterla processare con maggiore facilità. Nel dettaglio sono stati realizzati quattro metodi:

- messaggioInoltro
- annullamentoInoltroMittente
- confermaMessaggioInoltro
- annullamentoInoltroDestinatario

I primi due appartengono alla classe “ProtocolloDestinatarioServiceImpl” e gli ultimi due alla classe “ProtocolloMittenteServiceImpl”. Tutti questi metodi si avvalgono di una libreria aziendale interna che semplifica le operazioni di salvataggio sia del messaggio ricevuto all’interno del database che degli allegati associati sul “documentale” denominata “finmatica-smartdoc”.

In particolare, tutti i metodi estraggono dall’oggetto passato come paramento le informazioni necessarie per memorizzare in modo persistente il messaggio appena

ricevuto, ovvero: Mittente, Destinatario, Oggetto, Data, Id del messaggio, gli allegati associati e il CS_TAG.

Un aspetto da sottolineare è quello che interessa la gestione degli allegati che verranno poi salvati su un database secondario detto “documentale” per evitare sia di congestionare il database principale che la replicazione di quest’ultimi. Per memorizzare un allegato è necessario indicare il Nome, il Content type e il contenuto del file. Nello specifico:

- Nel caso del metodo messaggioInoltro vi sono sicuramente più allegati: il primo è un file XML rappresentante la Segnatura ottenuto attraverso il marshalling dell’omonimo oggetto Java, in più sono presenti il documento principale e gli eventuali documenti amministrativi informatici opzionali.
- Mentre nel caso degli altri metodi verrà salvato un solo allegato che è un file XML che rappresenta il messaggio appena ricevuto ottenuto anch’esso tramite il marshalling dell’oggetto passato come parametro d’ingresso.

Per quanto riguarda l’operazione di marshalling viene utilizzata un’apposita libreria denominata “javax.xml.bind”. Essa è una libreria Java standard che fa parte del Java Architecture for XML Binding (JAXB) e offre potenti metodi per la manipolazione di dati XML in applicazioni Java. In particolare, semplifica le operazioni come il marshalling (conversione da oggetti Java a XML) e l'unmarshalling (conversione da XML a oggetti Java). Per effettuare queste operazioni è necessario aggiungere l’annotazione @XmlRootElement alle classi che devono essere trasformate da oggetti Java a XML e viceversa.

Perciò con l’ausilio di tale libreria viene effettuata la conversione da oggetto Java (Segnatura o messaggio ricevuto) a rappresentazione XML che viene poi salvata in un file temporaneo. Successivamente questo file viene utilizzato per associare il contenuto all’entry da memorizzare su documentale e infine si procede all’eliminazione del file, per liberare tutte le risorse adoperate.

A titolo di esempio e per mettere in chiaro le operazioni svolte, nell’immagine sottostante viene riportato la parte di codice del metodo annullamentoInoltroMittente dove vengono effettuate le azioni appena descritte.

```
try {
    JAXBContext jaxbContext = JAXBContext.newInstance(RequestAnnullamentoInoltroMittenteType.class);
    Marshaller jaxbMarshaller = jaxbContext.createMarshaller();
    jaxbMarshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);

    File tmp = File.createTempFile("tmp", ".xml");
    jaxbMarshaller.marshal(parameters, tmp);

    doc.setContentStream(new FileInputStream(tmp));

    tmp.deleteOnExit();

} catch (JAXBException | IOException  e ){
    e.printStackTrace();
}
```

Figura 8. Operazione di marshalling

Infine, il CS_TAG è un parametro che indica l'implementazione da utilizzare in fase di invio del messaggio ed è utilizzato anche come filtro per quanto concerne i messaggi ricevuti. Il suo valore è ricavato tramite una lettura da database ed è contenuto in una tabella denominata REGISTRO che mantiene tutti i dati necessari per la configurazione generale e delle varie applicazioni che fanno riferimento a quel particolare database. In questa situazione si sfrutta la "Dependency Injection" di Spring Boot. In particolare, i due concetti utilizzati sono:

- L'iniezione di dipendenze, che consente di fornire a una classe le dipendenze di cui ha bisogno, cioè altre classi o oggetti da cui dipende, anziché crearle autonomamente;
- l'annotazione @Autowired, che indica al framework di iniettare automaticamente una dipendenza in una classe. Quando Spring rileva questa annotazione, cerca un'istanza della classe richiesta dal suo contesto di applicazione e la fornisce alla classe in cui è stata applicata l'annotazione.

In questo caso la dipendenza che viene iniettata è un'istanza della classe di utility chiamata si4csService che semplifica la lettura di parametri dalla tabella REGISTRO. La figura sottostante rappresenta le istruzioni utilizzate per sfruttare la Dependency Injection.

```
@Autowired
private Si4csService si4csService;

// Altre istruzioni

msgOnDB.setCsTag(si4csService.getParametroRegistro("CS_TAG","PRODUCTS/CIM/CONFIG/PDND_INTEROP"));
```

Figura 9. Dependency Injection di si4csService

3.1.3 Client

Il client che si occupa della generazione e dell'invio delle buste SOAP per l'invocazione dei quattro metodi implementati lato server, mostrati nei paragrafi precedenti, è stato integrato in un contesto già esistente chiamato "Si4Cim" che si occupa di tutte le operazioni di invio. In particolare, "Si4Cim" è una libreria standard per i prodotti Finmatica per risolvere le problematiche di comunicazione e fornisce varie implementazioni Java per l'invio di messaggi di tipo mail, fax, sms, PEC, ecc.

Nel dettaglio, all'interno di questa libreria, è stata inserita una nuova classe denominata PDNDInteropMessage che si occupa specificatamente di produrre e spedire i messaggi SOAP destinati al server per invocare il metodo desiderato.

3.1.3.1 PDNDInteropMessage

PDNDInteropMessage estende la classe astratta già presente nel contesto chiamata GenericMessage che, come si evince dal nome, rappresenta un messaggio generico. Quest'ultima viene estesa dalle varie classi che implementano le specifiche tecnologie di invio di un messaggio.

Nello specifico, PDNDInteropMessage effettua l'overriding di alcuni metodi ereditati dalla classe astratta suddetta che vengono utilizzati per la configurazione iniziale, quali:

- Metodi di setting come setCsTag, setMessageId e setParam che sono sfruttati per impostare i parametri iniziali della classe.
- Metodi che aggiungono elementi a vettori, ovvero addAttachment e addRecipient che servono rispettivamente per l'inserimento di nuovi allegati e destinatari.

Oltre a questi vi è anche il metodo send che contiene la logica di business vera e propria, ossia quello che si occupa della generazione e dell'invio della busta SOAP. Nel dettaglio, quest'ultimo, come prima operazione estrae dal vettore recipient l'URI del destinatario, successivamente discrimina quale metodo invocare basandosi sul numero degli allegati da inviare e dal loro nome.

In particolare, se il vettore attachment contiene un solo allegato il metodo corrispondente da invocare è individuato in base al nome di quest'ultimo, ovvero:

- Conferma.xml e Eccezione.xml rappresentano due varianti del metodo ConfermaMessaggioInoltro;
- Annullamento.xml indica che dovrà essere invocato il metodo AnnullamentoInoltroDestinatario;
- Infine, l'allegato AnnullamentoInoltroMittente.xml porterà all'invio della busta SOAP che provocherà la chiamata dell'omonimo metodo lato server.

Per tutte queste varianti la logica utilizzata è la medesima. Come prima operazione viene inizializzato il Protocollo corretto in base all'operazione da effettuare, Protocollo Mittente

per i primi due casi e destinatario per l'ultimo. Nell'immagine sottostante è riportata la parte di codice che rappresenta questa operazione, ovvero viene generato l'endpoint a cui inviare la busta SOAP e nell'header Authorization viene inserito il token JWT richiesto alla piattaforma PDND Interoperabilità (le modalità utilizzate per ottenere il voucher verranno dettagliate nel paragrafo Sicurezza). Come si può evincere dalla figura anche lato client vengono utilizzate le classi di supporto autogenerate con il plugin WTP di Eclipse.

```
ProtocolloMittenteService mittenteService = null;
try {
    mittenteService = new ProtocolloMittenteService(new URL(uri +"?wsdl"), SERVICE_NAME_MITT);
} catch (MalformedURLException e) {
    e.printStackTrace();
}

ProtocolloMittentePortType portMitt = mittenteService.getProtocolloMittenteService();
BindingProvider bp_mitt = (BindingProvider)portMitt;
bp_mitt.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, uri);

// Header
TokenPDND tokenPDND = TokenPDND.getInstance();
Map<String, List<String>> header = new HashMap<>();
List<String> t = new ArrayList<>();
t.add("Bearer "+tokenPDND.getAccessToken(PDND_INTEROP_CHIAVE_REGISTRO));
header.put("Authorization", t);
bp_mitt.getRequestContext().put(MessageContext.HTTP_REQUEST_HEADERS, header);
```

Figura 10. Inizializzazione Protocollo Mittente

Una volta inizializzato il protocollo corretto, l'operazione successiva è quella di trasformare il file XML presente negli allegati in un oggetto adeguato, a differenza di quanto fatto lato server, in questo caso, non viene utilizzata "javax.xml.bind" ma la libreria "org.dom4j.io". Questa scelta è stata presa per essere allineati con l'applicazione soprastante che utilizzerà il client implementato per inviare le buste SOAP, chiamata Protocollo. Infatti "javax.xml.bind" richiede una struttura molto specifica del file XML con pochi gradi di libertà e non compatibile con l'applicazione già esistente, mentre questa libreria consente di poter lavorare correttamente con i file provenienti dal Protocollo.

In dettaglio, "org.dom4j.io" è una libreria open-source per la manipolazione e l'analisi di documenti XML in Java. Quest'ultima fornisce funzionalità per la lettura e la scrittura efficiente di documenti XML utilizzando la struttura DOM, consentendo agli sviluppatori di lavorare con tali documenti in maniera agevole. In questo modo consente di estrapolare dal file le informazioni necessarie per popolare correttamente l'oggetto Request opportuno.

È doveroso sottolineare come la scelta di questa strategia abbia complicato leggermente l'implementazione delle operazioni di unmarshalling in quanto con la libreria utilizzata lato server il processamento del file XML in possesso era preso in carico da "javax.xml.bind", che si occupava in maniera autonoma della conversione in oggetto Java, sgravando di

questo compito il programmatore. Mentre “org.dom4j.io”, si limita a facilitare l’accesso al file XML, ma è il programmatore che si occupa di creare l’oggetto Java corrispondente, estrapolare dal file le informazioni necessarie e sfruttarle per popolare opportunamente l’oggetto appena creato. Sostanzialmente il programmatore è costretto a lavorare a più basso livello, aumentando così la complessità del suo lavoro ma ottenendo gradi di libertà maggiori, per arrivare al medesimo risultato.

A titolo d’esempio viene riportato un estratto di codice per mostrare l’utilizzo della libreria “org.dom4j.io” per il popolamento della Request corretta. Da questa immagine si può notare l’utilizzo di una DatatypeFactory per poter convertire data e ora nel particolare formato XMLGregorianCalendar e come vengono trattati i campi opzionali presenti nelle richieste, appunto per sottolineare come le operazioni di unmarshalling siano più complesse e di più basso livello rispetto a quelle viste lato server con la libreria “javax.xml.bind”.

```
IdentificatoreType id = new IdentificatoreType();
DatatypeFactory factory = DatatypeFactory.newInstance();

// Codice amministrazione
CodiceIPA codA = new CodiceIPA();
codA.setValue(doc.selectSingleNode(ns+"CodiceAmministrazione)"+str).getStringValue());
id.setCodiceAmministrazione(codA);

// Estrapolazione di altre informazioni

// Ora registrazione
if(doc.selectSingleNode(ns+"OraRegistrazione)"+str) != null) {
    XMLGregorianCalendar ora = factory.newXMLGregorianCalendar(
                               doc.selectSingleNode(ns+"OraRegistrazione)"+str).getStringValue());
    log.info(ora.toString());
    id.setOraRegistrazione(ora);
}
```

Figura 11. Unmarshalling del file XML utilizzando la libreria “org.dom4j.io”

Infine, l’ultima operazione consiste nell’invio della richiesta e la ricezione dell’ACK corrispondente.

