ARPACAL **REGIONECALABRIA** **REPUBBLICA ITALIANA**

ALLEGATO 016

LINEA GUIDA AGID INTEROPERABILITA' DEI SISTEMI (versione maggio 2019)

AGID Agenzia per l'Italia Digitale

Linee Guida Modello Interoperabilita

Release Bozza in consultazione

italia

16 mag 2019

Indice

Consultazione pubblica

La consultazione pubblica per questo documento è attiva dal 16/05/2019 al 14/06/2019.

Questo documento raccoglie il testo delle linee guida del *Modello di interoperabilità per la Pubblica Amministrazione*, disponibile per la consultazione pubblica.

Leggi le istruzioni per la consultazione

Vai al testo delle linee guida:

CAPITOLO 1

Istruzioni per la consultazione pubblica

1.1 Informazioni sulla consultazione

- **Durata della consultazione:** dal 16/05/2019 al 14/06/2019
- **Settore:** ICT

1.2 Esiti della consultazione

I risultati della consultazione pubblica on line saranno presi in considerazione dall'Agenzia per l'Italia Digitale[39] per la redazione del testo definitivo della Guida Tecnica.

1.3 Destinatari

Tutte le pubbliche amministrazioni che si trovano a sviluppare o mantenere un parco applicativo da utilizzare nell'ambito dei propri compiti istituzionali che richieda interazione con altre pubbliche amministrazioni o soggetti terzi (cittadini e imprese), gli operatori del mercato ICT (aziende, sviluppatori, integratori, etc) e tutti gli addetti ai lavori o gli interessati al tema.

1.4 Obiettivo della consultazione

Il documento rappresenta l'aggiornamento delle linee guida sul tema della cooperazione applicativa (SPCoop)[40], pubblicate dall'allora CNIPA.

Le linee guida sono redatte per l'attuazione del Codice dell'Amministrazione Digitale ai sensi dell'art. 71[79] del Codice stesso.

L'esigenza di aggiornare le precedenti linee guida nasce:

[39] http://www.agid.gov.it/
[40] http://www.agid.gov.it/agenda-digitale/infrastrutture-architetture/sistema-pubblico-connettivita/cooperazione-applicativa
[79] http://cad.readthedocs.io/it/v2017-12-13/_rst/capo7_art71.html

- dalle novità introdotte, anche sotto l'aspetto strategico, dal «Piano Triennale per l'informatica nella Pubblica amministrazione 2019-2021»[102];
- dal tempo trascorso dall'ultima revisione del documento, in relazione alla rapidità con cui notoriamente evolve il settore dell'ICT e gli standard disponibili;

1.5 Come partecipare

Le linee guida sul Modello di Interoperabilità per le PA sono pubblicate su Docs Italia[103] ed è possibile commentarle su Forum Italia[104].

È possibile inviare i propri commenti fino al 14/06/2019.

[102] http://pianotriennale-ict.readthedocs.io/it/latest/
[103] http://lg-modellointeroperabilita.readthedocs.io
[104] https://forum.italia.it/c/documenti-in-consultazione/linee-guida-modello-di-interoperabilita

CAPITOLO 2

Presentazione del Modello di Interoperabilità

La visione generale del Modello di Interoperabilità, considerando il contesto Europeo, introduce gli elementi che saranno considerati e le modalità con cui si provvederà al costante aggiornamento dello stesso.

2.1 Introduzione

Il Modello di Interoperabilità[1] (nel seguito in breve ModI) rappresenta il modello di supporto alla strategia di interoperabilità e cooperazione tra le Pubbliche Amministrazioni (di seguito PA), che definendo i contesti di interazione e integrazione tra le PA, i cittadini e le imprese permette di vedere la PA nella sua interezza come un unico sistema informativo (virtuale).

La definizione del ModI deve essere coerente con il nuovo *European Interoperability Framework* (EIF) oggetto della Comunicazione COM (2017)134[1052] della Commissione Europea del 23 marzo 2017, al fine di assicurare anche l'interoperabilità nel contesto Europeo e per l'attuazione del *Digital Single Market* (Mercato Unico Digitale).

Gli obiettivi del nuovo Modello di Interoperabilità sono:

- definire le modalità di integrazione tra le PA;
- armonizzare le scelte architetturali delle PA;
- individuare le scelte tecnologiche che favoriscano lo sviluppo, da parte delle PA, cittadini e imprese, di soluzioni applicative innovative che abilitino l'utilizzo dei servizi individuati nelle Infrastrutture immateriali del Piano triennale per l'informatica nella PA[1063];
- promuovere, quando applicabile, l'adozione dell'approccio *API first*, al fine di favorire la separazione dei livelli di back end e front end, con logiche aperte e standard pubblici che garantiscano ad altri attori, pubblici e privati, accessibilità e massima interoperabilità di dati e servizi;
- privilegiare standard tecnologici che soddisfino l'esigenza di rendere sicure le interazioni tra le PA e tra queste con i cittadini e le imprese;

[1] Il ModI è concettualmente la seconda versione (aggiornamento) del framework di interoperabilità della PA che nella prima versione fu definito nel 2005 con il nome di SPCoop - Servizio Pubblico di Cooperazione Applicativa, cf. http://www.agid.gov.it/agenda-digitale/infrastrutture-architetture/sistema-pubblico-connettivita/cooperazione-applicativa Il termine *modello* trova corrispettivo nel termine inglese framework, e pertanto nel presente documento i due termini verranno considerati sinonimi.

[105] https://ec.europa.eu/transparency/regdoc/rep/1/2017/IT/COM-2017-134-F1-IT-MAIN-PART-1.PDF

[2] Cf. https://ec.europa.eu/transparency/regdoc/rep/1/2017/IT/COM-2017-134-F1-IT-MAIN-PART-1.PDF

[106] https://pianotriennale-ict.italia.it/assets/pdf/Piano_Triennale_per_l_informatica_nella_Pubblica_Amministrazione.pdf

[3] Cf. https://pianotriennale-ict.italia.it/assets/pdf/Piano_Triennale_per_l_informatica_nella_Pubblica_Amministrazione.pdf

- semplificare le procedure di scambio di servizi tra le PA e, ove possibile, tra PA e privati, facilitando la realizzazione dei sistemi che le realizzano.

2.2 Il contesto europeo

Lo *European Interoperability Framework (EIF)* (in italiano Quadro Europeo di Interoperabilità - QEI[4]) fornisce orientamenti alle PA Europee su come operare le iniziative relative al tema dell'interoperabilità; tutto questo mediante una serie di raccomandazioni atte a stabilire relazioni tra le varie organizzazioni, razionalizzare i processi volti a sostenere i servizi digitali e assicurare che le norme esistenti e quelle nuove non pregiudichino gli sforzi di interoperabilità.

L'obiettivo dell'EIF è:

- orientare gli sforzi delle PA Europee nel progettare ed erogare servizi pubblici ad altre PA, cittadini e imprese che siano, per quanto possibile, *(i)* digitali per definizione, *(ii)* transfrontalieri per definizione e *(iii)* aperti per definizione;
- fornire alle PA orientamenti in merito alla progettazione e all'aggiornamento di quadri nazionali di interoperabilità o di politiche nazionali, strategie e orientamenti che promuovano l'interoperabilità;
- contribuire all'istituzione del Digital Single Market incoraggiando l'interoperabilità transfrontaliera e intersettoriale per l'erogazione di servizi pubblici europei.

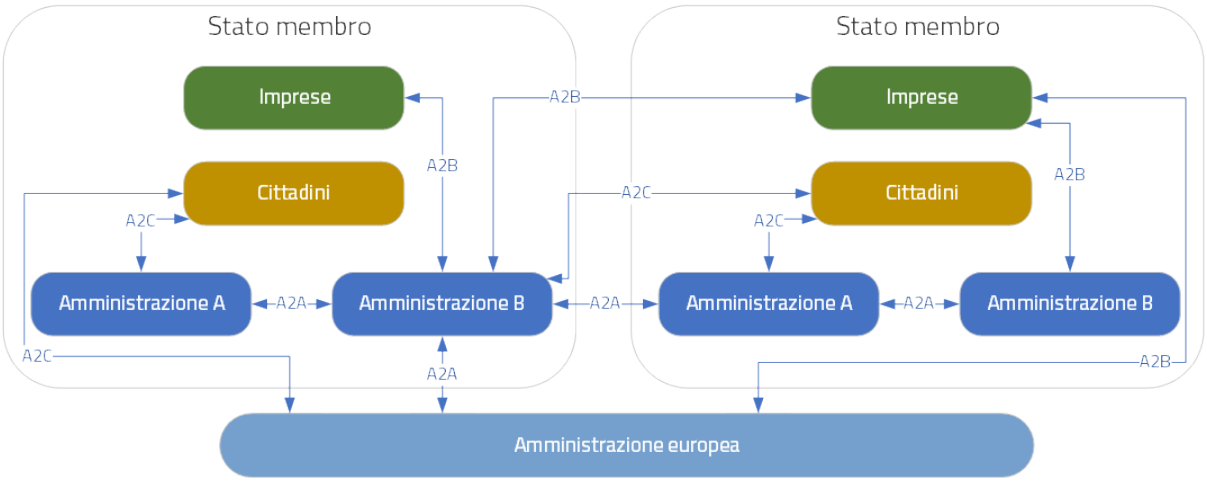


Figura 1 - Ambito di applicazione dell'EIF

L'ambito di applicazione dell'EIF comprende tre tipi di interazioni:

- A2A (*amministrazione-amministrazione*), ossia le interazioni tra PA;
- A2B (*amministrazione-impresa*), ossia le interazioni tra le PA e le imprese;
- A2C (*amministrazione-cittadino*), ossia le comunicazioni tra le PA e i cittadini.

Il modello di interoperabilità delineato nell'EIF è applicabile a tutti i servizi pubblici digitali, lo stesso include:

- quattro livelli di interoperabilità:
  - giuridico, per garantire che le organizzazioni che operano nell'ambito di diversi quadri giuridici (nazionali e settoriali), possano lavorare insieme;
  - organizzativo, per favorire l'allineamento delle procedure e processi delle organizzazioni coinvolte delineando le responsabilità e le aspettative per raggiungere obiettivi comuni concordati e reciprocamente vantaggiosi;

[4] In precedenti documenti a cura di AgID e del Team Digitale, il termine inglese framework è stato sovente tradotto in italiano come modello, ed è questo il termine utilizzato nel presente documento. La dicitura *quadro* è la traduzione letterale della Commissione Europea. Nel seguito di questo documento verrà preferito il termine modello, pur considerando i termini framework, modello e quadro come sinonimi.

  - semantico, per assicurare che il formato e il significato delle informazioni e dei dati scambiati siano mantenuti e compresi durante tutti gli scambi che avvengono tra le parti;
  - tecnico, in cui, attraverso l'adozione di specifiche di interfaccia, di servizi di interconnessione, di servizi di integrazione dei dati, la presentazione e lo scambio dei dati e i protocolli di comunicazione sicuri, si assicuri l'interoperabilità delle applicazioni e delle infrastrutture che collegano sistemi e servizi.

- una componente trasversale ai quattro livelli, denominata *governance dei servizi pubblici integrati*, per assicurare il necessario coordinamento e governance delle organizzazioni coinvolte nella erogazione di servizi pubblici in modo integrato;
- un livello di base, denominato *governance di interoperabilità*, per assicurare che le decisioni prese in merito ai quadri di interoperabilità, disposizioni istituzionali, strutture organizzative, ruoli e responsabilità, politiche, accordi e altri aspetti garantiscano e verifichino l'interoperabilità a livello nazionale e di UE.

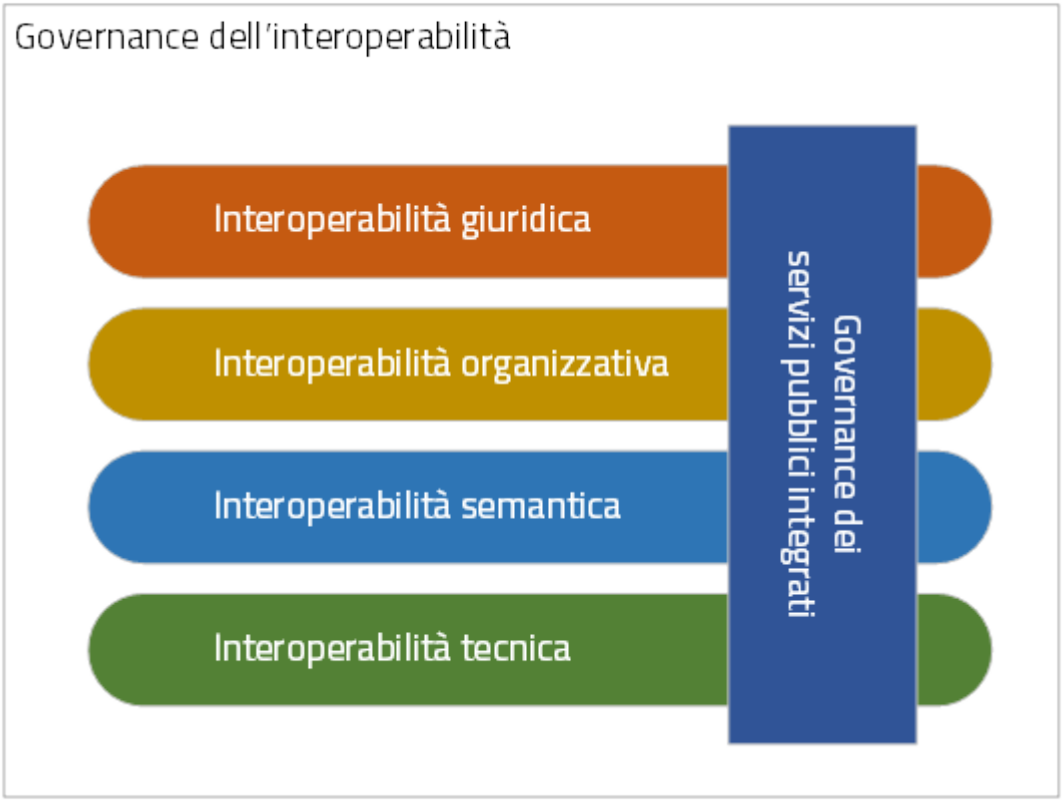


Figura 2 - Livelli di interoperabilità

Nel suo insieme il modello di interoperabilità delineato nell'EIF è stato disegnato sulla base dei 12 principi di interoperabilità, condivisi dagli Stati membri della Comunità Europea, individuati quali aspetti fondamentali per guidare le azioni tese a garantire l'interoperabilità:

1. *Sussidiarietà e proporzionalità*. Il principio di sussidiarietà prevede che le decisioni dell'UE vengano prese al livello più vicino a quello del cittadino mentre il principio di proporzionalità limita l'azione dell'UE a quanto necessario per raggiungere gli obiettivi dei trattati.
2. *Apertura*. Il principio di apertura fa riferimento principalmente ai dati, alle specifiche e al software. Nell'ottica di questo principio occorre: pubblicare i dati che si possiedono come dati aperti, fatta salva l'eventuale applicazione di determinate restrizioni; garantire condizioni di parità per il software open source e prenderne in considerazione l'utilizzo in modo attivo ed equo, tenendo conto del costo totale di proprietà della soluzione; prediligere le specifiche aperte, tenendo debitamente conto delle esigenze funzionali, del livello di maturità e del sostegno e dell'innovazione del mercato.
3. *Trasparenza*. In ottemperanza a questo principio occorre: conferire visibilità nel contesto amministrativo di una PA; assicurare la disponibilità di interfacce con i sistemi informatici interni e garantire il diritto alla tutela dei dati personali; garantire visibilità interna e fornire interfacce esterne per i servizi pubblici.

4. *Riusabilità*. Secondo tale principio si deve trarre vantaggio dal lavoro degli altri cercando le informazioni disponibili, valutandone l'utilità o la pertinenza rispetto al problema in questione e, se del caso, decidendo di usare soluzioni che si sono rivelate efficaci in altre situazioni.
5. *Neutralità tecnologica e portabilità dei dati*. Allorché istituiscono servizi pubblici, le PA devono concentrarsi sulle esigenze funzionali e posporre le decisioni in materia di tecnologia il più a lungo possibile per ridurre al minimo la dipendenza tecnologica, evitare di imporre tecnologie o prodotti specifici ai loro partner ed essere in grado di adattarsi all'ambiente tecnologico in rapida evoluzione.
6. *Centralità dell'utente*. Nel determinare quali servizi pubblici erogare e come farlo, si deve prendere in considerazione le esigenze degli utenti. Occorre perciò mettere a punto meccanismi per coinvolgere gli utenti nell'analisi, nella progettazione, nella valutazione e nell'ulteriore sviluppo dei servizi pubblici
7. *Inclusione e accessibilità*. Inclusione significa permettere a chiunque di approfittare delle opportunità offerte dalle nuove tecnologie per l'accesso e l'utilizzo dei servizi pubblici europei superando gli svantaggi e l'esclusione sociale ed economica. L'accessibilità garantisce che le persone anziane, i disabili e gli altri gruppi svantaggiati possano utilizzare i servizi pubblici alla stregua di tutti gli altri cittadini.
8. *Sicurezza e privacy*. Le interazioni con le autorità pubbliche devono svolgersi in un ambiente sicuro ed affidabile ed in totale conformità con le norme in materia di protezione dei dati, di identificazione elettronica e dei servizi fiduciari.
9. *Multilinguismo*. Occorre soddisfare le aspettative di cittadini e imprese che desiderano essere serviti nella loro lingua, o in un'altra lingua di preferenza, e la capacità delle PA di offrire servizi in tutte le lingue ufficiali.
10. *Semplificazione Amministrativa*. Le PA, laddove possibile, devono razionalizzare e semplificare le loro procedure amministrative migliorandole o eliminando quelle che non hanno utilità pubblica.
11. *Conservazione delle informazioni*. La legislazione impone che le decisioni e i dati siano conservati e che vi si possa accedere per un determinato periodo di tempo. Occorre pertanto formulare una politica di conservazione a lungo termine per le informazioni relative ai servizi pubblici.
12. *Valutazione dell'efficacia e dell'efficienza*. Esistono numerosi modi per misurare il valore offerto dall'interoperabilità dei servizi pubblici, quali le considerazioni circa il ritorno sull'investimento, il costo totale di proprietà, il livello di flessibilità e adattabilità, la riduzione degli oneri amministrativi, l'efficienza, la riduzione dei rischi, la trasparenza, la semplificazione, il miglioramento dei metodi di lavoro e il grado di soddisfazione degli utenti. Valutare l'efficacia e l'efficienza di diverse soluzioni di interoperabilità e opzioni tecnologiche, in considerazione delle esigenze dell'utente, della proporzionalità e dell'equilibrio tra costi e benefici.

L'EIF delinea uno schema concettuale per i servizi pubblici integrati al fine di orientarne la progettazione, lo sviluppo, la gestione e la manutenzione da parte degli Stati membri. Lo schema concettuale promuove l'idea di *interoperability-by-design* (*interoperabilità fin dalla fase di progettazione*). Lo schema promuove la riusabilità come motore per l'interoperabilità, riconoscendo che i servizi pubblici dovrebbero riutilizzare le informazioni e i servizi esistenti e provenienti da varie fonti, sia all'interno che all'esterno dei confini organizzativi delle PA. Le informazioni e i servizi dovrebbero essere recuperabili e resi disponibili in formati interoperabili.

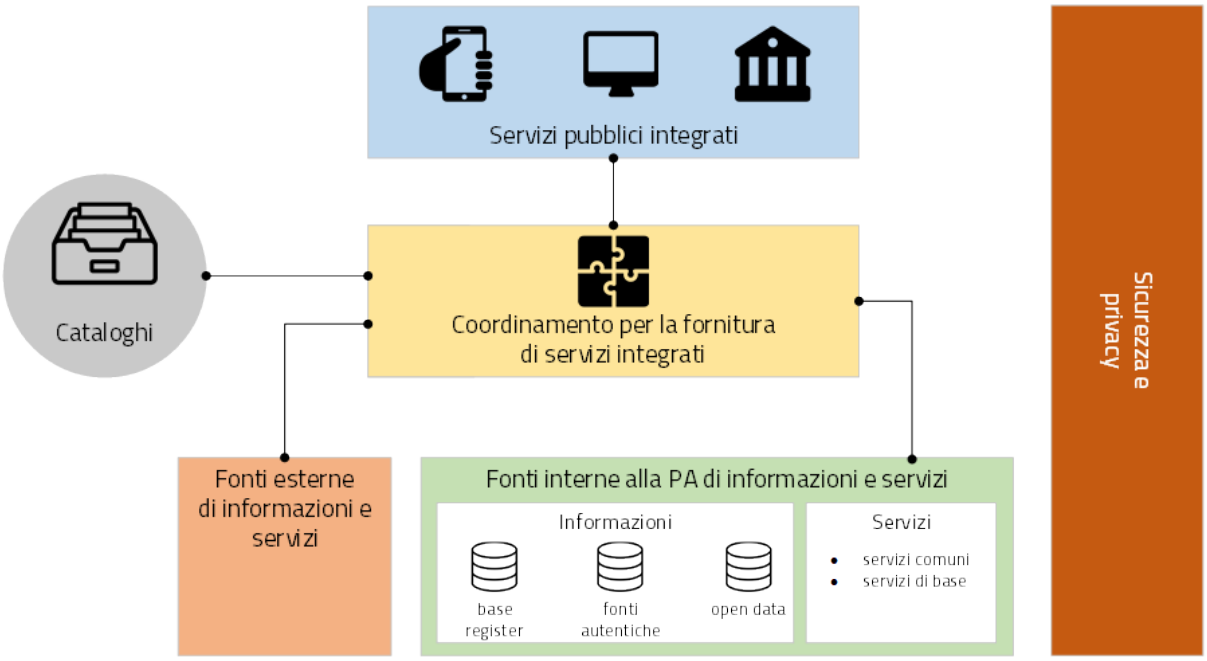


Figura 3 - Schema concettuale per i servizi pubblici integrati

La Commissione Europea ha individuato uno schema concettuale per i servizi pubblici che comprende:

- la *fornitura di servizi integrati* basata su una *funzione di coordinamento* per eliminare la complessità per l'utente finale;
- una politica di fornitura del servizio basata sul *concetto secondo cui tutte le porte sono buone* per offrire opzioni e canali alternativi per l'erogazione dei servizi, garantendo nel contempo la disponibilità di canali digitali (*digital first*);
- il riutilizzo di dati e servizi per ridurre i costi e accrescere la qualità dei servizi e l'interoperabilità;
- cataloghi che descrivono i servizi riutilizzabili e le altre risorse per aumentare la loro rintracciabilità e il loro utilizzo;
- la governance dei servizi pubblici integrati;
- la sicurezza e la tutela della privacy.

La funzione di coordinamento garantisce l'individuazione delle esigenze e il ricorso ai servizi coordinati per fornire complessivamente un servizio pubblico. Le fonti di informazioni (*base register*, portali sui dati aperti e altre fonti autorevoli di informazioni) e i servizi, disponibili non solo all'interno del sistema amministrativo ma anche in un contesto esterno, possono essere utilizzati per creare servizi pubblici integrati. Per favorire questi processi occorre sviluppare un'infrastruttura condivisa di servizi e fonti di informazioni riutilizzabili che possa essere adottata da tutte le amministrazioni pubbliche favorendo il riutilizzo, la pubblicazione e l'aggregazione dei servizi e delle fonti di informazioni.

La direttiva relativa al riutilizzo dell'informazione del settore pubblico prevede un quadro giuridico comune per il riutilizzo dei dati (*open data*); in essa l'accento è posto sulla messa a disposizione di dati *machine-readable* ad uso di terzi per promuovere la trasparenza, la concorrenza leale, l'innovazione e un'economia basata sui dati.

I *cataloghi* hanno la finalità di consentire la ricerca di servizi, dati, software e modelli di dati.

Le PA devono poter fruire dei servizi erogati da terzi al di fuori dei confini delle loro organizzazioni, quali i servizi di pagamento forniti dalle istituzioni finanziarie oppure i servizi di connettività erogati da fornitori di servizi di telecomunicazioni. Esse hanno bisogno anche di utilizzare le *fonti esterne di informazioni*, quali i dati aperti e i dati delle organizzazioni internazionali, delle camere di commercio, ecc.

Nell'EIF è raccomandato:

- rendere disponibili fonti autorevoli di informazioni a terzi, istituendo nel contempo meccanismi di accesso e controllo per garantire la sicurezza e la riservatezza in conformità con la normativa specifica in materia;

- sviluppare interfacce con i base register, pubblicare i mezzi tecnici e i documenti necessari affinché terze parti possano connettersi e riutilizzare le informazioni disponibili;
- abbinare ad ogni base register i metadati appropriati, compresi la descrizione del contenuto, la garanzia del servizio e le responsabilità, le tipologie di master data contenuti, le condizioni di accesso e le licenze, la terminologia, il glossario e le informazioni sugli eventuali master data utilizzati di altri base register;
- creare e monitorare piani di garanzia della qualità dei dati per i base register e i relativi master data;
- elaborare cataloghi di servizi pubblici, dati pubblici e soluzioni di interoperabilità e utilizzare modelli comuni per descriverli;
- adottare e riusare fonti di informazioni e servizi esterni, laddove utile e fattibile, nello sviluppo dei servizi pubblici.

La sicurezza e privacy sono aspetti che devono essere definiti in pieno accordo con l'"e-Government action plan 2016-2020 della Commissione EU[1075]. Per le PA è raccomandato:

- tenendo conto dei requisiti specifici di sicurezza e riservatezza, identificare per ogni servizio le contromisure in conformità con piani di gestione del rischio;
- utilizzare i servizi fiduciari, in base al regolamento in materia di identificazione elettronica e servizi fiduciari, come meccanismi per garantire lo scambio sicuro e protetto dei dati nei servizi pubblici (Regolamento (UE) 2014/910[1086]).

Per perseguire gli obiettivi dell'EIF, la Commissione Europea ha individuato i seguenti obblighi per gli stati membri.

- Le PA devono identificare, negoziare e approvare un approccio comune per i componenti di servizi integrati. Ciò è realizzato a diversi livelli amministrativi, in base all'assetto organizzativo di ogni paese, per garantire che piani nazionali e le strategie di interoperabilità siano allineati con l'EIF e, se necessario, adattati e ampliati per tenere conto del contesto e delle esigenze nazionali.
- L'accesso ai servizi e alle informazioni deve essere realizzato mediante specifiche interfacce e condizioni di accesso preventivamente definite (accordi di interoperabilità). Vanno favorite le politiche di riuso dei dati e dei servizi.
- Concordare uno schema comune per interconnettere i componenti dei servizi, nonché predisporre e mantenere l'infrastruttura necessaria per istituire e mantenere i servizi pubblici europei.
- Le PA devono documentare i propri processi lavorativi utilizzando tecniche di modellizzazione comunemente accettate per erogare un servizio pubblico.
- Percepire i dati e le informazioni come un bene pubblico che deve essere adeguatamente prodotto, raccolto, gestito, condiviso, protetto e preservato, elaborando una strategia di gestione delle informazioni al livello più alto possibile per evitare la frammentazione e la duplicazione.
- Promuovere l'istituzione di comunità di settore e intersettoriali che mirino a creare specifiche aperte sulle informazioni condividendo i propri risultati sulle piattaforme nazionali ed europee.
- Utilizzare specifiche aperte, per garantire l'interoperabilità tecnica quando si istituiscono servizi pubblici.