Se invece il vettore attachment presenta più allegati significa che l’operazione da effettuare è una MessaggioInoltro. Il primo allegato è sempre la Segnatura, mentre i successivi rappresenteranno il documento principale e gli eventuali allegati.

La logica utilizzata è simile ai metodi precedenti, infatti, dopo aver inizializzato il Protocollo destinatario e quindi aver creato l’endpoint con cui comunicare, si popola l’oggetto SegnaturaInformaticaType in maniera opportuna utilizzando anche in questo caso la libreria “org.dom4j.io”. L’unica differenza sta nel fatto che il documento principale e gli eventuali allegati devono essere inseriti in una lista di FileType che deve essere associata successivamente alla richiesta che viene generata, come viene mostrato nella figura sottostante.

```
// Rimozione segnatura
this.attachment.remove(a);

// Documento principale + eventuali allegati
List<FileType> listFile = new ArrayList<FileType>();

for( Object obj : this.attachment) {
    Attachment a = (Attachment) obj;
    FileType ft = new FileType();
    ft.setNomeFile(a.getFileName());
    ft.setValue(a.getByteContent());
    ft.setMimeType("application/pdf");
    listFile.add(ft);

}
reqMI.setFile(listFile);
```

Figura 12. Trattamento del documento principale e degli eventuali allegati

3.2 TECNOLOGIA REST

A differenza dell'implementazione con l'ausilio del protocollo SOAP, la realizzazione degli e-service che consentono la comunicazione tra AOO di Documenti Amministrativi Protocollati tramite tecnologia REST, non presenta ancora uno standard inserito all'interno dell'Allegato 6. Infatti, al momento, AgID ha fornito su GitLab solo le specifiche OpenAPI delle interfacce applicative, ovvero il file yaml denominato "protocollo-comunicazione-aoo.yaml" [protocollo-comunicazione-aoo.yaml], e ha annunciato che presto anche queste specifiche verranno standardizzate e pubblicate all'interno dell'allegato suddetto. Perciò sono state utilizzate queste indicazioni come linee guida per realizzare gli e-service richiesti.

Nello specifico, REST (Representational State Transfer) è uno stile architetturale di progettazione per i servizi web e le applicazioni distribuite basato sulla trasmissione di dati tra client e server su HTTP senza ulteriori livelli, quali ad esempio SOAP. I concetti chiave di tale tecnologia sono i seguenti:

1. Rappresentazione delle risorse: REST si basa sul concetto di "risorse", che possono essere oggetti, dati o servizi che possono essere acceduti o manipolati tramite una richiesta. Le risorse sono identificate da URI.
2. Operazioni CRUD: Le operazioni comuni eseguite su tali risorse sono Create (POST), Read (GET), Update (PUT), e Delete (DELETE).
3. Stateless: Le richieste HTTP in un'applicazione RESTful sono autonome e contengono tutte le informazioni necessarie per comprendere e gestire la richiesta. Il server non mantiene alcuno stato della sessione del client tra le richieste.

4. Interfaccia uniforme: Un'interfaccia uniforme definisce un insieme comune di vincoli, come per esempio l'uso di URI per identificare risorse e i verbi HTTP standard per eseguire le operazioni.
5. Scambio di rappresentazioni: Le risposte alle richieste REST contengono rappresentazioni dei dati o delle risorse, spesso in formato JSON o XML. I client richiedono una rappresentazione specifica e possono interagire con le risorse attraverso queste.
6. Sistema client-server: Il server è responsabile della gestione delle risorse, mentre il client è responsabile di presentare le informazioni all'utente e di comunicare con il server.
7. Cache: REST supporta la memorizzazione nella cache delle risposte da parte del client, il che può ridurre il carico sul server e migliorare le prestazioni.

REST è ampiamente utilizzato nell'ambito delle applicazioni web e delle API web, ed è spesso preferito per la sua semplicità e la sua natura “snella”.

3.2.1 Interazioni tra mittente e destinatario

Nello scenario REST, rispetto al protocollo SOAP, vi sono alcune differenze per quanto riguarda la comunicazione tra mittente e destinatario.

Nel caso delle operazioni di annullamento sia da parte del mittente che del destinatario viene seguita la stessa logica vista per il protocollo SOAP, ovvero il mittente o il destinatario, in base a chi vuole effettuare l’azione, invia una richiesta http (/mittente/annullamento/destinatario oppure /destinatario/annullamento/mittente) per invalidare il documento desiderato e riceve, in risposta, l’esito dell’annullamento. Per maggiore chiarezza di seguito sono riportati i diagrammi di sequenza delle due operazioni.

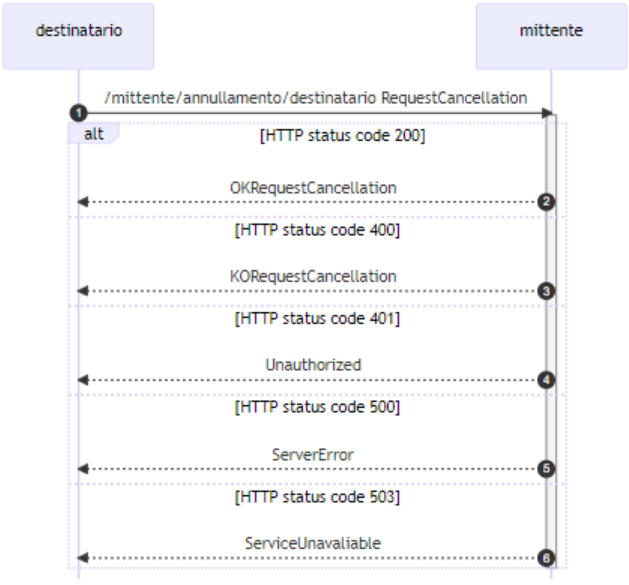


Figura 13. Annullamento Destinatario

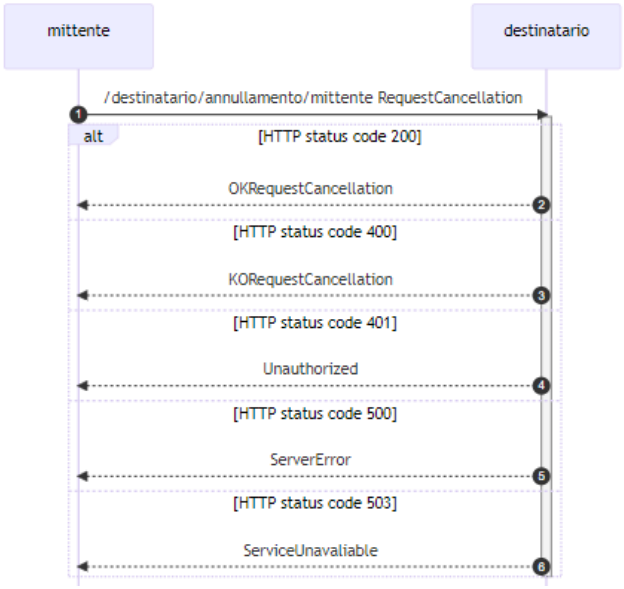


Figura 14. Annullamento Mittente

La differenza principale sta nell'interazione per effettuare l'invio del documento protocollato principale e dei suoi eventuali allegati. Infatti, tale comunicazione si sviluppa in due fasi, ovvero:

- Nella prima fase il mittente invia una richiesta http (/destinatario/inoltro) al destinatario, tale messaggio contiene una struttura denominata MessaggioProtocollo che al suo interno presenta la Segnatura e gli identificativi del documento principale e degli eventuali allegati. Tali id sono ottenuti come concatenazione di vari campi caratterizzanti i documenti, in particolare: codice amministrazione, codice AOO, codice registro, numero registrazione, data registrazione e id della risorsa;

- Nella seconda fase vi possono essere due scenari possibili:

    a. Se la Segnatura presenta qualche errore, il destinatario risponde, tramite chiamata http (/mittente/esito-destinatario), al mittente segnalando il problema riscontrato;

    b. Se, invece, la Segnatura risulta corretta, il destinatario effettua una serie di chiamate http (/mittente/documento) al mittente, pari al numero di documenti da scaricare (documento principale ed eventuali allegati). Per identificare tali file vengono utilizzate gli id univoci passati dal mittente nella fase precedente. Per concludere il destinatario invia un "esito-destinatario" per segnalare l'esito delle operazioni appena effettuate.

Anche in questo caso, di seguito, è riportato il diagramma di sequenza per chiarire al meglio l'interazione tra i due attori partecipanti.

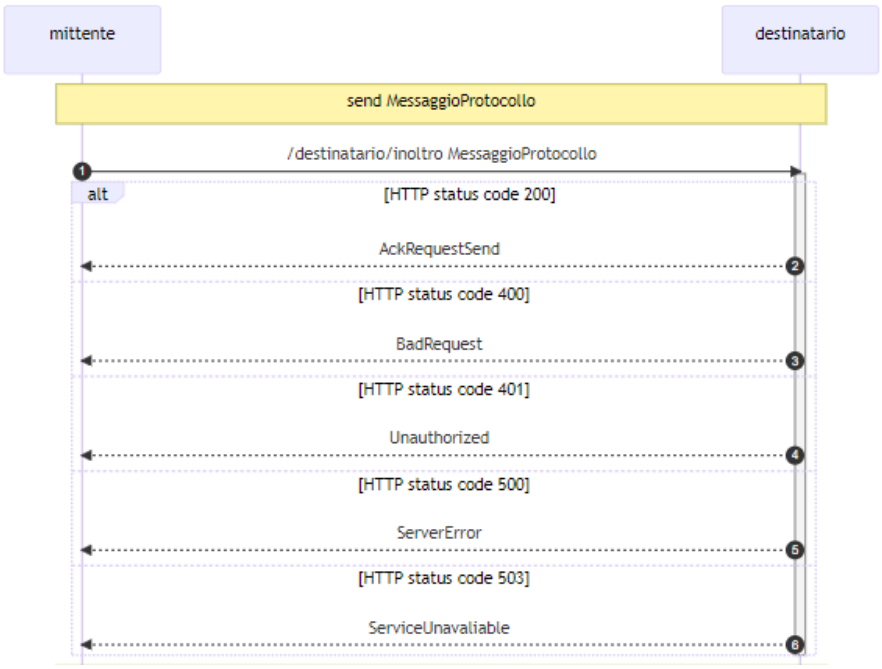


Figura 15. Invio documento "prima fase"

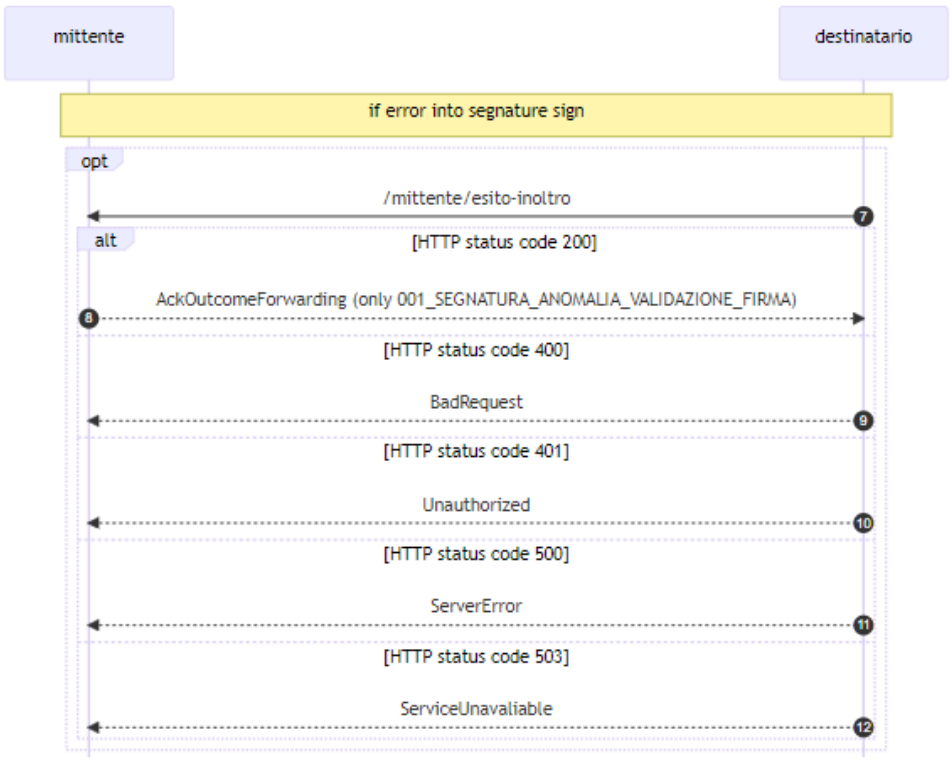


Figura 16. Invio documento "seconda fase, scenario a."