2.3 Il quadro di riferimento attuale

Il Piano triennale per l'informatica nella PA[1097] costituisce il quadro di riferimento entro cui si colloca il ModI all'interno del *Modello strategico di evoluzione del sistema informativo della PA*.

[107] https://ec.europa.eu/digital-single-market/en/news/communication-eu-egovernment-action-plan-2016-2020-accelerating-digital-transformation
[5] Cf. https://ec.europa.eu/digital-single-market/en/news/communication-eu-egovernment-action-plan-2016-2020-accelerating-digital-transformation
[108] http://eur-lex.europa.eu/legal-content/IT/TXT/HTML/?uri=CELEX%3A32014R0910&from=EN
[6] Cf. http://eur-lex.europa.eu/legal-content/IT/TXT/HTML/?uri=CELEX%3A32014R0910&from=EN
[109] https://pianotriennale-ict.italia.it/
[7] Cf. https://pianotriennale-ict.italia.it/

Figura 4 - Piano triennale per l'informatica nella PA

Il modello strategico, pensato per superare l'approccio a «silos», storicamente adottato dalla PA, mira a favorire la realizzazione di un sistema informativo unitario della PA ed è caratterizzato da:

1. Gli strumenti per la generazione e diffusione dei servizi digitali, indicati come **Accesso ai servizi**, che:
   (a) definiscono regole comuni per la progettazione di interfacce, servizi e contenuti, migliorando e rendendo coerente la navigazione e l'esperienza del cittadino e delle imprese;
   (b) facilitano il design, la realizzazione e la diffusione di servizi digitali;
   (c) definiscono linee guida e kit di sviluppo;
   (d) provvedono alla creazione di community di sviluppatori, di designer e di chiunque voglia scambiare informazioni, collaborare e partecipare.
2. Gli **Ecosistemi**, sono i settori o le aree omogenee in cui si svolge l'azione da parte delle PA. Ciascun ecosistema coinvolge enti e organismi pubblici, e soggetti privati che operano nella stessa area di interesse e che a vario titolo svolgono funzioni attive all'interno dell'ecosistema stesso. I soggetti interessati interagiscono per il raggiungimento di obiettivi comuni attraverso
   (a) la condivisione delle esigenze e delle modalità operative;
   (b) la condivisione delle differenti competenze;
   (c) la pianificazione e la realizzazione di progetti ICT.

3. Il **Modello di Interoperabilità,** definisce i meccanismi che facilitano e garantiscono la corretta interazione tra gli attori del sistema (cittadini, imprese e PA), favorendo la condivisione trasparente di dati, informazioni, piattaforme e servizi. Il Modello di Interoperabilità è costituito da linee guida, standard tecnologici e profili di interoperabilità che ciascuna PA dovrà seguire al fine di garantire l'interoperabilità dei propri sistemi con quelli di altri soggetti per l'implementazione complessiva del Sistema informativo della PA.
4. Le **Infrastrutture immateriali** e il **Data & Analytics Framework** (DAF) della PA, che incentivano la centralizzazione e la razionalizzazione dei sistemi per la gestione dei processi e dei dati, riducendo la frammentazione degli interventi. In particolare, le *Infrastrutture immateriali* facilitano, standardizzano e razionalizzano la creazione di servizi ICT e sono composte dalle Piattaforme abilitanti e dai Dati della PA:

   (a) nelle *piattaforme abilitanti* ricadono tutti quei servizi infrastrutturali (eg. servizio di identificazione, servizio di pagamenti, ANPR) che agevolano e riducono i costi per la realizzazione di nuovi servizi uniformando gli strumenti utilizzati dagli utenti finali durante la loro interazione con la PA;

   (b) relativamente ai *dati della PA* si distinguono: le basi di dati di interesse nazionale, gli open data, e i vocabolari controllati.

   Il *Data & Analytics Framework* è un ambiente centralizzato che acquisisce e rende più fruibili i dati pubblici di interesse e ha l'obiettivo *(i)* di rendere più semplice e meno onerosa l'interoperabilità dei dati pubblici tra PA e la distribuzione e standardizzazione dei dati aperti (open data) e *(ii)* di permettere lo studio dei fenomeni sottostanti ai dati pubblici.
5. Le **Infrastrutture fisiche,** che perseguono l'obiettivo di aumentare la sicurezza, ridurre il costo delle infrastrutture tecnologiche e migliorare la qualità dei servizi software della PA, attraverso la razionalizzazione dei data center, l'adozione sistematica del paradigma cloud e lo sviluppo della connettività, con particolare riferimento alla rete Internet nei luoghi pubblici e negli uffici della PA.
6. La **Sicurezza** che comprende:
   - le attività per la regolazione e regolamentazione della cyber-security nella PA per l'*assessment test,*
   - il CERT-PA quale strumento operativo per supportare l'adozione dei corretti livelli di sicurezza presso le PA.
7. La **Gestione del cambiamento** che è una componente definita per far fronte alle necessità di coordinamento, gestione e monitoraggio delle attività funzionali allo sviluppo del Piano.

2.4 Scenario pregresso dell'interoperabilità nella PA

Nell'ottobre 2005 il CNIPA (oggi Agenzia per l'Italia digitale - AgID) ha pubblicato un insieme di documenti che costituiscono il riferimento tecnico per l'interoperabilità fra le PA. Tali documenti delineano il quadro tecnico-implementativo del Sistema pubblico di cooperazione (SPCoop), framework di interoperabilità a livello applicativo[1108].

SPCoop ha costituito il modello concettuale ed architetturale della cooperazione applicativa tra differenti Amministrazioni e/o soggetti pubblici italiani. Tale sistema era organizzato in modo da:

- supportare una modalità di erogazione dei servizi articolata per procedimenti istituzionali;
- essere paritetico fra tutti i soggetti cooperanti;
- essere indipendente dagli assetti organizzativi dei soggetti cooperanti;
- lasciare a ciascun soggetto cooperante la responsabilità dei servizi erogati e dei dati forniti;
- garantire a ciascun soggetto autonomia nella gestione dei propri sistemi e nella definizione ed attuazione delle politiche di sicurezza del proprio sistema informativo;
- lasciare a ciascun soggetto la responsabilità delle autorizzazioni per l'accesso ai propri dati e/o servizi.

In sintesi, alla base di SPCoop vi erano i seguenti principi:

[110] http://www.agid.gov.it/agenda-digitale/infrastrutture-architetture/sistema-pubblico-connettivita/cooperazione-applicativa
[8] Cf. http://www.agid.gov.it/agenda-digitale/infrastrutture-architetture/sistema-pubblico-connettivita/cooperazione-applicativa

1. *cooperazione tra amministrazioni* attraverso la erogazione e fruizione di servizi offerti tramite un unico elemento logico denominato *Porta di Dominio;*
2. *ambito di responsabilità* delle singole Amministrazioni dei servizi erogati che costituiscono il *Dominio di servizi applicativi* della stessa Amministrazione;
3. *accordi di servizio* quale rappresentazione formale della cooperazione tra erogatore/i e fruitore/i costituiti sulla base di un fondamento normativo;
4. *tecnologie di cooperazione:* i servizi erano erogati come web service basati sugli standard che in quel momento erano consolidati ed in uso (SOAP, WSDL, UDDI).

Con l'obiettivo di assicurare agli utenti di avere una visione integrata dei servizi di ogni PA, le tematiche coperte da SPCoop sono state tutte quelle che interessano l'interoperabilità dei sistemi a diversi livelli, ovvero:

- interoperabilità applicativa;
- catalogazione dei servizi;
- semantica dei dati e dei servizi;
- identità digitale.

Lo scenario normativo di SPCoop è quello inquadrato dal DPCM 1 aprile 2008, recante regole tecniche e di sicurezza del Sistema pubblico di connettività (SPC), di cui SPCoop era un componente fondamentale, poi compiutamente delineato sul piano tecnico-implementativo da una suite di linee guida di seguito richiamate:

- Interoperabilità applicativa
  - Specifiche della busta di e-gov
  - Specifiche della porta di dominio
  - Linee guida busta di e-gov
  - Qualificazione della porta di dominio
  - Qualificazione porta di dominio con concorso delle regioni
- Catalogazione dei servizi
  - Specifiche dell'accordo di servizio
  - Specifiche del Registro SICA
  - Raccomandazioni stesura accordi di servizio
- Semantica dei dati e dei servizi
  - Nomenclatura e semantica
- Identità digitale
  - GFID - Gestione federata delle identità digitali

In particolare SPCoop prevedeva:

- Tutti i servizi applicativi di una PA erano offerti attraverso un unico elemento denominato *Porta di Dominio*, che svolgeva funzioni di proxy e dispatcher assicurando l'implementazione del protocollo applicativo denominato *Busta e-Gov*, un'estensione dello standard SOAP.
- I servizi infrastrutturali per la gestione di tutti gli aspetti legati agli *accordi di servizio*, nel loro insieme denominati *Servizi SICA*, prevedevano:
  - *Servizi di Registro*: la componente, realizzata a partire dallo standard UDDI, entro cui erano registrati gli Accordi di Servizio organizzati in modo distribuito prevedendo due livelli, ovvero Generale, che contiene la totalità degli *accordi di servizio*, e Secondario, contenente delle viste definite secondo differenti criteri;
  - *Catalogo degli Schemi/Ontologie*, che offre gli strumenti per ragionare sulla semantica dei servizi e delle informazioni da essi veicolati;

  - *Servizi di Sicurezza* assicuravano le funzionalità per la qualificazione degli elementi del sistema e garantire gli opportuni requisiti di autenticità, riservatezza, integrità, non ripudio e tracciabilità dei messaggi scambiati.

Il tempo trascorso dalla definizione del modello e il mutato quadro tecnico, organizzativo e normativo rendono necessario l'aggiornamento del modello, obiettivo appunto della presente iniziativa, come anticipato nel 2017 attraverso la Determinazione 219/2017 - Linee guida per transitare al nuovo modello di interoperabilità[1119].

L'esperienza maturata con SPCoop, di seguito sintetizzata, deve essere considerata nella definizione del ModI.

Cosa ha funzionato

- La definizione di un quadro comune per l'implementazione dei meccanismi di interoperabilità tra i sistemi delle Pubbliche Amministrazioni permette di orientare gli sforzi per la realizzazione di servizi pubblici sulla logica propria degli stessi.
- Il coordinamento, anche delegato ad organi intermedi quali elementi di aggregazione di un insieme omogeneo di Amministrazioni, permette di favorire l'applicazione del modello condiviso.
- Il sistema di gestione federata delle identità digitali, nonostante si ponesse come un elemento fortemente innovativo, è stato utilizzato a livello regionale e ha consentito di disegnare su tali basi tecniche il futuro SPID.

Cosa deve essere cambiato

- Le tecnologie e gli standard utilizzati dal modello SPCoop richiedono un consistente aggiornamento in considerazione delle innovazioni intervenute in tali ambiti.
- È necessario un modello di governance che permetta di gestire le specificità dei singoli domini applicativi determinati dalle caratteristiche delle amministrazioni e dei soggetti terzi coinvolti.

Cosa deve essere abbandonato

- L'adozione di un'unica modalità per attuare l'interoperabilità dei sistemi non permette di considerare la molteplicità e la specificità delle esigenze di scambio tra le Pubbliche Amministrazioni e di queste con i cittadini e le imprese.
- La necessità di componenti infrastrutturali disegnati per la sola Pubblica Amministrazione italiana (come Porta di Dominio e Registro SICA) determina che la spesa per il loro sviluppo ed evoluzione sia totalmente a carico della Pubblica Amministrazione.

2.5 Principi del nuovo modello di interoperabilità

2.5.1 Interazioni

L'ambito di applicazione del Modello di Interoperabilità comprende i tre tipi di interazioni previsti nell'EIF. Le interazioni prevedono che i soggetti coinvolti svolgano alternativamente la funzione di **erogatore** di servizio, nel caso del soggetto che mette a disposizione API o servizio utilizzati da altri, e la funzione di **fruitore**, nel caso invece del soggetto che utilizza le API o servizi messi a disposizione da altro soggetto.