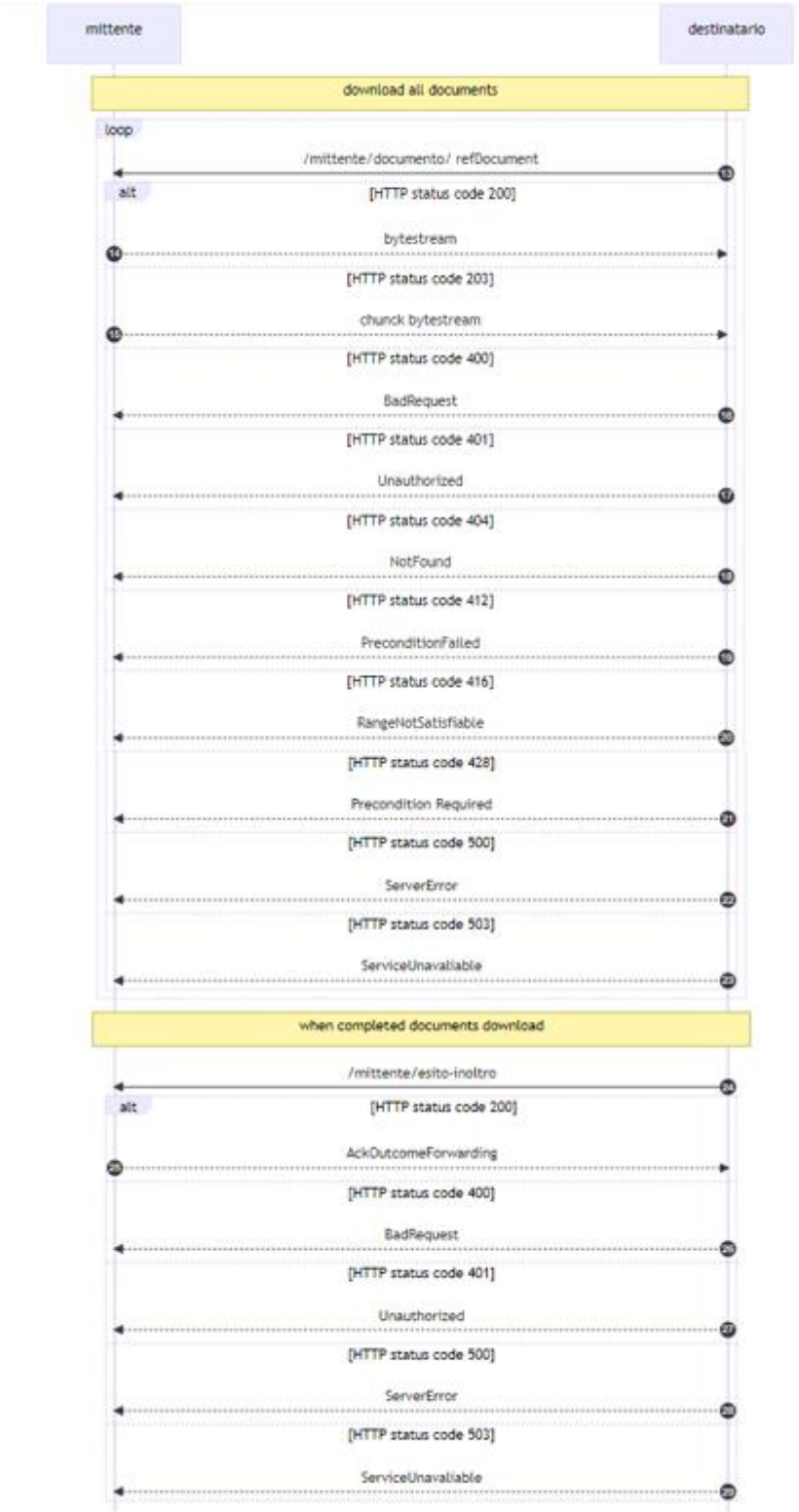


Figura 17. Invio documento "seconda fase, scenario b"

Sostanzialmente vi è un "ribaltamento" del flusso di comunicazione rispetto all'e-service realizzato con protocollo SOAP visto in precedenza. Infatti, nell'implementazione SOAP è il

client che effettua un upload dei documenti che devono essere ricevuti dal server, perciò il flusso va dal client verso il server; mentre utilizzando la tecnologia REST è il server che, mediante gli id indicati dal client al passo precedente, effettua il download dei documenti; quindi, in questo scenario, il flusso va dal server al client. Questo è stato uno degli aspetti che ha richiesto maggiore impegno dal punto di vista implementativo dato che vi è una completa inversione della logica con cui avviene lo scambio dei file tra le due entità che partecipano alla comunicazione.

3.2.2 Server

Il server REST è stato integrato all'interno di un contesto già esistente in Finmatica denominato "Si4csEngine". Questo contesto si occupa di tutte le operazioni di ricezione e invio di messaggi di qualunque genere e per questo motivo contiene anche la libreria "Si4Cim" dove è stato inserito il client SOAP e, vedremo, anche il client REST.

Per implementare i servizi REST lato server è stata sfruttata la libreria "javax.ws.rs". Essa fornisce gli strumenti necessari per creare e gestire servizi web RESTful, compresi l'esposizione di risorse, la gestione delle richieste HTTP e la generazione di risposte HTTP. La libreria offre una serie di annotazioni, come "@Path", "@GET", "@POST", "@PUT" e "@DELETE", che semplificano la definizione degli endpoint REST e il collegamento di metodi Java ai percorsi URL. Inoltre, supporta la Dependency Injection per l'accesso a variabili di contesto e parametri dei metodi.

Con "javax.ws.rs", è possibile creare applicazioni web che rispettano i principi di REST, consentendo la comunicazione tramite richieste HTTP standard, come ottenere informazioni (GET), inviare dati (POST), aggiornare risorse (PUT) o eliminarle (DELETE).

In particolare, per sviluppare gli e-service REST lato server sono state create due nuove classi Java chiamate DestinatarioApi e MittenteApi per definire i metodi e gli endpoint necessari.

3.2.2.1 DestinatarioApi e MittenteApi

I metodi implementati sono i seguenti:

- annullamentoDestinatario
- richiestaDocumento
- esitoInoltro
- annullamentoMittente
- inoltro

I primi tre sono contenuti nella classe MittenteApi e i restanti sono all'interno della classe DestinatarioApi.

Le operazioni di annullamento e di esitoInoltro non differiscono più di tanto dalla versione realizzata con protocollo SOAP vista nei paragrafi precedenti. Infatti, anche in questo caso, si fa uso della libreria aziendale "finmatica-smartdoc" che semplifica il salvataggio del messaggio ricevuto all'interno del database e del documento principale e degli eventuali allegati associati sul "documentale". Perciò viene utilizzato un oggetto MsgOnDbSmartdoc che viene popolato in maniera opportuno estrapolando tutte le informazioni necessarie dal messaggio appena ricevuto, dopodiché questo oggetto consentirà di aggiungere un'entry corrispondente sul database. È giusto evidenziare come le annotazioni offerte dalla libreria "javax.ws.rs" consentono, con poche istruzioni, di agganciare l'implementazione al servizio corretto.

A titolo d'esempio, l'immagine sottostante riporta la sezione di codice nella quale sono adoperate tali annotazioni per mostrarne l'utilizzo.

```
@Path("/destinatario")
public class DestinatarioApi {

    @POST
    @Path("/annullamento/mittente")
    @Consumes({ "application/json" })
    @Produces({ "application/json" })
    @ApiOperation(value = "", notes = "", response = annullamentoMittente200Response.class, authorizations = {
                                                                    @Authorization(value = "bearerAuth")}, tags={  })
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "Richiesta completata con successo", response = annullamentoMittente200Response.class),
            @ApiResponse(code = 400, message = "Richiesta completata con successo", response = annullamentoMittente400Response.class),
            @ApiResponse(code = 401, message = "Richiesta non autorizzata", response = Error.class),
            @ApiResponse(code = 500, message = "Errore processamento richiesta", response = Error.class),
            @ApiResponse(code = 503, message = "Servizio non disponibile", response = Error.class)
    })
public Response annullamentoMittente(@Valid @NotNull RequestCancellation req, @HeaderParam("Authorization") String authHeader){
```

Figura 18. Annotazioni "javax.ws.rs" per il mapping del metodo

Invece cambia leggermente la logica di inoltro e richiestaDocumento perché, come già detto nel paragrafo precedente, l'operazione di ottenimento del documento principale e degli eventuali allegati si sviluppa in due differenti fasi: nella prima il mittente indica solo gli identificativi dei vari documenti e in un secondo momento il destinatario effettua il download dei documenti, con l'ausilio degli id passati precedentemente. Perciò si ha un'inversione del flusso di trasferimento dei documenti.

In dettaglio, il metodo agganciato al servizio /destinatario/inoltro sfrutta l'oggetto MsgOnDbSmartdoc per salvare in modo persistente e correttamente il messaggio appena ricevuto nella base dati, ovvero estrae da tale messaggio il file XML rappresentante la Segnatura e lo converte, avvalendosi della libreria "javax.xml.bind", in un oggetto Java. Dopo questa operazione mediante la quale è stato possibile estrapolare tutte le informazioni necessarie (mittente, destinatario, oggetto, data, ecc.), si effettua il salvataggio dei file su documentale. L'unico documento di cui si possiede il contenuto è la Segnatura; perciò, essa viene salvata definitivamente sul database, mentre, in questa fase, del documento principale e degli eventuali allegati si dispone solo del denominativo e

dell'identificativo, rappresentato sottoforma di URL come concatenazione di vari campi presenti nel messaggio ricevuto.

Dunque, è necessario effettuare il download del contenuto dei documenti che si vogliono scambiare dal client, la scelta della strategia da implementare è stata effettuata dopo un'attenta analisi del problema da risolvere, nella quale sono state prese in considerazione tutte le esigenze e le criticità generate dalle operazioni da eseguire. Perciò dopo l'accurato studio di tutte le possibili alternative applicabili per realizzare il comportamento desiderato, si è optato per effettuare un primo salvataggio "falso" e agganciare, in un secondo momento, il contenuto del documento all'entry del database che lo rappresenta con l'ausilio di un job apposito, chiamato InteropJob che verrà descritto con maggiore dettaglio nel paragrafo successivo.