[111] http://www.agid.gov.it/sites/default/files/upload_avvisi/linee_guida_passaggio_nuovo_modello_interoperabilita.pdf
[9] Cf. http://www.agid.gov.it/sites/default/files/upload_avvisi/linee_guida_passaggio_nuovo_modello_interoperabilita.pdf

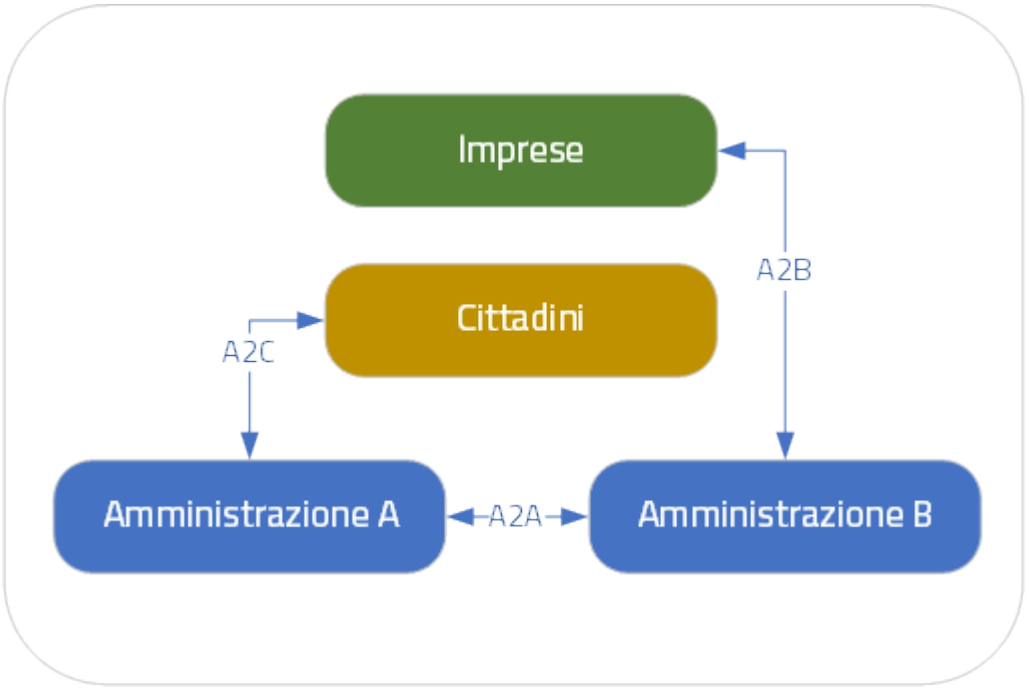


Figura 5 - Ambito di applicazione del modello di interoperabilità

I soggetti fruitori possono utilizzare le API/servizi[10] esposti dall'erogatore attraverso:

- una soluzione software attivata da un attore umano (*user agent/human*);
- un sistema applicativo *automatico*[11] (*server/machine*), anche allo scopo di definire nuovi servizi a valore aggiunto.

In considerazione di quanto sopra si individuano le seguenti possibili interazioni:

1. A2A in modalità *human-to-machine*;
2. A2A in modalità *machine-to-machine*;
3. A2B in modalità *human-to-machine*;
4. A2B in modalità *machine-to-machine*;
5. A2C in modalità *human-to-machine*.

2.5.2 Paradigmi di cooperazione

In generale, nell'integrazione dei sistemi software si individuano principalmente le seguenti tre casistiche che il modello di interoperabilità deve tener presente:

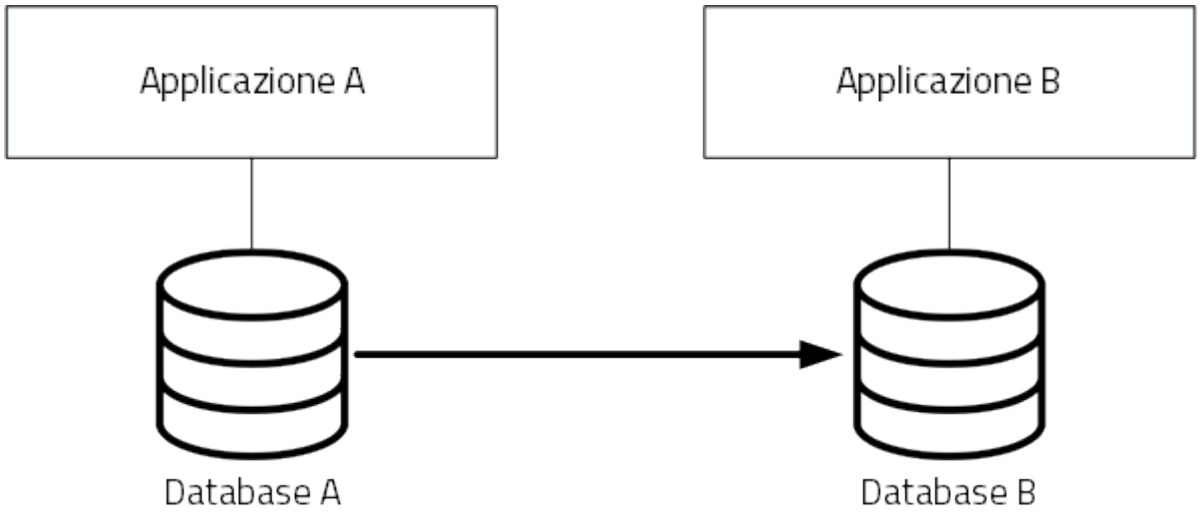
- **Condivisione di dati**: l'obiettivo è quello di tenere allineati i dati di uno o più sistemi; le applicazioni software che gestiscono (creano, aggiornano, leggono ed eventualmente cancellano[12]) tali dati, sono logicamente e fisicamente indipendenti. I processi che sovraintendono le applicazioni sono separati ed indipendenti. Il caso tipico è quello di un'Amministrazione, o soggetto privato, che per dare seguito ad una sua attività ha necessità di accesso ai dati posseduti dall'Amministrazione B, titolare degli stessi, senza che sia richiesto all'Amministrazione B nessuna elaborazione sui dati. Ad esempio, B è il Ministero delle Finanze che ha i dati del codice fiscale di ogni cittadino, ed A è un qualsiasi altro soggetto (pubblico o privato) che

[10] Con abuso di nomenclatura, ma intuitivamente chiaro, si intende nel presente documento servizio e API come sinonimo, ad indicare una componente software, esposta sul Web, che funge da servente e può essere utilizzata da client. In modo rigoroso, sia SPCoop che il ModI prevedono l'esposizione da parte di una PA di un'API accessibile sul Web come modalità base di interoperabilità e scambio di dati/informazioni, tale API permette la fruizione di un servizio offerto dalla PA stessa. La tecnologia web service è una particolare modalità con cui realizzare API che siano accessibili su Internet/intranet, da cui il termine Web. Tali concetti verranno ulteriormente approfonditi nel Modello di Interoperabilità.

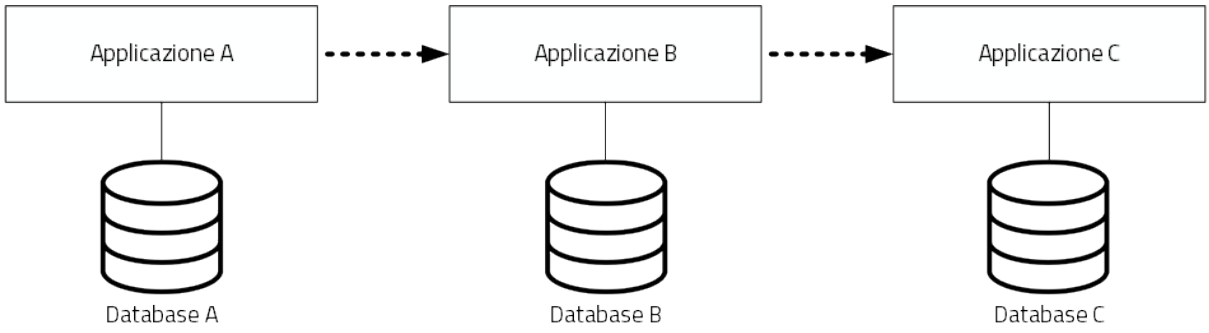
[11] Quindi non attivato da un utente umano, anche impropriamente detto *enterprise* in taluni contesti.

[12] Cf. le cosiddette operazioni CRUD - Create, Read, Update, Delete

all'interno della propria applicazione ha necessità di verificare la correttezza dei codici fiscali del proprio database, per poi utilizzarli in proprie elaborazioni.



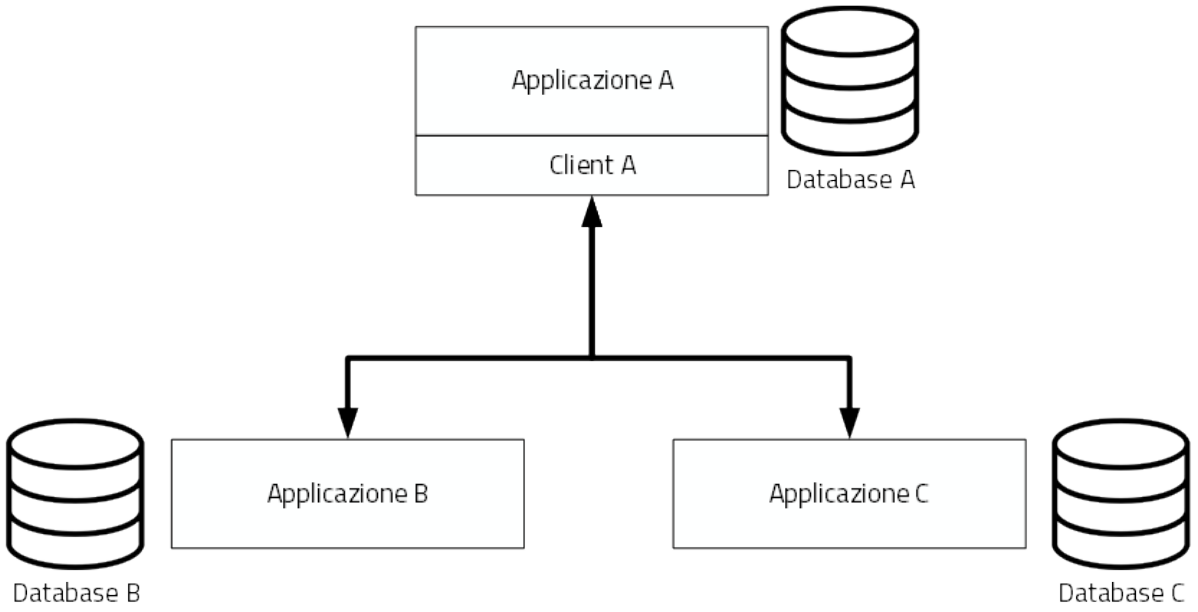
- **Notifica inter-PA**: in questo caso un'applicazione in un soggetto scatena un evento compie un'operazione che deve essere propagata sincronizzata con altre applicazioni di altri soggetti. Le applicazioni sono fisicamente indipendenti ma non logicamente, in quanto esiste un processo inter-organizzativo che sovraintende a tutte le organizzazioni che devono cooperare[13]. Il caso tipico è quello in cui il presentarsi di un evento all'interno di un'Amministrazione A debba essere comunicato ad altri soggetti B e C, pubblici e privati, che devono dare seguito a proprie procedure interne in relazione all'evento stesso, per vincoli normativi, ecc. Ad esempio, la registrazione di una nascita in un Comune è un evento che deve essere propagato all'Agenzia delle Entrate, per il rilascio di un nuovo codice fiscale, all'AUSL di riferimento per l'iscrizione al Servizio Sanitario Nazionale, ecc.



- **Composizione inter-PA**: in questo caso un insieme di applicazioni comunicano, anche in maniera bidirezionale, al fine di comporre una nuova logica applicativa ottenuta dalla loro interazione, ed erogare questa a sua volta come servizio a valore aggiunto. Talvolta questa nuova logica viene indicata come servizio/applicazione composito/a (o composto/a). Come nel caso precedente, esiste un processo inter-organizzativo che sovraintende a tutte le organizzazioni che vengono composte. Il caso tipico, nel mondo commerciale, è quello di un servizio che definite delle date ed una destinazione, propone all'utente voli aerei, hotel e noleggio auto, ecc, andando appunto a comporre servizi per la bigliettazione aerea, prenotazione alberghiera, noleggio auto, ecc. Nel caso della PA, un caso è una conferenza di servizi telematica[14] in cui diverse Amministrazioni compongono un nuovo servizio per dare seguito ad una istanza di un cittadino o di un'impresa.

[13] Nel caso della PA, questo processo inter-organizzativo corrisponde al concetto di macro-processo o di processo inter-amministrazione: M Mecella, C Batini (2001), Enabling italian e-government through a cooperative architecture. IEEE Computer 34 (2), pp. 40-45.

[14] La conferenza di servizi, cf. http://www.italiasemplice.gov.it/conferenza/guida-alle-novita-della-conferenza-di-servizi/ , è l'istituto che facilita l'acquisizione da parte della PA di autorizzazioni, atti, licenze, permessi e nulla-osta o di altri elementi comunque denominati, finalizzati all'emissione di un provvedimento amministrativo, coordinando differenti soggetti coinvolti. La conferenza semplificata in modalità sincrona è l'esempio di composizione di servizi, mentre la conferenza semplificata in modalità asincrona costituisce un altro caso della modalità precedente (notifica inter-PA).



È importante analizzare le analogie e differenze con il caso precedente: nel caso della notifica inter-PA, c'è una relazione peer-to-peer tra i vari soggetti coinvolti, e si parla di *coreografia* tra le applicazioni coinvolte[15]. Nel caso invece della composizione, una delle applicazioni ha un ruolo di *orchestrazione* nei confronti delle altre, e quindi c'è una relazione uno (l'orchestratore, che fa da *master*) a molti (le applicazioni orchestrate, che sono *slave*).

In entrambe le situazioni, esiste a livello concettuale (dovuto a norme, accordi, ecc.) un processo inter-organizzativo che sovraintende alle varie applicazioni, e l'espletamento del quale è l'obiettivo del servizio composto offerto.

La differenza tra i due casi risiede quindi nel grado di autonomia che i soggetti che concorrono al processo inter-organizzativo mantengono: se si sceglie un approccio completamente decentralizzato, si è nel caso notifica inter-PA, se si opta per un approccio per cui uno dei soggetti prende in carico la fornitura del servizio finale composto a valore aggiunto, allora si è nel caso composizione inter-PA.