In particolare, le azioni eseguite sono due: come prima cosa si setta il flag "info", associato all'entry rappresentate il messaggio processato, con la stringa "TO DOWNLOAD", questa indicazione verrà utilizzata dall' InteropJob, dopodiché si effettua un salvataggio "falso" su documentale di tali file, ossia si memorizzano i file senza il loro contenuto, che verrà popolato in un secondo momento dal job sopra citato. Per chiarire meglio queste ultime operazioni viene riportata di seguito la parte di codice che si occupa di effettuarle.

```
// flag per il job che scarica i documenti
msgOnDB.setInfo("TO DOWNLOAD");

IdentificatoreType id_url = s.getIntestazione().getIdentificatore();
String url =id_url.getCodiceAmministrazione().getValue()+"/" + id_url.getCodiceAOO().getValue()+"/" +
           id_url.getCodiceRegistro()+"/" + id_url.getNumeroRegistrazione()+"/" + id_url.getDataRegistrazione()+"/" ;

// Salvataggio di doc e allegati "fake"
AttachmentOnDb doc = new AttachmentOnDb();
doc.setContentType(messaggioProtocollo.getRiferimentoPrimario().getMimetype());
doc.setContentDispositionFilename(messaggioProtocollo.getRiferimentoPrimario().getNomeFile());
doc.setContentTypeName(messaggioProtocollo.getRiferimentoPrimario().getNomeFile());
doc.setContentStream("");
doc.setInfo(url+messaggioProtocollo.getRiferimentoPrimario().getResourceId());
attachments.add(doc);
for(RiferimentoDocumento d : messaggioProtocollo.getRiferimentiAllegati()) {
    AttachmentOnDb all = new AttachmentOnDb();
    all.setContentTypeName(d.getMimetype());
    all.setContentDispositionFilename(d.getNomeFile());
    all.setContentTypeName(d.getNomeFile());
    all.setContentStream("");
    all.setInfo(url+d.getResourceId());
    attachments.add(all);
}
```

Figura 19. Settaggio del campo info, generazione URL e salvataggio "falso" dei file

L'ultimo metodo da analizzare è richiestaDocumento, che verrà invocato dall'InteropJob tramite una richiesta http GET. Sostanzialmente tale metodo, prima si occupa di salvare su database il messaggio appena ricevuto con l'ausilio dell'oggetto MsgOnDbSmartdoc, successivamente effettua una ricerca del documento richiesto sul database tramite l'id ricevuto come parametro passato all'interno dell'URL della richiesta ed infine invia il contenuto di tale file sottoforma di ByteArrayInputStream.

3.2.2.2 InteropJob

L'InteropJob è un job che, con una certa frequenza configurabile (per esempio ogni dieci minuti), si occupa di effettuare richieste all'endpoint mittente per ottenere il contenuto dei file di cui non è ancora stato eseguito il download. Tale classe viene implementata con l'ausilio di una libreria chiamata Quartz [Quartz]. In particolare, essa è una libreria open source per la gestione di job e attività pianificate in applicazioni Java. Offre una serie di funzionalità per automatizzare attività periodiche, consentendo agli sviluppatori di creare facilmente task che verranno eseguiti in base a un determinato schema temporale. Le caratteristiche principali di Quartz sono le seguenti:

- Pianificazione avanzata: permette di definire la pianificazione dei job utilizzando espressioni cron.
- Persistenza dei job: supporta la persistenza dei job e dei trigger, il che significa che le informazioni sulle attività programmate possono essere conservate anche dopo il riavvio dell'applicazione, garantendo l'affidabilità nella gestione delle attività.
- Supporto per il clustering: È possibile utilizzare Quartz in un ambiente di cluster, consentendo la distribuzione delle attività su più istanze dell'applicazione per garantire l'alta disponibilità e la scalabilità.
- Gestione degli eventi: offre un sistema di listener che consente agli sviluppatori di reagire agli eventi legati all'esecuzione dei job, come l'avvio e l'arresto dei task.
- Job con stato e senza stato: supporta job sia con stato che senza; perciò, è possibile mantenere o meno uno stato tra le esecuzioni. Questo offre una maggiore flessibilità nella gestione delle attività.
- Integrazione semplice: è facilmente integrabile in applicazioni Java.

Nello specifico l'InteropJob effettua, con cadenza periodica, le seguenti azioni:

1. Come prima operazione effettua due letture sul database, la prima per ottenere gli identificatori dei messaggi che presentano il flag "TO DOWNLOAD" nel campo "info" e la seconda per ricavare gli URL e gli identificativi dei documenti da scaricare. Nello specifico, l'URL verrà utilizzato per effettuare la richiesta al mittente ed indicare quale file si desidera, mentre l'id verrà adoperato per individuare la corretta entry da aggiornare nel database con il contenuto documento. La figura sottostante riporta la seconda lettura dalla base dati.

```
for ( Integer id : idMsg) {
    // Ottengo id e URL
    try (Connection con = ConnectionFactory.getConnection("jdbc/si4cs");
         PreparedStatement statURL = con.prepareStatement(
                 "SELECT a.INFO, ba.ID_DOCUMENTO, ba.ID_OGGETTO_FILE" +
                         "    FROM  ALLEGATI a, BINARY_ALLEGATI ba" +
                         "    WHERE a.MESSAGGIO = ? AND ba.ALLEGATO = a.ALLEGATO  AND a.INFO is NOT NULL");) {
        statURL.setInt(1, id);
        try (ResultSet rsURL = statURL.executeQuery()) {
            while (rsURL.next()) {
                urlAllegati.put(rsURL.getInt("ID_OGGETTO_FILE"), rsURL.getString("INFO"));
                idDoc.put(rsURL.getInt("ID_OGGETTO_FILE"), rsURL.getInt("ID_DOCUMENTO"));
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    // Altre oprazioni
}
```

Figura 20. Ottenimento URL e id dei documenti da scaricare

2. Successivamente, sempre tramite lettura dalla tabella “REGISTRO” del database, acquisisce l’endpoint del mittente a cui effettuare la richiesta http /mittente/documento.

3. Dopodiché effettua una richiesta http GET per ogni documento, sfruttando il framework Apache HttpClient [Apache_HttpClient] (descritto nel paragrafo successivo), per ottenere il contenuto di questo file dal mittente, come mostrato nell’immagine sottostante.

```
for (Integer idAll : urlAllegati.keySet()) {
    try (CloseableHttpClient httpClient = HttpClients.createDefault()) {
        // Crea una richiesta GET con l'URL del servizio REST
        HttpGet httpGet = new HttpGet(endpoint + "mittente/documento/" + urlAllegati.get(idAll));
        // Esegue la richiesta
        try (CloseableHttpResponse response = httpClient.execute(httpGet)) {
            // Ottiene il contenuto del file
            byte[] responseBody = EntityUtils.toByteArray(response.getEntity());
            // Altre operazioni
```

Figura 21. Richiesta http per acquisire il contenuto del file

4. Poi realizza l’aggiornamento dell’entry della base dati che rappresenta il file in questione, selezionandola tramite l’id corrispondente, avvalendosi delle API messe a disposizione dalla libreria “finmatica-smartdoc”, come illustrato nella sezione di codice riportata.

```
DocumentaleService documentaleService = DocumentaleFactory.creaDocumentaleServiceGdm(conGdm, "GDM");
Documento d = new Documento();
d.addChiaveExtra("ESCLUDI_CONTROLLO_COMPETENZE", "Y");
d.setId(idDoc.get(idAll).toString());
ArrayList componenti = new ArrayList<Documento.COMPONENTI>();
componenti.add(Documento.COMPONENTI.VERSIONI);
Documento doc= documentaleService.getDocumento(d, componenti);
File f = new File();
f.setId(idAll.toString());
f.setNome("Prova");
f.setInputStream(new ByteArrayInputStream(responseBody));
d.addFile(f);
documentaleService.salvaDocumento(d);
conGdm.commit();
```

Figura 22. Aggiornamento contenuto del documento

5. Infine, l'ultima operazione, consiste nel cambiare il flag associato al messaggio da "TO DOWNLOAD" a "DOWNLOADED", effettuando una query di update sul database.

3.2.3 Client

Il client che effettua l'invio delle richieste http per l'invocazione dei quattro servizi REST implementati lato server (inoltro, esitoInoltro, annullamentoDestinatario e annullamentoMittente) è stato inserito, come il client SOAP, nel già esistente contesto "Si4Cim" che realizza l'invio di ogni genere di messaggio. Il client non si occupa di inviare messaggi al metodo richiestaDocumento in quanto, come visto nel paragrafo precedente, sarà compito dell'InteropJob interrogarlo per ottenere il contenuto di tutti i documenti desiderati.

In particolare, in questa libreria, è stata inserita una nuova classe denominata PDNDInteropRestMessage che si occupa specificatamente dell'invio delle richieste http destinate al server per invocare i metodi realizzati.

3.2.3.1 PDNDInteropRestMessage

La logica con cui è stata implementata la classe PDNDInteropRestMessage è molto simile a quella con cui è stata realizzato il client SOAP visto in precedenza, ovviamente la differenza sostanziale sta nel fatto che verranno inviate delle richieste http, invece che delle buste SOAP.

Per inoltrare tale tipo di richieste si sfrutta il framework Apache HttpClient. Esso è un framework open source per la creazione di client http in Java, che fornisce una vasta gamma di funzionalità per effettuare richieste, gestire risposte e interagire con servizi web. Tra le sue caratteristiche principali vi sono la configurazione flessibile, la gestione di cookie, l'interazione con sistemi di autenticazione e il supporto per la gestione di connessioni persistenti.

Nello specifico, anche PDNDInteropRestMessage estende la classe astratta già presente in "Si4Cim" GenericMessage che rappresenta un messaggio generico. In particolare, tale classe effettua l'overriding di alcuni metodi quali:

- Metodi di setting, ossia setCsTag, setMessageId e setParam, che sono utilizzati per impostare i parametri per la configurazione iniziale della classe.
- Metodi che popolano vettori, ovvero addAttachment e addRecipient che servono rispettivamente per l'aggiunta di nuovi allegati e destinatari.
- Infine, il metodo send che contiene la logica di business vera e propria. Quello che si occupa dell'invio della richiesta http.

Come nel caso del client SOAP, quest'ultimo metodo come prima operazione ottiene l'URL del server a cui inviare la richiesta http, estraendolo dal vettore recipient. Successivamente, il servizio da invocare viene discriminato attraverso il nome del file XML inserito nel vettore attachment. Infatti, in questo scenario ci sarà sempre un solo file che verrà utilizzato per generare l'oggetto da inserire nella richiesta http:

- Sia "AnnullamentoMittente.xml" che "AnnullamentoDestinatario.xml" verranno trasformati in un oggetto RequestCancellation;
- "EsitoInoltro.xml" verrà convertito in un OutcomeForwarding;
- Infine, da un "MessaggioInoltro.xml" si otterrà un oggetto MessaggioProtocollo.

La struttura di questi oggetti è definita univocamente da AgID e la trasformazione di un file XML in un oggetto Java (unmarshalling) viene effettuata sfruttando la libreria "org.dom4j.io", vista anche per il client SOAP.