2.5.3 Incrementalità del modello

In base alle considerazioni precedenti, il Modello di Interoperabilità si concretizza nella definizione, lo sviluppo, il miglioramento, la resa operativa, il mantenimento e la promozione di servizi, strumenti, norme tecniche e specifiche per l'interoperabilità delle soluzioni ICT basata su un'architettura modulare che include componenti interconnessi con l'ausilio di infrastrutture comuni. Questo modello, al fine di evitare le problematiche di possibile obsolescenza, e fronteggiare la necessità di continui aggiornamenti, si estrinsecherà concretamente in rilasci successivi e cadenzati nel tempo, di una serie di 6 documenti, in particolare:

1. **Presentazione del Modello di Interoperabilità**, che è il documento attuale, rilasciato nella prima versione a maggio 2018.
2. **Tecnologie ed approcci all'Integrazione ed Interoperabilità**, che nella prima versione (maggio 2018) viene rilasciato contestualmente al presente documento. Ha come oggetto l'individuazione delle possibili tecnologie ed approcci che possono essere utilizzati dalle PA.
3. **Profili e linee guida**, che fornirà indicazioni concrete, a livello tecnico, su differenti modalità operative per realizzare l'interoperabilità, tenendo conto delle possibili tecnologie ed approcci disponibili. Il Modello introduce il concetto di profilo di interoperabilità e come esso possa essere evoluto nel tempo.
4. **Sicurezza**, che fornirà indicazioni concrete per rispondere ad esigenza di sicurezza da applicare ai profili di interoperabilità definiti nel precedente documento.

[15] Approfondimenti sui concetti di orchestrazione e coreografia possono essere trovati in: https://stackoverflow.com/questions/4127241/orchestration-vs-choreography (C Peltz (2003), Web Services Orchestration and Choreography. IEEE, Computer 36(10), pp. 46-52 e R M Dijkman, M Dumas (2004), Service-Oriented Design: A Multi-Viewpoint Approach. Int. J. Cooperative Inf. Syst. 13(4), pp. 337-368)

5. **Governance del Modello di Interoperabilità**, che presenterà compiutamente la governance dell'intero modello e le sue modalità di evoluzione.
6. **Registri e Cataloghi**, che si occuperà di definire le linee guida per i registri e cataloghi necessari a supportare il modello stesso.

Gli interventi mirano, in coordinamento con le altre azioni presenti nel Piano Triennale per l'Informatica nella PA, a:

- definire e attuare specifiche comuni sui termini e le condizioni per gestire e accedere ai *base register*;
- attuare e promuovere modelli comuni per descrivere e classificare i servizi pubblici;
- individuare misure volte a creare sicurezza, tracciabilità e SLA - Service Level Agreement nell'erogazione dei servizi;
- analizzare i dati contenuti e i sistemi esistenti per l'informatizzazione delle PA;
- individuare gli ostacoli al reciproco riconoscimento, sviluppare mappature e sostenere gli sforzi di armonizzazione.

Gli **standard tecnologici** adottati, in particolare per i web service REST e SOAP, rispecchiano l'attuale stato di evoluzione delle tecnologie ed il loro utilizzo è consolidato nelle pratiche adottate nell'ambito dell'interoperabilità dei sistemi informativi.

2.5.4 Profili e pattern di interoperabilità

Il nuovo modello introduce i concetti di **caso d'uso**, **pattern** e **profilo di interoperabilità**.

Un caso d'uso di interoperabilità è la formalizzazione di una specifica esigenza di interoperabilità, che si manifesta frequentemente tra PA, o che può manifestarsi in particolari contesti applicativi. Tale necessità viene descritta mostrandone il contesto di applicazione, i problemi progettuali che ne derivano, i possibili schemi di soluzione e le implicazioni di ognuno di essi.

Ogni caso d'uso può essere risolto in vari modi, ognuno di questi schemi verrà indicato come pattern di interoperabilità. Esso fornisce una serie di linee guida per l'implementazione e l'interoperabilità che raccomandano come utilizzare una specifica tecnologia od approccio, e permette eventualmente di risolvere eventuali ambiguità/punti non adeguamente definiti in alcune tecnologie possibili con cui le PA possono interoperare.

Un profilo infine, in maniera trasversale rispetto ai casi d'uso ed ai pattern, risolve le diverse opzionalità o aspetti non adeguatamente specificati dagli standard tecnologici.

L'applicazione dei casi d'uso, pattern e profili agevola l'azione nello sviluppo e nella distribuzione di API/servizi. Il nuovo Modello proporrà un catalogo *di casi d'uso*, *profili* e *pattern di interoperabilità* messi a disposizione delle PA, popolato in maniera incrementale sulla base di esigenze individuate dall'Agenzia per l'Italia Digitale anche a fronte dell'evidenza di nuovi bisogni per le PA.

Ogni PA che offre un'API/servizio deve, nel nuovo modello, offrire un insieme di artefatti che lo accompagnano, in particolare:

- meccanismi di controllo delle versioni;
- documentazione coordinata alla versione;
- Software Development Kit - SDK - per l'interfacciamento e un ambiente di test (in analogia a quanto avviene per alcuni servizi commerciali di largo utilizzo in applicazioni Web[16]);
- dichiarazione sulla qualità del servizio che si impegna a rispettare. In questo secondo caso, deve anche definire le modalità di misurazione e deve offrire un'opportuna modalità di monitoraggio, che i fruitori possono sfruttare per la verifica.

Nello scambio informativo tra PA mediante API/servizi, le soluzioni che verranno adottate devono assicurare: *(i)* autenticità, *(ii)* integrità e *(iii)* non ripudio. In questo contesto il Regolamento (UE) 2014/910 fornisce una base

[16] Ad es., Paypal, cf. https://developer.paypal.com/ , offre SDK ed un servizio di prova, cosiddetta sandbox, che permette agli sviluppatori che si vogliono integrare con Paypal di provare le interazioni prima di rilasciare i propri sistemi.

normativa comune per le interazioni elettroniche sicure fra cittadini, imprese e PA; le soluzioni software conformi al Modello di Interoperabilità devono applicare i principi indicati in esso.

2.5.5 Catalogo delle API/servizi

Il Modello di Interoperabilità prevede la presenza del *Catalogo* quale componente che assicura alle parti coinvolte nel rapporto erogazione/fruizione la consapevolezza sulle interfacce e i livelli di servizio dichiarati.

La presenza del Catalogo è funzionale a:

- facilitare l'interoperabilità tra le PA e tra queste e i soggetti privati interessati;
- contenere la spesa della PA riducendo la replicazione di API/servizi;
- manifestare gli impegni dei fornitori o erogatori di API/servizi.

La realizzazione del Catalogo deve, fatti salvi i principi comuni che saranno emanati dall'Agenzia per l'Italia Digitale al fine di permettere una normalizzazione a livello nazionale, tener conto della:

- specificità dei territori e dei diversi ambiti entro cui la PA opera che potrà determinare la specializzazione del catalogo, prevedendo contenuti con un livello di aggregazione territoriale (eg. su base regionale) e/o relativamente agli ambiti tematici entro cui opera la PA (eg. giustizia). Tale scelta è ulteriormente giustificata dalla opportunità di favorire momenti di aggregazione di soggetti omogenei.
- esigenza di assicurare la governance del Catalogo, quale presupposto per garantire una semantica univoca e condivisa, per evitare ridondanze e/o sovrapposizioni in termini di competenze e contenuti.
- esigenza di assicurare una descrizione formale delle API/servizi che, attraverso l'utilizzo di *interface description language*, permetta di descrivere le interfacce degli stessi in maniera indipendente dal linguaggio di programmazione adottato dall'erogatore e dai fruitori degli stessi. L'attuale stato di evoluzione degli standard tecnologici indicati in precedenza determina la scelta di *WSDL* per i *web service SOAP* e *OpenAPI v3* per i *web service REST.*

2.5.6 Governance del modello

L'Agenzia per l'Italia Digitale è responsabile delle attività di *governance* del ModI con l'obiettivo di definire, condividere ed assicurare l'aggiornamento continuo dei seguenti aspetti:

- l'*insieme delle tecnologie* che abilitano l'interoperabilità tra le PA, e tra queste e cittadini ed imprese;
- i *casi d'uso di interoperabilità*;
- i *pattern di interoperabilità*;
- i *profili di interoperabilità*;
- il *catalogo* dei servizi resi disponibili dalle PA.

I progetti che realizzano gli Ecosistemi, previsti nel Piano Triennale per l'Informatica nella PA, si basano sul Modello di Interoperabilità, e possono determinare l'esigenza di nuovi *casi d'uso*, *pattern* e *profili di interoperabilità* che verranno definiti con un approccio collaborativo.

Nel precedente SPCoop, l'uso di servizi/API richiedeva un accordo tra amministrazioni anche tramite la firma di convenzioni bilaterali. Questo non sarà più necessario nel nuovo modello, in cui l'adesione si estrinsecherà nell'atto di registrazione da parte della PA di un'API/servizio nel catalogo. In ottemperanza al principio «once-only» definito nell"EU eGovernment Action Plan 2016-2020[11][12][17], l'erogatore si impegna a fornire l'accesso alle proprie API/servizi a qualunque soggetto registrato ne faccia richiesta[18]. Gli erogatori devono descrivere le loro API/servizi classificando le informazioni scambiate ove possibile collegandole ai vocabolari controllati e a concetti semantici predefiniti, utili anche a determinare l'impatto rispetto ai regolamenti in tema privacy e GDPR, e applicando tag di categoria. Il Catalogo può facilitare questo processo attraverso opportune euristiche.

[11][2] https://ec.europa.eu/digital-single-market/en/news/communication-eu-egovernment-action-plan-2016-2020-accelerating-digital-transformation)

[17] Cf. EU eGovernment Action Plan 2016-2020, https://ec.europa.eu/digital-single-market/en/news/communication-eu-egovernment-action-plan-2016-2020-accelerating-digital-transformation]

[18] Cf. Codice dell'Amministrazione Digitale Capo 1 Sez. 2 Art. 3 http://cad.readthedocs.io/it/v2017-12-13/_rst/capo1_sezione2_art3.html

In virtù degli articoli 12 e 14 del Codice dell'Amministrazione Digitale, AgID è formalmente incaricata della gestione di tutto il catalogo e di garantire il rispetto delle regole suddette e per farlo si avvale della collaborazione di alcuni enti, che vengono indicati come Capofila.

Gli enti Capofila si proporranno per eseguire questo compito su porzioni del catalogo; ci saranno enti che si occupano della gestione di aree geografiche e, allo stesso tempo, enti che si occupano della gestione di particolari aree tematiche.

In prima istanza si prevede che gli enti Capofila possano essere:

- a livello territoriale, le Regioni (e.g., la Regione per conto delle ASL regionali)
- a livello di ecosistema, gli enti individuati dai GdL descritti nel Piano Triennale al capitolo 6 Ecosistemi.

A tal fine, sul fronte delle aree tematiche il Piano Triennale 2017-2019 introduce:

- gli Ecosistemi[113][19], settori o aree di intervento in cui si svolge l'azione delle PA, che raggruppano i vari enti per aree tematiche;
- i Gruppi di Lavoro[114][20] che, all'interno degli Ecosistemi, indirizzano il vero e proprio lavoro di standardizzazione coinvolgendo sia tecnici che esperti dei rispettivi domini applicativi.

I Gruppi di Lavoro devono formalizzare le specifiche di dettaglio, attraverso il meccanismo dei profili e dei pattern di interoperabilità, e revisionare periodicamente le specifiche rilasciate.

Il nuovo Modello opera in assenza di elementi centralizzati che mediano l'interazione tra le entità comunicanti (erogatore e fruitore del servizio), pur prevedendo la presenza di un catalogo dei servizi disponibili allo scopo di permettere a tutti i soggetti interessati, pubblici e privati, di acquisire conoscenza dei servizi disponibili e delle loro modalità di erogazione/fruizione.

L'Agenzia per l'Italia Digitale ha il ruolo di:

- recepire le esigenze, anche applicative, delle PA, astrarre tali esigenze ed eventualmente formalizzare i casi d'uso ed i pattern di interoperabilità;
- coordinare il processo di definizione dei profili di interoperabilità;
- rendere disponibile il catalogo, attraverso un'interfaccia di accesso unica per permettere a tutti i soggetti interessati, pubblici e privati, di assumere consapevolezza dei servizi disponibili;
- verificare il rispetto delle regole del Modello di Interoperabilità, quale condizione di accesso al catalogo, e controllare con continuità il rispetto dei requisiti per l'iscrizione al catalogo.

[113] http://pianotriennale-ict.readthedocs.io/it/latest/doc/06_ecosistemi.html
[19] Cf. http://pianotriennale-ict.readthedocs.io/it/latest/doc/06_ecosistemi.html
[114] http://pianotriennale-ict.readthedocs.io/it/latest/doc/06_ecosistemi.html#linee-di-azione
[20] Cf. http://pianotriennale-ict.readthedocs.io/it/latest/doc/06_ecosistemi.html#linee-di-azione

CAPITOLO 3

Tecnologie ed Approcci all'Integrazione ed Interoperabilità

Il secondo documento del Modello di Interoperabilità, così come introdotto nella Visione generale, funge da guida alle possibili tecnologie che possono essere considerate dalle PA per l'integrazione e l'interoperabilità. Data la veloce evoluzione tecnologica, questo documento verrà continuamente aggiornato ed approfondito, e costituisce il riferimento per «Pattern e Profili di Interoperabilità». Questa versione si focalizza prevalentemente sugli approcci e tecnologie basati su SOAP, REST e Message Broker, ed accenna ad altre possibili scelte che in futuro potrebbero essere valutate.

3.1 Introduzione alle interfacce di servizio

3.1.1 Il concetto di servizio

I servizi sono sempre più rilevanti nella nostra vita e nei paesi di tutto il mondo. Il concetto di servizio copre un ampio spettro di aspetti nelle relazioni moderne tra amministrazioni pubbliche, fornitori privati e utenti finali.

Introduciamo il concetto di **servizio**[1] così come intuitivamente percepito nella vita quotidiana. Interagiamo ogni giorno con le persone e le imprese per soddisfare i nostri bisogni, facendo uso di transazioni, ad esempio di tipo economico in cui, dato un pagamento, possiamo acquisire un bene, o utilizziamo un bene che non è nostro, per raggiungere un obiettivo. Nel secondo caso stiamo parlando dell'uso di un servizio. Ad esempio, quando compriamo un biglietto ferroviario Milano-Roma, stiamo utilizzando un bene non nostro (il treno) per soddisfare la nostra necessità di mobilità. Arrivati a Roma, il nostro bisogno è soddisfatto e nulla ci rimane per l'uso, in termini di possesso del bene (il treno) utilizzato.

Un servizio consiste quindi in un'attività, o in una serie di attività, di natura più o meno intangibile, che si svolgono in uno scambio tra un fornitore e un cliente, in cui l'oggetto della transazione è un bene immateriale.

I servizi sono espletati in un sistema di servizi. Un ecosistema di servizi è l'insieme delle regole, delle componenti sociali, delle organizzazioni, dei processi, delle risorse umane, dei materiali e delle tecnologie che nella società coincidono con la produzione e l'uso dei servizi. Un sistema di servizi è caratterizzato da tre tipi di utenti finali: cittadini, imprese e l'ambiente circostante. Vari tipi di produttori forniscono servizi; e per questo verranno indicati più in generale come *erogatori* di servizi.

Per fornire servizi, gli erogatori devono eseguire una serie di attività, indicate come *processi di servizio*. Un processo di servizio è un insieme di attività la cui esecuzione, in base a un determinato flusso di controllo, produce in uscita un servizio fornito a un utente che ne ha bisogno. Se, ad esempio, vogliamo prenotare e beneficiare di un

[1] La trattazione si basa in parte su C. Batini, M. Castelli, M. Comerio, M. Cremaschi, L. Iaquinta, A. Torsello, G. Viscusi (2015): The Smart methodology for the life cycle of services. Cf. https://boa.unimib.it/retrieve/handle/10281/98632/144883/SmartBook-0315.pdf

viaggio in treno, la prenotazione e l'acquisizione di un biglietto sono la prima fase del processo corrispondente, che avviene presso un'agenzia di viaggi o, sempre più spesso, attraverso Internet. La seconda fase è il momento del viaggio, in cui sono coinvolte risorse quali il treno che ci trasporta, il materiale rotabile, il personale a bordo, il personale nelle stazioni. Nell'esecuzione del processo di servizio, ci sono delle interazioni tra il cliente e l'erogatore; tali interazioni, dal punto di vista del cliente, sono percepite come *operazioni* a sua disposizione, con cui è possibile richiedere e modificare aspetti specifici del servizio. Nell'esempio del servizio di trasporto ferroviario, le operazioni sono quelle che permettono al cliente di acquisire un biglietto, modificare la prenotazione (modifica del posto, del giorno, ecc.), accedere al treno, ecc. La *granularità* delle operazioni offerte al cliente dipende dall'erogatore dal processo di servizio che viene messo in atto per espletare il servizio.

La PA, in tutto il mondo, è fornitrice di una vasta gamma di servizi. Le differenze tra PA ed erogatori privati sono molteplici. Soprattutto, la fornitura di servizi è un obbligo legale per la PA.

Ad esempio, in Italia, sulla base di una legge emanata nel 1950, i Comuni sono responsabili dei registri della popolazione residente. Pertanto, se un cittadino ha bisogno di un certificato di residenza, deve andare al comune, che è responsabile per la validità e la correttezza delle informazioni che contiene il certificato. I fornitori privati forniscono servizi in base alla convenienza economica. Secondo il contesto, i prezzi dei servizi sono generalmente regolati nella PA da leggi, decreti o direttive che mantengono nella società una forma di equità sociale. A volte la PA fornisce servizi gratuitamente, mentre di fatto li finanzia attraverso le tasse. I fornitori privati forniscono servizi a pagamento e le entrate sono la ragione della loro attività come azienda. Infine, la PA nella pianificazione della produzione e della fornitura di servizi si ispira a criteri che tengono conto delle esigenze delle comunità, ed è quindi ispirata da una visione sociale, mentre le società private sono indirizzate dal mercato.

Il concetto di servizio include una grande quantità di aspetti. Di conseguenza, è necessario determinare il perimetro di osservazione del concetto di servizio nella PA, il dominio considerato nel Modello di Interoperabilità. Delineiamo nel seguito diverse classificazioni di servizi.

1. Classificazione in base alla natura del fornitore di servizi. In questo caso, abbiamo:

   (a) servizi amministrativi (o abilitanti), la cui fornitura non ha carattere discrezionale da parte della PA, in quanto derivano da procedimenti amministrativi definiti dalla legge;

   (b) servizi che chiamiamo orientati al mercato o facilitanti, che la PA può decidere o meno di fornire, in base alla presenza di un obbligo procedurale, e che sono più spesso erogati da fornitori privati del mercato dei servizi.

   I servizi amministrativi forniti dalla PA sono di primario interesse, ma è anche importante attirare l'attenzione su servizi orientati al mercato, che sono parte delle aspettative e dei bisogni degli utenti e potrebbero essere forniti da soggetti pubblici o privati.

2. Classificazione in base alla natura finale del servizio prodotto. In questo caso, abbiamo:

   (a) servizi che rispondono alle esigenze degli utenti che modificano il loro stato. Verranno indicati come servizi che modificano lo stato (dell'utente e/o del mondo) o semplicemente servizi;

   (b) servizi il cui scopo è quello di fornire informazioni e/o conoscenze che l'utente non possiede e che sono utili per un'attività operativa o un processo decisionale. Verranno indicati come servizi informativi o semplicemente informazioni.

   Un esempio della prima categoria è la fornitura di una licenza commerciale che consente a un'azienda di vendere la propria merce; questo servizio modifica lo stato dell'azienda perché consente una nuova attività commerciale. Un esempio di servizio informativo è l'informazione resa disponibile sugli orari di apertura di un laboratorio, che non cambia lo stato del soggetto che ha richiesto l'informazione, ma gli dà la possibilità di intraprendere un'azione o di prendere una decisione per andarci.

3. Classificazione in base al consumatore. In questo caso, possiamo distinguere tra:

   (a) servizi esterni, quando il servizio è focalizzato al di fuori della PA, verso la comunità di cittadini e imprese;

   (b) servizi interni, quando il servizio è dedicato agli utenti interni all'organizzazione erogatrice, sia essa PA che erogatore di servizi privato.

Oltre ai servizi, sappiamo che altri tipi di oggetti coinvolti nelle transazioni sono beni e informazioni; abbiamo visto che l'informazione può essere vista come un tipo specifico di servizio, quindi non c'è una chiara distinzione

tra servizi e informazioni, nel senso che entrambi i tipi di concetti appartengono a un concetto di servizio più generale. Allo stesso modo, tra prodotti e servizi non è possibile distinguere una linea precisa.

Consideriamo il caso in cui dobbiamo viaggiare in India, e il nostro obiettivo immediato è ottenere un visto per l'India; contattiamo due agenzie che, quando richiesto per le condizioni che applicano per fornire il visto, rispondono come mostrato nella tabella seguente:

Obiettivo del servizio	**Agenzia 1**	**Agenzia 2**
Necessità di un visto per andare in India	*nella nostra agenzia rilasciamo il visto in 7 giorni, al costo di € 30, e la penalità per un giorno di ritardo è di € 2*	*nella nostra agenzia facciamo il possibile per rilasciare il visto in 2 settimane, il costo è di € 20*

Guardando le due specifiche, il nostro obiettivo ora è fornire loro una struttura, distinguendo le diverse parti che hanno ruoli diversi.

Possiamo identificare i tipi di proprietà:

- proprietà funzionale, che esprime «cosa» otteniamo dal servizio;
- qualità del servizio, riferito a caratteristiche (ad es., tempo di consegna) che specificano vantaggi o utilità percepita, associati al servizio;
- proprietà non funzionali, esprimendo «come» il servizio ci viene consegnato.

La tabella seguente mostra la classificazione delle proprietà applicate all'esempio di cui sopra:

Tipo di proprietà	**Agenzia 1**	**Agenzia 2**
funzionale	rilascio del visto	rilascio del visto
qualità del servizio	in 7 giorni	il possibile in 2 settimane (*best effort*)
altra non funzionale	prezzo : € 30 penale : € 2 / giorno ritardo	prezzo : € 20

Le proprietà funzionali di un servizio descrivono cosa fa il servizio per il cliente. Una proprietà funzionale consente un cambiamento di stato del mondo reale, coerentemente con gli obiettivi espressi dal cliente. Le proprietà non funzionali di un servizio definiscono il modo in cui il servizio esegue le proprietà funzionali. Lo schema dei dati del servizio (talvolta chiamato *information model*) descrive i tipi di dati che rappresentano lo stato del mondo reale quando il servizio viene eseguito. I servizi possono essere visti come cambiamenti di stato del mondo reale ad un alto livello di astrazione, quindi un modo di descrivere i tipi di dati coinvolti in tali cambiamenti sono gli schemi concettuali, ad esempio diagrammi Entity Relationship o UML Class Diagram.

Quindi l'esempio mostra che i servizi possono essere descritti in termini delle seguenti caratteristiche:

1. un nome;
2. un insieme di proprietà funzionali, le operazioni appunto discusse in precedenza;
3. un insieme di proprietà non funzionali, tra cui quelle relative alla qualità del servizio;
4. uno schema di dati di servizio.

Finora abbiamo introdotto un modello che ci consente di descrivere un singolo servizio. Nei nostri eventi della vita quotidiana, per raggiungere i nostri obiettivi, abbiamo bisogno di invocare un numero elevato di servizi, facendo riferimento a un numero elevato di proprietà funzionali (operazioni). Consideriamo cosa accade in corrispondenza a un cambio di indirizzo di abitazione. Quando cambiamo il nostro indirizzo di casa, dobbiamo scegliere un nuovo medico, un nuovo fornitore di elettricità e acqua, dobbiamo cambiare il nostro indirizzo nella patente di guida, ecc. Inoltre, la procedura amministrativa è diversa nel caso in cui ci si trasferisce da un comune ad un altro comune, o se cambiamo il nostro indirizzo a causa della partenza dal nostro paese per andare a vivere all'estero.

I servizi interessati sono ovviamente concettualmente correlati. Ci concentriamo su due relazioni concettuali fondamentali, *part-of* e *is-a*. Una relazione *part-of* vale tra due servizi quando la specifica di uno ha come componente la specifica dell'altro. Nell'esempio, i servizi che (offrono le operazioni che) aggiornano l'indirizzo di casa nella patente di guida, scelgono il nuovo medico e scelgono il nuovo fornitore di energia elettrica, sono

tutti legati al servizio «cambio di indirizzo di casa». Diciamo che «cambio di indirizzo di casa» è un servizio composito, e i quattro servizi *part-of* con esso sono servizi elementari. Un servizio è elementare quando non siamo interessati a rappresentarlo ulteriormente in termini di componenti più atomici.

Fondamentalmente, un *servizio* è *elementare* se e solo se non esiste un altro servizio con una relazione *part-of* con esso, altrimenti è un *servizio composito*.

Il costrutto *part-of*, pur essendo efficace nel relazionare servizi elementari e compositi, non ci aiuta ad esprimere la relazione esistente tra i diversi tipi di servizi relativi al «cambio di indirizzo di casa» nei diversi contesti in cui si applicano. Abbiamo bisogno per questo scopo di un nuovo costrutto. Una relazione *is-a* vale tra un servizio s_i (servizio figlio/specifico) e un servizio s_j (servizio padre/generale) quando s_i è una specializzazione (caso specifico) di s_j. Secondo la proprietà di ereditarietà dell'*is-a*, s_i eredita tutte le proprietà (funzionali e non funzionali) di s_j. Inoltre, s_i eredita tutte le relazioni tra s_j e le sue componenti. s_i può avere proprietà aggiuntive, non in s_j. Ad esempio, tre servizi che cambiano indirizzo tra due comuni, cambiano indirizzo tra Italia e estero, e cambiano indirizzo tra due paesi stranieri, possono essere considerati casi specifici del servizio generico di «cambio di residenza». Le caratteristiche comuni a tutti e quattro i servizi sono la necessità di aggiornare due basi di dati, mentre i database specifici cambieranno in base ai luoghi coinvolti nel cambio di indirizzo. Inoltre, quando ci si sposta dall'Italia all'estero, possiamo immaginare che verranno attivate ulteriori procedure amministrative specifiche, ad es., per questioni relative alla cittadinanza.

Concludiamo questa breve introduzione sui servizi, rimarcando che i servizi sono erogati attuando dei processi. Un processo pubblico è un processo che definisce le interazioni tra i partecipanti (nel processo) e le attività che sono visibili al pubblico per ogni partecipante. Un processo privato è un processo che, oltre alle interazioni e alle attività definite nei processi pubblici, definisce le interazioni e le attività interne ai singoli partecipanti.