Perciò, dopo aver individuato la particolare richiesta da generare ed effettuare, si realizza l'unmarshalling del file XML in possesso e si ottiene l'oggetto Java corrispondente, successivamente questo oggetto viene serializzato in formato JSON per consentirne l'invio, questa operazione viene eseguita con l'ausilio della classe ObjectMapper. Nello specifico, essa è parte della libreria Jackson, ed è utilizzata in Java per la serializzazione e la deserializzazione di oggetti Java in e da formato JSON. A titolo di esempio nell'immagine sottostante si mostra la sezione di codice che rappresenta questi passaggi.

```
case "AnnullamentoDestinatario.xml":
RequestCancellation rc = new RequestCancellation();
try {
    rc.setIdentificatoreMittente( parseIdentificatore(document, "(//", ""));
    rc.setIdentificatoreDestinatario( parseIdentificatore(document,"(//",""));
    rc.setNote(document.selectSingleNode("//Motivo").getStringValue());
    rc.setRiferimentoProvvedimento(document.selectSingleNode("//Provvedimento").getStringValue());

} catch (DatatypeConfigurationException e ){
    e.printStackTrace();
}

String json = null;
ObjectMapper objectMapper = new ObjectMapper();
    try {
     json = objectMapper.writeValueAsString(rc);
} catch (Exception e) {
    e.printStackTrace();
}
```

Figura 23. Unmarshalling del file in un oggetto RequestCancellation e serializzazione di quest'ultimo

Infine, si procede all'invio della richiesta http POST al server per invocare il servizio desiderato, come rappresentato nella figura sottostante.

```
HttpClient httpClient = HttpClients.createDefault();
HttpPost httpPost = new HttpPost(url+"/mittente/annullamento/destinatario");

StringEntity requestEntity = new StringEntity(json, "application/json", "UTF-8");
httpPost.setEntity(requestEntity);

try {
    HttpResponse response = httpClient.execute(httpPost);
} catch (Exception e) {
    e.printStackTrace();
}
break;
```

Figura 24. Invio della richiesta http in POST

3.3 SICUREZZA

Il meccanismo di sicurezza che regola gli e-service implementati viene realizzato mediante l'ausilio di token JWT, detto anche voucher. Nel dettaglio, un token JWT (JSON Web Token) è un formato di token che viene comunemente utilizzato per rappresentare informazioni in modo sicuro tra due parti. Le sue caratteristiche principali sono le seguenti:

- Struttura: Un token JWT è una stringa codificata in formato JSON che consiste in tre parti separate da punti (“.”), ossia header, payload e firma:
    1. Header: specifica il tipo del token e l'algoritmo di firma utilizzato per proteggerlo. Questa parte è codificata in base64 e decodificabile.
    2. Payload: contiene le informazioni o i “claim” del token. Questi ultimi sono coppie chiave-valore che includono informazioni sull'entità (ad esempio utente) e altre metainformazioni. Alcuni claim sono predefiniti, come l’emittente (“iss”), la scadenza (“exp”) e il soggetto (“sub”), mentre altri possono essere personalizzati.
    3. Firma: è opzionale ma consigliata, essa protegge l'integrità del token e garantisce che non sia stato modificato durante il trasporto. Viene creata utilizzando un algoritmo di firma specificato nell'header e una chiave segreta.
- Codifica: queste tre parti del token vengono codificate in base64 per creare una stringa compatta e sicura.
- Utilizzo: sono ampiamente utilizzati per l'autenticazione e l'autorizzazione. Inoltre, vengono spesso inclusi come header Authorization nelle richieste HTTP per verificare l'identità di un utente o fornire accesso a risorse protette.
- Privi di stato: sono "stateless", il che significa che ogni richiesta può essere gestita indipendentemente e non richiede uno stato di sessione sul server. Questo li rende scalabili e adatti all'architettura delle applicazioni distribuite.
- Scadenza: hanno un tempo di scadenza, dopo il quale non sono più validi, indicato nel claim “exp” del payload.

In particolare, lato client viene generata un client assertion per ottenere dalla piattaforma PDND Interoperabilità un token JWT, se non se ne possiede uno già valido. Successivamente tale token viene inserito nell’header Authorization della richiesta SOAP o http. Una volta giunto il messaggio, lato server avverrà la validazione del token e se tutti i controlli daranno esito positivo si procederà ad effettuare l’operazione richiesta.

Perciò, per quanto concerne il discorso sicurezza, l’approccio utilizzato è il medesimo sia per gli e-service con protocollo SOAP sia per quelli realizzati con tecnologia REST.

3.3.1 Client

Per quanto riguarda l’implementazione del meccanismo di sicurezza per entrambi i client (SOAP e REST) è stata introdotta una classe denominata TokenPDND che si occupa di generare una client assertion e di inviarla alla piattaforma PDND per ottenere un token JWT, se invece se ne possiede già uno la classe si limita a restituirlo. Infine, una volta ottenuto il voucher, esso viene inserito all’intero della busta SOAP o della richiesta http in base al “tipo” di e-service che si sta utilizzando, per fare quest’ultima operazione le API sono leggermente diverse ma con il medesimo effetto.

```
TokenPDND tokenPDND = TokenPDND.getInstance();
Map<String, List<String>> header = new HashMap<>();
List<String> t = new ArrayList<>();
t.add("Bearer "+tokenPDND.getAccessToken(PDND_INTEROP_CHIAVE_REGISTRO));
header.put("Authorization", t);
```

Figura 25. Inserimento token JWT nell'header Authorization nel client SOAP

```
TokenPDND tokenPDND = TokenPDND.getInstance();
request.setHeader(HttpHeaders.AUTHORIZATION, "Bearer " + tokenPDND.getAccessToken(PDND_INTEROP_CHIAVE_REGISTRO));
```

Figura 26. Inserimento token JWT nell'header Authorization nel client REST

3.3.1.1 TokenPDND

Questa classe è stata realizzata avvalendosi del design pattern Singleton che garantisce una sola istanza della classe all'interno dell'applicazione e fornisce un unico punto di accesso globale a tale istanza. Per realizzare ciò, è stato utilizzato un costruttore privato, un campo statico a cui associare l'unica istanza e un metodo statico (getInstance) che restituisce l'istanza, alla prima volta invocazione esso la crea e alle successive restituisce sempre l'istanza esistente, come rappresentato nella figura sottostante.

```
private static TokenPDND INSTANCE;
// ...
private TokenPDND() { ... }
// ...
public static TokenPDND getInstance() {
    if(INSTANCE == null) {
        INSTANCE = new TokenPDND();
    }
    return INSTANCE;
}
```

Figura 27. Realizzazione del design pattern Singleton

Il metodo che si occupa del vero e proprio rilascio del token JWT è getAccessToken, esso come prima operazione va a verificare se sono già stati letti dal database, nella tabella REGISTRO, i parametri di client id e client assertion, in caso contrario effettua una lettura per ottenerli. Successivamente controlla se possiede già un token JWT e se esso non è scaduto, se l'operazione dà esito positivo restituisce il voucher in suo possesso altrimenti effettua una richiesta http POST alla piattaforma PDND Interoperabilità per "staccarne" uno nuovo e valido.

Per verificare la validità del token, si effettua una decodifica della stringa che lo rappresenta, poi si tratta quest'ultima come un oggetto JSON del quale si va ad estrarre il campo "exp" che ne rappresenta la scadenza. Questo campo contiene una data e un'ora in formato Unix Timestamp, ovvero un numero intero che rappresenta i secondi trascorsi dal 1° gennaio 1970, noto come "epoch". Perciò si va a confrontare il numero di secondi trascorsi fino a quel momento con il valore contenuto nel token per valutarne la validità, come rappresentato nella figura sottostante.

```
boolean isValid = false;
String[] chunks = accessToken.split("\\.");
Base64.Decoder decoder = Base64.getUrlDecoder();
String header = new String(decoder.decode(chunks[0]));
String payload = new String(decoder.decode(chunks[1]));
JSONParser parser = new JSONParser();
JSONObject jsonPayload = null;
try {
    jsonPayload = (JSONObject)parser.parse(payload);
} catch (ParseException e) {
    e.printStackTrace();
}
String exp = jsonPayload.get("exp").toString();
Long expLong = new Long(exp);
long input = (long) expLong.longValue()*1000;
if (Instant.ofEpochMilli(input).compareTo(Instant.now()) > 0) isValid = true;
return isValid;
```

Figura 28. Verifica della validità del token JWT posseduto

Per quanto concerne la richiesta di un nuovo token JWT perché non valido o non posseduto, si realizza una client assertion alla piattaforma PDND per ottenerne uno nuovo. In particolare, una client assertion è una tecnica di autenticazione utilizzata nel contesto di OAuth 2.0 per consentire a un'applicazione client di dimostrare la propria identità durante una richiesta.

Le client assertion sono rappresentate da un token crittograficamente firmato che viene inviato insieme a una richiesta OAuth 2.0. Tale token contiene informazioni che dimostrano l'identità dell'applicazione client, come in questo caso le "client_credentials", solitamente in formato JWT. Il server verifica la firma per confermare l'identità dell'applicazione client.

In questo scenario la client assertion viene inviata con l'ausilio del framework Apache HttpClient, già illustrato nei paragrafi precedenti, che permette di preparare ed inoltrare una richiesta http POST. La risposta è una stringa in formato JSON di cui si effettua il parsing e si estrae il campo "access_token" rappresentante il nuovo token JWT staccato dalla piattaforma PDND. Per maggiore chiarezza viene riportato nella figura sottostante la parte di codice che si occupa di effettuare questa operazione.

```
CloseableHttpClient httpClient = HttpClients.createDefault();
HttpPost httpPost = new HttpPost(urlAuthPDND);
httpPost.setHeader("Content-Type", "application/x-www-form-urlencoded");
List<NameValuePair> params = new ArrayList<NameValuePair>();
params.add(new BasicNameValuePair("client_id", PDND_INTEROP_clientId));
params.add(new BasicNameValuePair("client_assertion", PDND_INTEROP_clienAssertion));
params.add(new BasicNameValuePair("client_assertion_type", "urn:ietf:params:oauth:client-assertion-type:jwt-bearer"));
params.add(new BasicNameValuePair("grant_type", "client_credentials"));

httpPost.setEntity(new UrlEncodedFormEntity(params));

CloseableHttpResponse response = httpClient.execute(httpPost);
if (response.getStatusLine().getStatusCode() == 200) {
    HttpEntity entity2 = response.getEntity();
    String tokenResp = EntityUtils.toString(entity2);
    JSONParser parser = new JSONParser();
    JSONObject jsonResp = (JSONObject) parser.parse(tokenResp);
    PDND_INTEROP_accessToken = jsonResp.get("access_token").toString();
}
```

Figura 29. Client assertion e parsing della risposta per estrarre il nuovo token

3.3.2 Server

Lato server, non avendo a disposizione Spring Boot oppure possedendo una versione datata di tale framework, non si è potuto usufruire delle funzionalità offerte da Spring Security.

È doveroso sottolineare come la scelta di questa strategia abbia complicato leggermente l'implementazione della validazione e verifica del token in quanto con l'ausilio di Spring Security tali operazioni erano demandate e a carico del framework Spring, che si occupava in maniera autonoma di effettuare tutti i controlli necessari. Mentre effettuando una validazione personalizzata tutti i controlli sono a carico del programmatore che ha l'incarico di decodificare il token ricevuto, trattarlo come un oggetto JSON, verificare la firma e controllare la validità di alcuni campi. In sostanza con questo approccio il programmatore è costretto a lavorare a più basso livello, aumentando così la complessità del suo lavoro ma ottenendo la compatibilità con entrambe le versioni degli e-service implementati.

Quindi sia per il server SOAP che per quello REST si è effettuata una validazione personalizzata del voucher ricevuto, utilizzando la medesima logica di business. L'unica differenza si ha nel modo in cui si ottiene il token JWT contenuto nell'header Authorization: nel caso SOAP ci si è sfruttato un interceptor (JWTAuthorizationInterceptor) grazie all'utilizzo di Spring Boot, mentre nello scenario REST è stata utilizzata l'annotazione @HeaderParam("Authorization") che consente di estrapolare l'header Authorization e di conseguenza il token JWT presente al suo interno. Una volta ottenuta la stringa rappresentante il voucher, le verifiche su di esso sono state effettuata nella medesima maniera.

Infine, per entrambe le implementazioni è stata introdotta la possibilità di poter abilitare/disabilitare il meccanismo di sicurezza che si occupa della validazione e verifica

del token JWT mediante il parametro booleano “VALIDAZIONE_TOKEN_ATTIVA” che viene letto dalla tabella REGISTRO del database all’avvio dell’applicazione.

3.3.2.1 Verifica e validazione del token JWT

I passi necessari per effettuare la verifica e validazione del token JWT ricevuto dal client sono i seguenti:

1. Ottenere la chiave pubblica del client dalla piattaforma PDND mediante il kid (Key ID) presente nell’header del voucher;
2. Verificare la firma del token JWT con l’ausilio della chiave pubblica ottenuta al passo precedente;
3. Controllare che il token sia ancora valido e quindi non scaduto tramite il campo “exp”;
4. Effettuare il controllo sul tipo del voucher indicato nel campo “typ”.