3.1.2 Servizio digitale, API e Interfaccia di servizio

Un **servizio digitale** (talvolta anche indicato come *electronic service* o *e-service*) è un servizio che *viene erogato via Internet o in una rete, la fornitura è essenzialmente automatizzata o comporta solo un intervento umano minimo, ed è impossibile da garantire in assenza di tecnologia informatica*[2]. Quanto detto per i servizi, vale anche per quelli digitali, essendo questi una specializzazione.

La trasposizione di un *servizio* in un *servizio digitale* non si riduce al solo utilizzo di tecnologie informatiche ma, per ottenere la totalità dei vantaggi conseguenti da tale possibilità, richiede la necessità di ridefinire i processi attraverso una riprogettazione degli stessi (*Business Process Reengineering*, in breve BPR). Il BPR deve, tra le altre, assicurare:

- la formazione degli atti amministrativi direttamente in digitale, per ridurre gli oneri legati alla gestione degli originali analogici;
- superare una visione document-oriented favorendo una visione record-oriented, al fine di agevolare la circolarità delle informazioni in possesso della PA;
- efficientare le azioni realizzate da parte della PA, per razionalizzare le proprie funzioni e compiti;
- mettere al centro dell'azione amministrativa i cittadini ed imprese, per l'attuazione della semplificazione amministrativa.

Nella progettazione di sistemi software, tipicamente si distinguono tre strati logici di funzionalità in comunicazione tra loro:

- logica di presentazione (presentation layer) o front-end (ad es., un'applicazione web, una app mobile, ecc.), ha il compito di presentare i risultati dell'elaborazione all'utente umano ed inviare le richieste di questi verso la parte centrale/elaborativa del sistema, facendo dunque da interfaccia uomo-macchina;

[2] Cf. Wikipedia, https://en.wikipedia.org/wiki/E-services] Rowley (Rowley J. (2006): An analysis of the e-service literature: towards a research agenda. Internet Research, 16 (3), 339-359) defines e-services as » [...] deeds, efforts or performances whose delivery is mediated by information technology. Such e-service includes the service element of e-tailing, customer support, and service delivery». This definition reflect three main components - service provider, service receiver and the channels of service delivery (i.e., technology). For example, as concerned to public e-service, public agencies are the service provider and citizens as well as businesses are the service receiver. The channel of service delivery is the third requirement of e-service. Internet is the main channel of e-service delivery while other classic channels (e.g. telephone, call center, public kiosk, mobile phone, television) are also considered. [...] The provision of services via the Internet (the prefix "e" standing for "electronic", as it does in many other usages), thus e-service may also include e-commerce, although it may also include non-commercial services (online), which is usually provided by the government».

- logica applicativa (application layer o business layer);
- logica di accesso ai dati (access data layer) o back-end, interroga il database o il sistema legacy[3].

Tale architettura viene poi spesso mappata a livello fisico-infrastrutturale in altrettanti strati fisici (*tier*) corrispondenti all'unità di computazione su cui risiede lo strato logico. Tali strati sono intesi interagire fra loro secondo le linee generali del paradigma client/server (il presentation layer è cliente della logica applicativa, e questa è cliente del modulo di gestione dei dati) e utilizzando interfacce ben definite. In questo modo, ciascuno dei tre strati può essere modificato o sostituito indipendentemente dagli altri, conferendo scalabilità e manutenibilità al sistema. Nella maggior parte dei casi, si intende anche che i diversi strati fisici (*tier*) siano distribuiti su diversi nodi di una rete anche eterogenea. Questa architettura di base può anche essere estesa ipotizzando che gli strati siano a loro volta «stratificati»; in questo caso si giungerebbe a una architettura multi-layer/tier.

Nello specifico dei servizi digitali, che appunto vengono erogati su Internet, il presentation layer verso l'utente può essere rappresentato da un Web server e da eventuali contenuti dinamici e statici (es. pagine di scripting che producono HTML visualizzato nel browser dell'utente), oppure da applicazioni mobili (*App*) che risiedono sul device mobile dell'utente (cellulare, tablet); la logica applicativa corrisponde a una serie di moduli integrati in un server applicativo, ed i dati sono depositati in maniera persistente su un DBMS o su un sistema legacy.

Con **application programming interface** (in acronimo **API**) si indica ogni *insieme di procedure/funzionalità/operazioni disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per l'espletamento di un determinato compito*. Spesso con tale termine si intendono le librerie software disponibili in un certo linguaggio di programmazione. Una buona API fornisce una «scatola nera», cioè un livello di astrazione che evita al programmatore di sapere come funziona l'implementazione dell'API ad un livello più basso. Questo permette di ri-progettare o migliorare le funzioni all'interno dell'API senza cambiare il codice che si affida ad essa. Una API che non richiede il pagamento di diritti per il suo accesso ed utilizzo è detta «aperta» (open). La finalità di un'API è di ottenere un'astrazione a più alto livello, di solito tra lo strato sottostante l'API e quello che la utilizza (client).

Per realizzare un servizio digitale, come detto, è necessario progettare e realizzare i tre strati; lo strato di logica applicativa offre la sua API affinché chi sviluppa lo strato di presentazione all'utente possa utilizzarla come se la logica applicativa fosse una libreria; estendendo, se vari sistemi esportano le proprie logiche applicative come API, la logica di presentazione può utilizzarle insieme, mischiandole (*mash-up*), esattamente come nello sviluppo di software moderno si programma riutilizzando le librerie offerte nel linguaggio di programmazione, sistema operativo, ecc. Quando il servizio digitale è erogato su Internet (e prevalentemente sul Web che si basa sul protocollo HTTP) si parla di Web API. Per le Web API l'erogatore potrebbe decidere di rendere disponibile l'API non soltanto a chi sviluppa la logica di presentazione, ma «aperta» anche ad altre organizzazioni che volessero collaborare con l'erogatore, in questo caso si parla di Open API. In molti contesti, con abuso di nomenclatura, ma intuitivamente chiaro, i due termini vengono confusi e considerati sinonimi (dato che l'apertura è spesso associata al Web/Internet).

Per il W3C un **web service** è qualsiasi software che si rende disponibile su Internet e standardizza la sua interfaccia tramite la codifica XML[1174]. Un client richiama un'operazione offerta da un web service inviando una richiesta (solitamente sotto forma di un messaggio XML) e il web service invia una risposta XML. I web service invocano la comunicazione su una rete, con HTTP come protocollo più comune. I web service si basano principalmente su standard come XML-RPC e SOAP (Simple Object Access Protocol). Quindi un web service è un possibile modo di realizzare una Web API. Il termine web service (originatosi intorno ai primi anni 2000) è nato proprio per indicare la logica applicativa, esposta sul web, sottostante ad un servizio digitale. A partire dalla seconda metà degli anni 2000, creando possibili confusioni, il termine Web API è stato utilizzato come alternativa a web service per indicare altri approcci/protocolli/tecnologie (come REST) per realizzare API senza utilizzare XML-RPC e SOAP. Ma anche una Web API indica la logica applicativa, esposta sul web, sottostante ad un servizio digitale.

Al fine di evitare ogni possibile ambiguità, spesso dovuta semplicemente all'utilizzo di termini differenti per indicare gli stessi concetti, nel seguito del documento si utilizza il termine **interfaccia di servizio** per indicare **l'esposizione delle funzionalità applicative che sono necessarie per realizzare un servizio digitale**. Tutte le classificazioni e considerazioni presentate per i servizi, valgono per i servizi digitali e quindi per le interfacce di servizio. In particolare come queste classificazioni e considerazioni si calano in specifiche tecnologie/protocolli/standard è

[3] Un sistema legacy (letteralmente «ereditato», che è un lascito del passato) è un sistema informatico, un'applicazione o un componente obsoleto, che continua ad essere usato poiché l'utente (di solito un'organizzazione) non intende o non può rimpiazzarlo. Legacy equivale a versione «retrodatata» (rispetto ai sistemi/tecnologie correnti).

[117] https://www.w3.org/TR/ws-arch/#whatis

[4] Cf. https://www.w3.org/TR/ws-arch/#whatis

uno degli obiettivi del presente documento. Un'interfaccia di servizio si compone in generale di varie operazioni, e può essere realizzata come un web service, un'API, una Web API, ecc.

Ogni qualvolta c'è un servizio, si può immaginare che nella moderna spinta all'innovazione, si giunga prima o poi ad una controparte digitale.
Un servizio digitale, se sviluppato seguendo i più moderni approcci di ingegneria del software, deve essere organizzato separando la logica di presentazione da quella applicativa, dove quest'ultima deve esporre le proprie operazioni tramite una interfaccia di servizio. Una interfaccia di servizio è l'esposizione delle funzionalità applicative che sono necessarie per realizzare un servizio digitale; tale esposizione deve essere operata con un approccio/tecnologia/standard che ne permetta l'invocazione da un modulo software client.
Emerge in ultima analisi che ogni qualvolta c'è un servizio digitale, ci può essere una interfaccia di servizio equivalente, e viceversa ogni qualvolta c'è una interfaccia di servizio, è immediato ipotizzare il servizio digitale equivalente.
Una interfaccia di servizio può offrire più operazioni (almeno una). Una interfaccia di servizio può essere realizzata utilizzando approcci/tecnologie/standard web service, API, Web API, REST API, ecc.

Nel prosieguo di questo documento, ci si focalizza solamente sulle interfacce di servizio, che sono il fondamento del Modello di Interoperabilità.

3.1.3 Caratteristiche delle interfacce di servizio

Interfacce semplici e complesse In prima istanza, le interfacce di servizio possono essere distinte in due categorie: semplici e complesse.

Una interfaccia di servizio semplice implementa operazioni atomiche come ad esempio:

- Fornire contenuti puri, ad esempio informazioni dettagliate riguardo una risorsa (come le informazioni fiscali riguardanti una azienda) oppure le notizie del giorno;
- Effettuare una aggregazione semplice di informazioni provenienti da diversi sistemi back-end;
- Effettuare operazioni con effetti circoscritti ad un unico sistema di back-end in maniera atomica (che non richieda supporto alle transazioni).

Le interfacce di servizio semplici eseguono unità di lavoro atomiche che lasciano i sistemi sottostanti in uno stato consistente. Le operazioni non necessitano del mantenimento di uno stato tra una chiamata e l'altra e perciò sono anche note come interfacce di servizio stateless (senza stato). Si noti come il concetto di stato sia espresso in relazione all'interazione tra i due sistemi (client ed erogatore) e non alla persistenza di informazioni circa le risorse di interesse.

Le interfacce di servizio complesse coinvolgono l'utilizzo e la composizione di altre interfacce di servizio (in alcuni casi esposte da organizzazioni diverse) richiedendo il supporto all'esecuzione di processi e funzionalità di tipo transazionale. Questo significa che, rispetto alle interfacce di servizio semplici, in quelle complesse le operazioni hanno una granularità alta (meno fine) e richiedono il mantenimento di uno stato condiviso; per questo motivo vengono anche definite interfacce di servizio stateful (con stato). Concetti potenzialmente connessi a quello di stato sono il mantenimento di una sessione o conversazione.

Interfacce sincrone ed asincrone Un altro modo di classificare le interfacce di servizio è lo stile di interazione richiesto dalle diverse operazioni disponibili: sincrono (eg. di tipo Remote Procedure Call - RPC, chiamata remota a procedura) o asincrono (eg. basato sullo scambio di messaggi o documenti). Nelle operazioni sincrone, un client esprime la sua richiesta nella forma di una chiamata ed attende una risposta prima di continuare l'esecuzione. Nelle operazioni asincrone, invece, il client invia un documento/messaggio ma non si aspetta nessuna risposta (se non in alcuni casi il fatto che la richiesta è stata presa in carico). La risposta da parte dell'interfaccia di servizio, nei casi in cui ci sia, può apparire ore o anche giorni più tardi.

Interfacce semplici e mission-critical Un modo ulteriore di classificare le interfacce di servizio è quello di distinguere quelle sostituibili da quelle mission-critical. Una interfaccia di servizio sostituibile può essere fornita da diverse organizzazioni e la produttività è impattata in maniera limitata nel caso di disservizi. Una interfaccia di servizio mission-critical è invece di solito fornita da un'unica organizzazione e la indisponibilità della stessa può provocare dei forti disservizi.

Caratteristiche funzionali e non funzionali delle interfacce Le classificazioni introdotte non sono strette poiché a seconda delle operazioni fornite, una interfaccia di servizio può essere catalogata in una posizione qualsiasi tra i due estremi delle stesse.

Le interfacce di servizio devono essere accompagnate da una descrizione delle operazioni offerte il cui linguaggio dipende dalla tecnologia con cui l'interfaccia è implementata (si veda a partire dalla Sezione 3 per maggiori dettagli). La descrizione di una interfaccia di servizio di solito include caratteristiche funzionali e non funzionali. La descrizione funzionale si concentra sulle caratteristiche operative dell'interfaccia di servizio che descrivono il funzionamento in termini di operazioni offerte, i parametri richiesti da ognuna, gli *endpoint*[5] da utilizzare, il formato dei messaggi ed i protocolli di rete da utilizzare. La descrizione non funzionale si concentra invece sulla *qualità del servizio* (o qualità dell'interfaccia di servizio) in termini di limiti di utilizzo, costi e metriche di performance quali scalabilità, disponibilità, tempo di risposta, accuratezza, transazionalità, sicurezza e affidabilità.