Quindi come prima azione è necessario estrarre il kid dall’header del token JWT; perciò, si effettua la decodifica e il parsing dell’header per ottenere tale informazione.

Successivamente si ottiene la lista di chiavi pubbliche in uso sulla piattaforma rappresentata da un file JSON esposto nella cartella “.well-known” di PDND Interoperabilità, per acquisirla si effettua una richiesta http GET. Ricevuto il file si realizza il parsing e si ottiene l’oggetto JSON corrispondente, poi si ricerca la chiave corrispondente al kid estrapolato alla prima operazione. Per maggior chiarezza viene riportata di seguito la porzione di codice che si occupa di effettuare quest’ultima azione.

```
JSONParser jsonP = new JSONParser();
JSONObject jsonObj = (JSONObject) jsonP.parse(response.toString());

//Ricerca chiave pubblica
JSONArray keysArray = (JSONArray) jsonObj.get("keys");
for (Object keyObject : keysArray) {
    if (keyObject instanceof JSONObject) {
        JSONObject _key = (JSONObject) keyObject;
        String _kid = (String) _key.get("kid");
        if (_kid != null && _kid.equals(kid)) {
            String nValue = (String) _key.get("n");
            String eValue = (String) _key.get("e");
            if (nValue != null && eValue != null) {
                mod = nValue;
                esp = eValue;
            } else {
                LOG.info("Chiave nulla!");
            }
            break;
        }
    }
}
```

Figura 30. Ricerca della chiave pubblica associata allo specifico kid

Una volta identificata la chiave pubblica del cliente si procede con la verifica della firma, grazie alle classi e ai metodi messi a disposizione dalla libreria “java.security”. In particolare, essa è una parte fondamentale dell'API Java Standard Edition ed è responsabile della gestione di aspetti legati alla sicurezza in applicazioni Java.

In sintesi, “java.security” è un componente chiave per implementare la sicurezza in applicazioni Java, proteggere i dati sensibili e garantire l'integrità delle operazioni. Essa fornisce un insieme di strumenti e API che consentono agli sviluppatori di creare applicazioni sicure e affidabili in ambienti dove la sicurezza è una preoccupazione prioritaria. Di seguito la parte di codice che utilizza questi metodi per effettuare la verifica della firma.

```
byte espB[] = Base64.getUrlDecoder().decode(esp);
byte modB[] = Base64.getUrlDecoder().decode(mod);
BigInteger bigEsp = new BigInteger(1,espB);
BigInteger bigMod = new BigInteger(1,modB);

// Chiave pubblica
PublicKey publicKey  = KeyFactory.getInstance("RSA").generatePublic(new RSAPublicKeySpec(bigMod, bigEsp));

// Verifica Firma
String signedData = token.substring(0, token.lastIndexOf("."));
String signatureB64u = token.substring(token.lastIndexOf(".")+1,token.length());
byte signature[] = Base64.getUrlDecoder().decode(signatureB64u);

Signature sig = Signature.getInstance("SHA256withRSA");
sig.initVerify(publicKey);
sig.update(signedData.getBytes());
if(!(sig.verify(signature))) throw new Exception("Signature non verificata");
```

Figura 31. Verifica della firma presente nel token JWT

Gli ultimi controlli sono sulla scadenza (exp) e sul tipo (typ). Nello specifico, visto che il token JWT è già stato decodificato, è sufficiente effettuare il parsing del body e trattarlo come un oggetto JSON dal quale vengono estratti i valori dei campi “exp” e “typ”. La scadenza viene controllata con la stessa strategia utilizzata lato client nella classe TokenPDND, mentre il valore del tipo deve sempre corrispondere alla stringa “at+jwt”.

4 VALIDAZIONE SPERIMENTALE E RISULTATI DI PERFORMANCE

L'ultima fase del progetto realizzato si è sviluppata nel testare le performance degli e-service appena implementati. L'aspetto su cui ci si è concentrati maggiormente è il tempo di trasferimento dei documenti variando aspetti differenti.

Sono state realizzate diverse varietà di test che cambiano in base alla tecnologia adoperata per l'implementazione, questo per il fatto che, come già detto nel capitolo precedente, i due e-service presentano un flusso di trasferimento dei documenti opposto, infatti:

- Nell'e-service SOAP è il client che effettua un upload dei file; perciò, il client inserisce nella richiesta SOAP che invia al server i documenti che desidera scambiare.
- Nell'e-service REST è il server che, dopo aver ricevuto gli identificativi dal client nell'interazione precedente, effettua il download dei file; quindi, il server riceve gli id e in un secondo momento, utilizzando l'InteropJob, effettua un certo numero di richieste http GET al client pari al numero di documenti da scaricare per ottenere il loro contenuto.

Per i motivi appena citati, si è deciso di effettuare tre tipologie di test, in particolare:

- Tempo necessario per eseguire l'upload di un file al variare della sua dimensione;
- Tempo impiegato per l'invio di un messaggio al variare del numero di file presenti;
- Tempo necessario per effettuare il download di un documento al variare della sua dimensione.

I primi due sono stati realizzati sull'e-service SOAP, mentre l'ultimo sull'e-service REST.

Prima di descrivere nel dettaglio i risultati ottenuti è necessario fare chiarezza su alcuni aspetti. Nello specifico, il tempo è stato misurato come di Round Trip Time (RTT), ovvero la somma del tempo impiegato per inviare i dati dal mittente al destinatario e quello necessario per ricevere una risposta. Il RTT è generalmente misurato in millisecondi (ms) ed è un parametro importante per valutare le prestazioni di una connessione di rete.

Infine, come ambiente di test è stato utilizzato un server aziendale, perciò sono state simulate le condizioni reali di una comunicazione distribuita con client e server installati su due macchine diverse.

4.1 E-SERVICE SOAP

Per realizzare i test sull'e-service SOAP si è sfruttato lo strumento SoapUI [SoapUI] per simulare nella macchina locale il comportamento del client, ovvero quello di preparare ed inviare la busta SOAP contenente le informazioni necessarie e i documenti che si desidera scambiare.

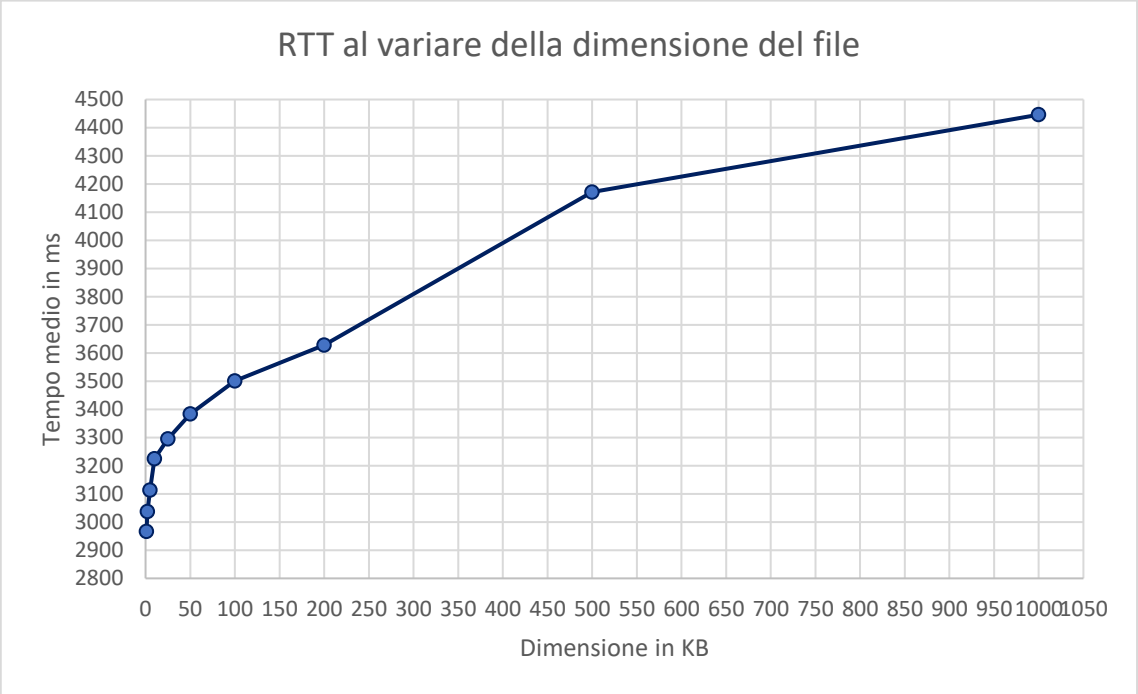
Nello specifico, SoapUI è uno strumento di testing e automazione progettato per semplificare il testing di servizi web, in particolare quelli basati su protocolli SOAP. Permette agli sviluppatori e ai tester di creare, eseguire e automatizzare test su servizi web in modo efficiente. SoapUI offre funzionalità di generazione di richieste, analisi delle risposte (tra i vari parametri anche il RTT), monitoraggio dei servizi e reportistica dettagliata.

4.1.1 RTT al variare della dimensione del file inviato

Per effettuare questa tipologia di test sono stati creati file di diverse dimensioni, in particolare: 1 KB, 2 KB, 5 KB, 10 KB, 25 KB, 50 KB, 100 KB, 200 KB, 500 KB e 1000 KB (ovvero 1 MB). Il test è stato ripetuto tre volte per ogni dimensione, facendo poi una media dei RTT misurati nei vari tentativi. I risultati ottenuti sono riportati nella tabella sottostante.

	Tentativo numero (ms)			
Dimensione (KB)	**1**	**2**	**3**	**Tempo medio (ms)**
1	2977	2964	2959	2967
2	3041	3034	3037	3037
5	3115	3111	3114	3113
10	3229	3225	3220	3225
25	3294	3297	3293	3295
50	3388	3385	3380	3384
100	3504	3499	3499	3501
200	3631	3626	3628	3628
500	4174	4169	4172	4171
1000	4443	4449	4445	4446

Successivamente i dati sono stati inseriti in un grafico per permetterne una lettura e una visualizzazione più chiara e migliore.



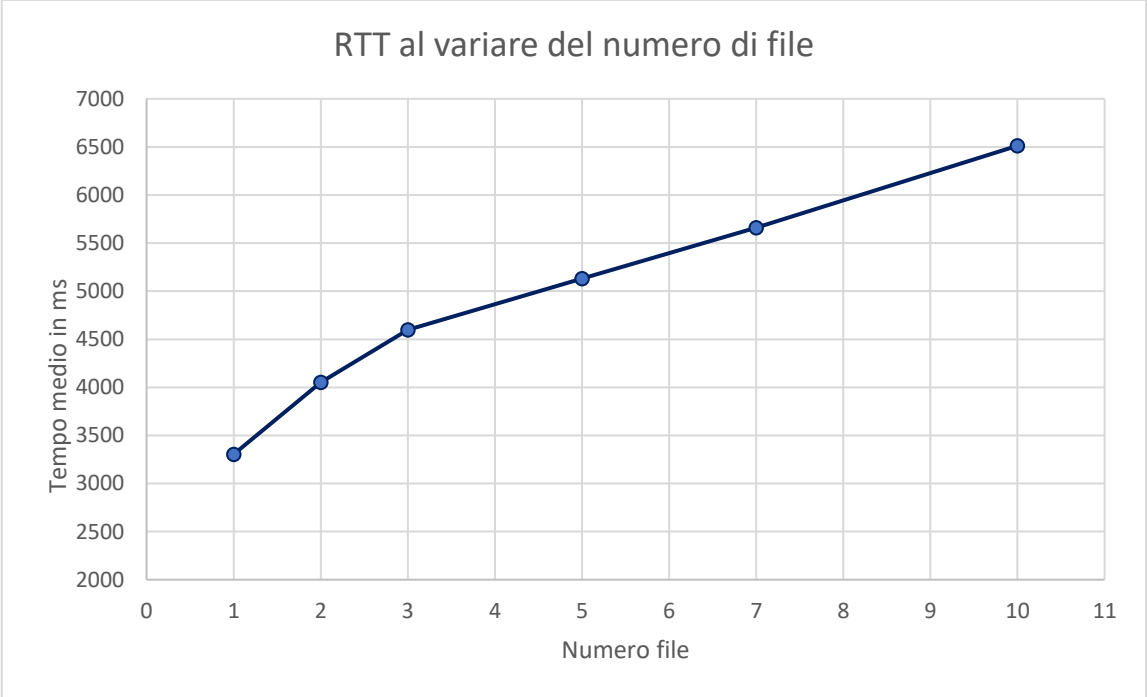
L'andamento ottenuto è affine a quello atteso, infatti all'aumentare della dimensione del file vi è un aumento del RTT. Questo è un comportamento intuitivo: trasferire file più grandi richiede più tempo. Come si può evincere dal grafico, la relazione che vi è tra queste due grandezze non è lineare; infatti, il RTT cresce in modo più lento man mano che la dimensione del file aumenta.

4.1.2 RTT al variare del numero dei file da inviare

Per realizzare questo test è stata scelta una dimensione fissa dei file da inviare. Nello specifico, dopo un'analisi delle tipologie di documenti che verranno scambiate tra le varie AOO in uno scenario reale, si è optato per utilizzare file da 25 KB perché rappresentano la dimensione media dei documenti che verranno trasferiti. Successivamente è stato scelto di effettuare il test con un numero di file che rappresenta un caso realistico, in dettaglio: 1, 2, 5, 7 e 10 documenti. La prova è stata ripetuta tre volte per ogni numero di file, realizzando poi una media dei RTT misurati nei vari tentativi. I risultati ottenuti sono riportati nella tabella sottostante.

	Tentativo numero (ms)			
Numero file	1	2	3	Tempo medio (ms)
1	3309	3305	3299	3304
2	4053	4048	4052	4051
3	4601	4598	4596	4598
5	5130	5128	5131	5130
7	5664	5658	5660	5661
10	6513	6510	6512	6512

I dati poi sono stati inseriti all'interno di un grafico a dispersione per consentirne una miglior lettura e comprensione.



Come si può notare dal grafico soprastante, i risultati ottenuti sono in linea con quanto atteso, infatti, logicamente, all'aumentare del numero di file da trasferire aumenta il RTT. Inoltre, si può vedere come l'andamento sia molto simile ad una relazione lineare tra le due grandezze. Nel caso di alcune prove, il RTT è leggermente superiore o inferiore rispetto a quello che ci si aspetterebbe, queste variazioni possono essere il risultato di vari fattori, come il traffico presente nella rete in quel particolare momento o la presenza di altre attività concorrenti sulla macchina server.

4.2 E-SERVICE REST

Per effettuare i test sull'e-service si è utilizzata l'applicativo Insomnia [Insomnia] per simulare le operazioni che verrebbero effettuate dall'InteropJob, ossia generare ed effettuare la richiesta http GET al client per realizzare il download del file con l'ausilio dell'id in possesso.

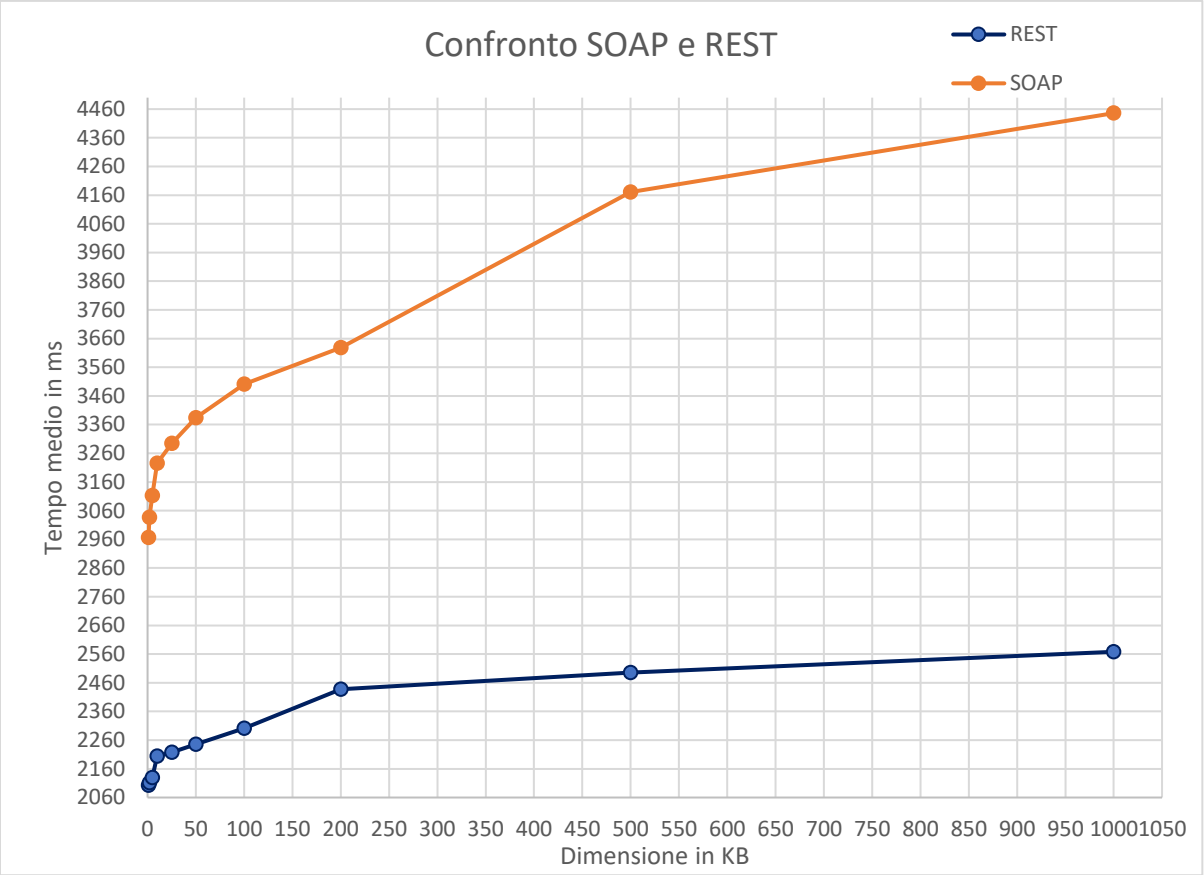
Nello specifico, Insomnia è un'applicazione software progettata per agevolare lo sviluppo, il test e il debug di API REST. Essa permette agli sviluppatori di creare, organizzare e testare facilmente richieste http per poter verificare il comportamento dei servizi REST realizzati. Inoltre, fornisce molte informazioni riguardanti la richiesta effettuata e la risposta ricevuta, tra queste anche il RTT.

4.2.1 RTT al variare della dimensione del file da scaricare

Per effettuare questo test sono stati utilizzati i medesmi file di diverse dimensioni adoperati nel test precedente per l'e-service SOAP, in particolare: 1 KB, 2 KB, 5 KB, 10 KB, 25 KB, 50 KB, 100 KB, 200 KB, 500 KB e 1000 KB (ovvero 1 MB). Anche in questo caso, la richiesta è stata ripetuta tre volte per ogni dimensione, facendo poi una media dei RTT misurati nei vari tentativi. I risultati ottenuti sono riportati nella tabella sottostante.

	Tentativo numero (ms)			
Dimensione (KB)	**1**	**2**	**3**	**Tempo medio (ms)**
1	2107	2102	2099	2103
2	2110	2114	2111	2112
5	2129	2130	2127	2129
10	2206	2204	2201	2204
25	2216	2216	2221	2218
50	2251	2242	2246	2246
100	2302	2298	2302	2301
200	2435	2438	2439	2437
500	2498	2494	2495	2496
1000	2566	2569	2568	2568

Il primo aspetto che si nota, come evidenziato anche del grafico sottostante, è che file delle medesime dimensioni impiegano un tempo decisamente minore per essere trasferiti tramite la tecnologia REST piuttosto che con l'ausilio del protocollo SOAP.

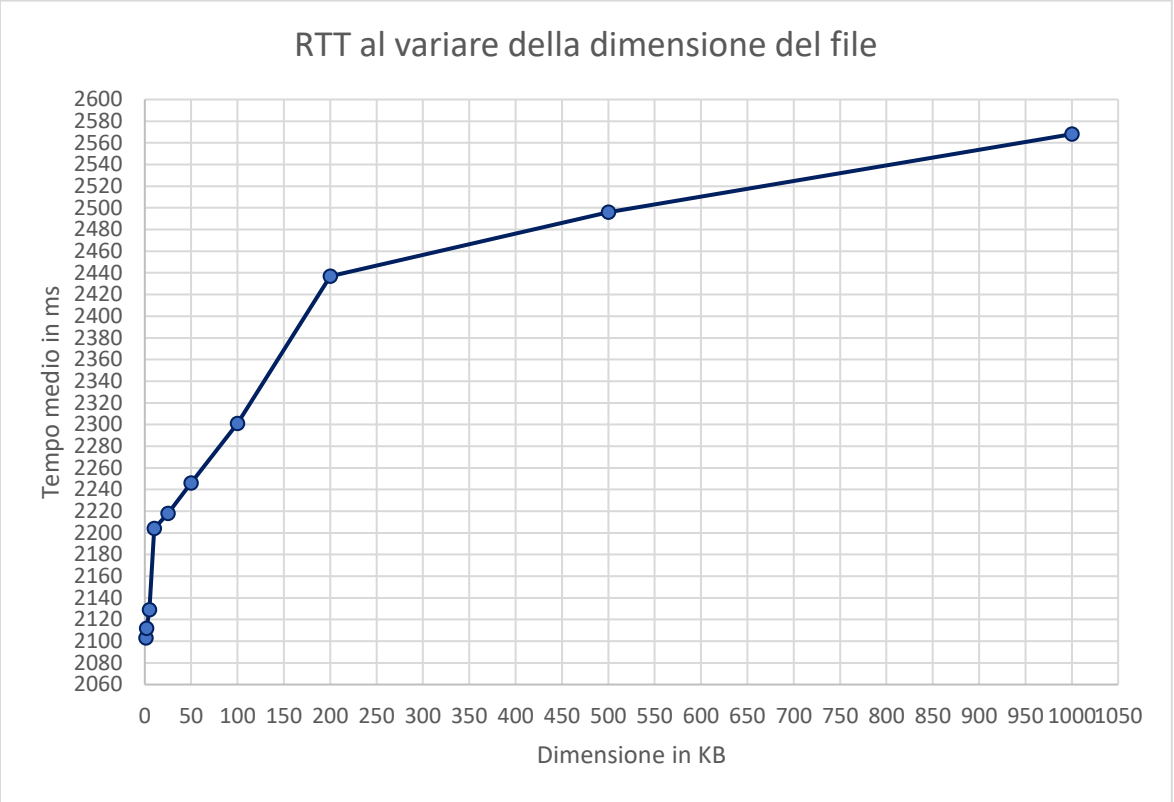


Le ragioni sono varie, in particolare:

- Struttura più “leggera” di REST: esso utilizza spesso formati di scambio leggeri come JSON, a differenza di SOAP che usa XML in modo più strutturato e complesso. JSON è solitamente più efficiente in termini di dimensioni dei dati rispetto a XML, riducendo la quantità di dati da trasferire e accelerando il processo.
- Minore Overhead: SOAP richiede un maggiore numero di informazioni di controllo e può essere più verboso, mentre REST è spesso più conciso e mirato.
- Uso di Metodi http standard: REST sfrutta i metodi http standard per interagire con le risorse. Questi metodi sono ben supportati e ottimizzati nei server http, il che può contribuire ad una maggiore efficienza nelle operazioni di trasferimento di file.
- Flessibilità: REST è spesso considerato più flessibile perché non impone uno schema rigoroso come fa SOAP. Ciò consente agli sviluppatori di progettare servizi REST in base alle esigenze specifiche.

Tuttavia, è importante notare che la scelta tra REST e SOAP dipende anche dalle esigenze specifiche dell'applicazione e dei dati da trasferire. REST è spesso preferito per scenari in cui la leggerezza e la flessibilità sono cruciali, ma SOAP può essere più appropriato in contesti che richiedono maggiore rigidità e sicurezza.

I dati ottenuti sono stati poi inseriti all'interno di un grafico per consentirne una miglior lettura e poter arrivare ad ulteriori conclusioni.



Come nel caso del test effettuato per l'e-service SOAP, l'andamento è vicino a quello atteso; infatti, all'aumentare della dimensione del file da scaricare si verifica un aumento del RTT. Inoltre, non vi è una relazione lineare tra le due grandezze; infatti, il RTT cresce in maniera più lenta man mano che la dimensione del file aumenta.

4.3 ASSEVERAZIONE

Con il termine asseverazione si intende l'atto formale di attestazione di veridicità. Nel contesto di procedure e verifiche, un asseveratore è una persona incaricata di attestare ufficialmente la validità o la conformità di determinate informazioni, documenti o processi. L'asseverazione può coinvolgere la verifica di fatti, l'attestazione della conformità a determinati standard, come in questo caso, o l'affermazione della correttezza di una dichiarazione. In molti casi, l'asseverazione è sottoposta a procedure rigorose per garantire la credibilità e la precisione dell'attestazione.

Perciò, una volta completati e testati gli e-service si avviano le fasi di verifica di conformità tecnica, in questa fase non si possono più modificare i dati di progetto. Queste verifiche vengono effettuate dal Dipartimento per la trasformazione digitale [DTD] e possono includere sia controlli automatici di conformità con sistemi esterni che verifiche effettuate da un soggetto terzo asseveratore.

Le verifiche di conformità tecnica possono portare a tre esiti:

- Positivo: tutti i controlli vengono superati, consentendo di avanzare con la richiesta di erogazione del finanziamento.
- Parzialmente positivo: le verifiche non confermano la piena conformità tecnica, ma i termini non sono ancora scaduti.
- Negativo: i controlli non vengono superati e i termini sono scaduti, oppure il progetto risulta non ammissibile. A questo punto, il Dipartimento avvia il procedimento di revoca del finanziamento.

Gli e-service visti nei capitoli precedenti hanno subito il processo di asseverazione e i controlli di conformità tecnica su di essi hanno dato esito positivo; quindi, è stato possibile procedere con la richiesta di erogazione del finanziamento da parte del Dipartimento.

CONCLUSIONI

Alla fine di questo percorso si può affermare che l'obiettivo principale prefissato per il tirocinio e la tesi è stato ampiamente raggiunto: l'e-service che consente la comunicazione tra Aree Organizzative Omogenee di Documenti Amministrativi Protocollati è stato studiato, realizzato, testato e opera correttamente. In aggiunta, esso ha ottenuto anche l'asseverazione, risultando così conforme agli standard definiti da AgID. Perciò, Gruppo Finmatica ha attualmente a disposizione un nuovo prodotto, sviluppato sia in tecnologia REST che tramite il protocollo SOAP, che può essere introdotto nel mercato a tutti gli effetti.

Il fatto che l'e-service sia stato implementato con entrambe le tecnologie consente al cliente di poter scegliere quella che soddisfa maggiormente le sue esigenze. Nello specifico, REST con la sua architettura leggera e basata su standard web, semplifica l'accesso e la fruizione del servizio, garantendo così una maggiore agilità nell'interscambio di documenti. Mentre SOAP assicura elevati standard di sicurezza e affidabilità nelle comunicazioni; infatti, viene scelto per applicazioni aziendali complesse con requisiti avanzati e standardizzazione più rigorosa. Questo conferisce maggiore libertà al cliente, che, in questo modo, può scegliere la soluzione migliore per lo scenario in cui deve andare ad operare il servizio.

Inoltre, l'e-service sviluppato rappresenta una concreta alternativa al già esistente meccanismo della Posta Elettronica Certificata (PEC) che, prima dell'introduzione di quest'ultimo, rappresentava l'unico mezzo per la comunicazione tra AOO di Documenti Amministrativi Protocollati. Ciò rappresenta un notevole progresso, perché non solo modernizza il processo di scambio dei documenti, ma ne potenzia anche l'efficienza, la sicurezza e la versatilità, rappresentando una prospettiva innovativa. Infatti, la PEC ha sicuramente introdotto benefici nell'ambito delle comunicazioni elettroniche, ma presenta alcuni aspetti negativi che sono emersi nel corso del tempo, tra cui:

- La complessità procedurale e le formalità amministrative possono risultare onerose, rappresentando una barriera all'adozione diffusa.
- L'accesso ai servizi comporta dei costi.
- I formati di messaggistica della PEC sono spesso più complessi e restrittivi, generando limitazioni tecniche nella gestione dei documenti.
- Le opzioni di personalizzazione e flessibilità della PEC sono limitate rispetto a nuove tecnologie emergenti.
- La natura regolamentata e standardizzata della PEC potrebbe rallentare l'adozione di nuove tecnologie e metodologie più moderne, limitando così l'innovazione.

Tutti questi aspetti evidenziano come fosse necessario l'introduzione di un servizio evoluto rispetto al consolidato meccanismo delle PEC.

Infine, come si può evincere anche dal capitolo precedente, i test effettuati sulle prestazioni in fase di trasferimento dei documenti su entrambi gli e-service hanno prodotto risultati molto positivi, a testimonianza dell'adeguatezza delle scelte progettuali e dell'efficacia delle soluzioni implementate. Attestando così che il prodotto sviluppato, insieme alle tecnologie adottate, rappresenta un'ottima alternativa ai meccanismi già presenti.

BIBLIOGRAFIA

[Gruppo_Finmatica] Gruppo Finmatica, https://www.ads.it/

[AgID] Agenzia per l'Italia Digitale, https://www.agid.gov.it/

[PNRR] Piano Nazionale di Ripresa e Resilienza, https://www.italiadomani.gov.it/content/sogei-ng/it/it/home.html

[PDND] Piattaforma PDND, https://developers.italia.it/it/pdnd/

[Infrastruttura_ Interoperabilità_PDND] Linee guida di AGID (Agenzia per l'Italia Digitale), https://www.agid.gov.it/it/agenzia/stampa-e-comunicazione/notizie/2021/12/15/infrastruttura-interoperabilita-pdnd-agid-adotta-linee-guida

[Allegato_2] ALLEGATO 2: Pubblicazione e fruizione delle API, https://docs.italia.it/AgID/documenti-in-consultazione/lg-pdnd-docs/it/bozza/doc/02_Allegato%202/index.html

[REST] Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California, Irvine. https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[SOAP] World Wide Web Consortium (W3C), Simple Object Access Protocol (SOAP) 1.2. https://www.w3.org/TR/soap12/

[RFC_7519] M. Jones, Microsoft, J. Bradley, N. Sikimura, "JSON Web Token (JWT)", May 2015, https://datatracker.ietf.org/doc/html/rfc7519

[RFC_6750] M. Jones, Microsoft, D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", Oct. 2012, https://datatracker.ietf.org/doc/html/rfc6750

[RFC_7521] B. Campbell, C. Mortimore, M. Jones, Y. Goland, "Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants", May 2015, https://datatracker.ietf.org/doc/html/rfc7521

[Allegato_6] Allegato 6 - Comunicazione tra AOO di Documenti Amministrativi Protocollati, May 2021, https://www.agid.gov.it/sites/default/files/repository_files/all.6_comunicazione_tra_aoo_di_documenti_amministrativi_protocollati.pdf

[Segnatura_protocollo] segnatura_protocollo.xsd, https://github.com/AgID/protocollo-comunicazione-aoo/blob/update-05-2023/segnatura_protocollo.xsd

[Messaggio_protocollo] messaggio_protocollo.xsd, https://github.com/AgID/protocollo-comunicazione-aoo/blob/update-05-2023/messaggio_protocollo.xsd

[IPA] Indice dei domicili digitali della Pubblica Amministrazione e dei Gestori di Pubblici Servizi https://indicepa.gov.it/ipa-portale/

[protocollo_mittente] protocollo_mittente.wsdl, https://github.com/AgID/protocollo-comunicazione-aoo/blob/update-05-2023/interfaces_SOAP/protocollo-mittente.wsdl

[protocollo_destinatario] protocollo_destinatario.wsdl, https://github.com/AgID/protocollo-comunicazione-aoo/blob/update-05-2023/interfaces_SOAP/protocollo-destinatario.wsdl

[Eclipse] Eclipse Foundation. Eclipse IDE. https://www.eclipse.org/

[WTP] Web Tools Platform (WTP) plugin for Eclipse, https://www.eclipse.org/webtools/

[Spring_Boot] Spring Boot Project, Spring Boot, https://spring.io/projects/spring-boot

[protocollo-aoo.yaml] protocollo-aoo.yaml, May 2023, https://github.com/AgID/protocollo-comunicazione-aoo/blob/update-05-2023/interfaces_REST/openapi/protocollo-aoo.yaml

[Quartz] Quartz Scheduler: Open-Source Job Scheduling**,** Sito Web Ufficiale, http://www.quartz-scheduler.org/

[Apache_HttpClient] Apache HttpClient: Apache Software Foundation, https://hc.apache.org/httpcomponents-client-4.5.x/index.html

[SoapUI] SmartBear, "SoapUIDocumentation", https://www.soapui.org/documentation/

[Insomina] Insomnia REST Client. 2022. https://insomnia.rest/

[DTD] Dipartimento per la trasformazione digitale. https://innovazione.gov.it/

RINGRAZIAMENTI

In primis non posso che ringraziare i miei genitori. Mia mamma, l'amica su cui sempre contare per uno sfogo o un buon consiglio, grazie per tutte le materie che mi ha ascoltato ripetere e per il sostegno che mi hai sempre dimostrato. A mio padre, o meglio mio babbo, per tutte le cose che non ho fatto in tempo a chiedere che già erano state fatte e per tutti quei gesti semplici ma allo stesso tempo fondamentali. Siete il mio esempio e le mie fondamenta, per tutto quello che avete fatto, fate e farete i grazie non saranno mai abbastanza!

Un grazie alle nonne Sandra e Maria che, a loro modo, non hanno mai fatto mancare il loro supporto. E ai nonni Carlo ed Ermanno che da lassù staranno festeggiando con due risate accompagnate da un buon rosso.

Un ringraziamento anche ai miei amici per avermi ricordato che l'importante nella vita è versare i contributi. Per non avermi mai fatto mancare il sostegno e per avermi fatto capire che ogni tanto la soluzione migliore, anche nelle situazioni peggiori, è prendere tutto alla leggera e con un sorriso.

Un grazie ai miei compagni di avventura, Fabione e Coma, con i quali abbiamo trovato sempre l'intuizione per rendere meno monotono e più piacevole questo viaggio. Algise è fiero di noi!

Infine, non può mancare il ringraziamento alla persona e in particolare alle sue parole quando ho scelto di intraprendere questa facoltà: *“Cosa ci vai a fare? Tanto a te l’informatica non piace, non duri più di un mese.”* Queste hanno sempre rappresentato per me una delle motivazioni per arrivare all'obiettivo e terminata questa esperienza non posso che esserne entusiasta e affermare che è stata la scelta migliore che potessi mai fare.

Si è chiuso un capitolo e direi che è terminato nel migliore dei modi. Ora si gira pagina e si comincia una nuova avventura sempre con l'aspirazione di puntare il più in alto possibile, perché il motto è e rimarrà sempre: *"Ma ti giuro che da sempre io punto all'eccellente, se devo avere poco scelgo di avere niente".*