3.1.4 Qualità del servizio

Il concetto di *quality of service - QoS*, fa riferimento alla descrizione non funzionale di una interfaccia servizio, cioè la capacità di una interfaccia di servizio di soddisfare le aspettative dei fruitori. Assicurare la QoS nell'ambito Internet e quindi ai fini dell'interoperabilità è una sfida critica a causa della natura dinamica ed impredicibile del contesto applicativo. Cambiamenti negli schemi di traffico, la presenza di transazioni business-critical, gli effetti dei problemi di rete, le performance dei protocolli e degli standard di rete richiedono una definizione precisa della QoS offerta da una interfaccia di servizio.

Gli elementi chiave a supporto della QoS possono essere riassunti come segue:

- *Disponibilità*. La probabilità che una interfaccia di servizio sia disponibile e funzionante in un istante casuale. Associato al concetto di disponibilità è quello di Time-To-Repair (TTR), cioè il tempo necessario a ripristinare una interfaccia di servizio una volta che questa diventa indisponibile. La disponibilità di una interfaccia di servizio dovrebbe potere essere verificata tramite l'esposizione di un'altra interfaccia di servizio di monitoraggio, dedicata ed a basso impatto (e quindi ad elevata disponibilità).
- *Accessibilità*. Misura la capacità di una interfaccia di servizio di essere contattabile da un elevato numero di richieste.
- *Prestazioni*. Le prestazioni vengono misurate solitamente rispetto a due valori: il *throughput* e la *latenza*. Il throughput rappresenta il numero di richieste soddisfatte in un dato intervallo. La latenza rappresenta la quantità di tempo che passa tra l'invio di una richiesta e la ricezione di una risposta. Una interfaccia di servizio con buone prestazioni ha un elevato throughput ed una bassa latenza.
- *Affidabilità*. Rappresenta la capacità di una interfaccia di servizio di funzionare correttamente e consistentemente fornendo la stessa QoS a dispetto di malfunzionamenti di diversa natura. Di solito viene espressa in termini di fallimenti in un dato lasso di tempo.
- *Scalabilità*. L'abilità di servire in maniera consistente le richieste a dispetto di variazioni nel numero delle richieste[6]. È strettamente connesso al concetto di accessibilità, ma qui il concetto fondamentale è il mantenimento delle prestazioni.
- *Sicurezza*. La sicurezza implica aspetti quali confidenzialità, integrità, autorizzazione ed autenticazione che saranno oggetto della Sezione 2.
- *Transazionalità*. Ci sono alcuni casi (ad es., interfacce di servizio stateful) in cui è necessario assicurare l'esecuzione transazionale di una operazione. La capacità di una operazione di rispettare questa proprietà è parte della QoS.

Gli erogatori devono prendere tutte le iniziative necessarie a mantenere i requisiti di QoS richiesti dal caso d'uso. Questo include anche l'utilizzo di buone pratiche. Ad esempio, per assicurare prestazioni e scalabilità il risparmio

[5] Con il termine endpoint si indica l'identificativo unico da utilizzare per richiamare un'interfaccia di servizio. Ad esempio, nel caso della tecnologia SOAP è l'URL del web service, nel caso di REST le URL (che hanno tutte un suffisso comune) delle risorse offerte, nel caso dei Message Broker il nome univoco della coda di messaggi o un topic nella stessa.

[6] In ambito cloud, si utilizzano i termini di scale-up/scale-down per indicare la scalabilità ottenuta incrementando o riducendo le risorse di singoli sistemi (ad es., memoria RAM), di scale-out/scale-in per indicare la scalabilità ottenuta mediante distribuzione, aggiungendo o diminuendo il numero dei sistemi utilizzati.

della banda è una condizione fondamentale. Le interfacce di servizio dovrebbero quindi implementare meccanismi di compressione del payload[7] e supportare la paginazione[8].

Quando si utilizzano meccanismi di caching, essi devono essere documentati nelle specifiche delle interfacce di servizio, ed essere conformi alle specifiche RFC-7234[1189].

Questa sezione si è concentrata sul concetto di QoS nel campo delle interfacce di servizio. Misure di QoS possono essere introdotte anche per quanto riguarda i servizi digitali utilizzando metriche introdotte nei campi della Interazione Uomo-Macchina. Queste ultime sono fuori dagli obiettivi di questo documento.

Service Level Agreement - SLA

L'integrazione può coinvolgere numerose organizzazioni e erogatori esterni di interfacce di servizio. Al fine di accordarsi sulla QoS, erogatori di interfacce di servizio e fruitori utilizzano quelli che vengono definiti *Service Level Agreement - SLA*, ovvero *accordi sul livello di servizio*. Uno SLA può contenere le parti seguenti:

- *Scopo*. Le ragioni che hanno portato alla definizione dello SLA.
- *Parti*. I soggetti interessati nello SLA con i loro rispettivi ruoli (ad es., l'erogatore dell'interfaccia di servizio e il fruitore).
- *Periodo di validità*. L'intervallo di tempo, espresso mediante data e ora di inizio e data e ora di fine, per il quale si ritiene valido un particolare termine di accordo all'interno dello SLA.
- *Perimetro*. Quali sono operazioni interessate dallo specifico SLA.
- *Service Level Objectives - SLO*, ovvero *obiettivi sul livello di servizio*. I singoli termini di accordo all'interno di uno SLA. Di solito vengono definiti utilizzando dei *Service Level Indicators - SLI*, ovvero *indicatori sul livello di servizio*, che quantificano i singoli aspetti di QoS come indicato in questa sezione (ad es., disponibilità).
- *Penalità*. Le sanzioni che si applicano nel caso che l'erogatore dell'interfaccia di servizio non riesca ad assicurare gli obiettivi specificati nello SLA.
- *Esclusioni*. Gli aspetti della QoS non coperti dallo SLA.
- *Amministrazione*. I processi mediante i quali le parti possono monitorare la QoS.

Gli SLA possono essere statici o dinamici. Negli SLA dinamici, gli SLO (con associati SLI) variano nel tempo ed i periodi di validità definiscono gli intervalli di validità di questi ultimi (ad es., in orario lavorativo gli SLO possono essere differenti di quelli imposti durante la notte). La misurazione dei livelli di QoS all'interno di uno SLA richiedono il tracciamento delle operazioni effettuate in un contesto infrastrutturale multi-dominio (geografico, tecnologico e applicativo). In uno scenario tipico, ogni interfaccia di servizio può interagire con molteplici altre interfacce di servizio, cambiando il suo ruolo da erogatore a fruitore in alcune interazioni, ognuna governata da un differente SLA.

Recentemente, gli SLA hanno iniziato ad includere non soltanto vincoli relativi all'erogatore, ma anche vincoli che impongono ai singoli fruitori delle interfacce di servizio dei limiti relativi al ritmo ed alla quantità delle richieste. A tal fine gli erogatori devono definire ed esporre ai fruitori politiche di throttling[10] (anche noto come rate limiting) segnalando eventuali limiti raggiunti. Gli erogatori dovrebbero far rispettare le quote anche se se il sistema non è in sovraccarico, incentivando i fruitori a rispettarle.

Esempi di SLI sono i seguenti:

- dimensione massima di ogni richiesta accettata. Le richieste più grandi possono essere rifiutate;
- latenza al 90º percentile. Utilizzata per calcolare la responsività;

[7] Il payload è il contenuto informativo di un messaggio di rete (eliminando la parte relativa al protocollo). Per compressione del payload si intende applicare un algoritmo di compressione (molto spesso gzip) al payload in modo da ridurre il traffico di rete.

[8] Per paginazione si intende la capacità di una operazione nell'interfaccia di servizio di fornire un risultato composto da molte voci per singole pagine sfruttando un qualche criterio di ordinamento.

[118] https://tools.ietf.org/html/rfc7234

[9] Cf. https://tools.ietf.org/html/rfc7234

[10] Con il termine throttling (o rate limiting) si intendono le politiche intraprese dalle interfacce di servizio al fine di limitare la frequenza con cui i fruitori possono chiamare l'interfaccia o specifiche operazioni all'interno della stessa.

- percentuale di minuti negli ultimi 30 gg in cui l'interfaccia di servizio è stata disponibile;
- valori a 1 giorno e 30 giorni del success rate (ad es., il numero di chiamate terminate con successo rispetto al numero totale di chiamate);
- percentuale di minuti negli ultimi 30 gg in cui l'interfaccia di servizio è stata responsiva (ad es., il numero di chiamate con latenza inferiore ad un certo limite);
- tempo di risposta medio delle richieste totali (includendo le richieste rifiutate causa throttling) nell'ultimo giorno e negli ultimi 30 giorni;
- throughput misurato in bytes/s.

Gli SLI calcolati devono includere la latenza aggiuntiva dovuta ad eventuali componenti infrastrutturali e di rete (ad es., proxy-gateway).

Essi inoltre devono:

- utilizzare unità di misura referenziate dal Sistema Internazionale (ad es., secondi, bytes);
- indicare nel nome identificativo l'eventuale periodo di aggregazione coi soli suffissi s (secondi), m (minuti), d (giorni) e y (anni) utilizzando al posto dei mesi il numero di giorni.

Ove possibile, gli SLO e gli SLA dovrebbero essere in relazione diretta con i valori associati (ad es., indicare success rate anziché l'error rate), in modo che a valori più alti corrispondano risultati positivi.

3.1.5 Middleware

Con il termine middleware si intende lo strato software che separa le risorse informative dai fruitori delle interfacce di servizio, di fatto permettendo la realizzazione delle interfacce stesse. In tal senso un middleware gestisce la complessità e l'eterogeneità tipica dei sistemi distribuiti. Le risorse informative di cui si parla in questo caso possono essere nel caso più semplice delle basi di dati, ma più comunemente includono altre interfacce di servizio (che a loro volta possono essere implementate utilizzando dei middleware) e sistemi legacy a cui il middleware contribuisce a fornire interfacce moderne. A tale fine i middleware forniscono una serie di funzionalità:

- Il supporto a framework per l'esposizione di interfacce di servizio implementati in differenti tecnologie e secondo differenti schemi di interazione. In questo senso essi nascondono agli sviluppatori le complessità legate all'esposizione di interfacce di servizio secondo specifici protocolli di rete.
- Facilitano il riuso di componenti software.
- Forniscono una serie di funzionalità di supporto alla sicurezza dei sistemi informatici che includono autenticazione ed autorizzazione.
- Forniscono funzionalità di scalabilità che sfruttano la distribuzione su risorse hardware.
- Aiutano in generale a soddisfare i requisiti di QoS dichiarati negli SLA.
- Integrano funzionalità utili quali il throttling, logging e caching.

Oltre a mascherare l'eterogeneità dell'hardware, i middleware mirano anche a mascherare l'eterogeneità delle piattaforme software permettendo di sviluppare i diversi componenti del sistema distribuito secondo i linguaggi e framework più adatti.

API Management

Gli API Management System sono dei middleware che concentrano tutte le funzionalità necessarie ad una organizzazione per gestire le loro interfacce di servizio su infrastrutture on-premises e cloud pubblici e privati. Essi si concentrano sullo sviluppo delle interfacce di servizio, la gestione del ciclo di vita delle stesse, il controllo degli accessi (tramite meccanismi di autorizzazione ed autenticazione), il throttling, il caching e le analitiche (utili al controllo degli SLA).

Un API management system può essere utilizzato ad esempio come strato di accesso alle API interne ad una amministrazione, rilasciando solo una parte delle stesse e con politiche personalizzate verso l'esterno e verso l'intranet.

Oltre alle funzionalità richieste nelle sezioni precedenti, alcuni API management system permettono di definire processi di automazione ed orchestrazione di breve durata (dette soft-orchestration). Si tratta di orchestrazioni molto semplici in cui non ci si aspetta intervento umano nel processo, la durata è brevissima e le regole definite sono molto semplici.

Logging

Il logging riveste un ruolo fondamentale nella progettazione e sviluppo di interfacce di servizio. Le moderne piattaforme middleware, oltre ad integrare meccanismi di logging interni, possono connettersi ad interfacce di servizio esterne che permettono la raccolta (log collection), la ricerca e la produzione di analitiche utili tra l'altro all'identificazione di problemi e al monitoraggio del sistema e della QoS. L'utilizzo di log collector permette di centralizzare non solo i log relativi all'utilizzo dell'interfaccia di servizio, ma anche quelli di eventuali digital service e componenti di rete (ad es., proxy e application-gateway). I messaggi applicativi possono, ai fini di non ripudio (vedi Sezione 2.1.4) essere memorizzati assieme alla firma digitale e quindi archiviati periodicamente nel rispetto delle direttive sulla privacy.

L'erogatore deve documentare il dettaglio del formato della tracciatura e le modalità di consultazione e reperimento delle informazioni.

L'erogatore deve inoltre tracciare un evento per ogni richiesta, contenente almeno i seguenti parametri minimi:

- data e ora della richiesta in formato RFC3339[119][11] in UTC e con i separatori Z e T maiuscolo. Questa specifica è fondamentale per l'interoperabilità dei sistemi di logging ed auditing, evitando i problemi di transizione all'ora legale e la complessità nella gestione delle timezone nell'ottica dell'interoperabilità con altre PA europee;
- URI che identifica erogatore ed operazione richiesta;
- tipologia di chiamata (ad es., HTTP method per i protocolli basati su HTTP, basic.publish per AMQP);
- esito della chiamata (ad es., HTTP status per i protocolli basati su HTTP, SOAP fault nel caso di web services SOAP, OK/KO in assenza di specifici requisiti, eventuali messaggi di errore);
- identificativo del fruitore;
- ove applicabile, identificativo del consumatore o altro soggetto operante la richiesta comunicato dal fruitore - è cura del fruitore procedere a codifica e anonimizzazione ove necessario;
- ove applicabile, l'Indirizzo IP del client;
- ove applicabile, un identificativo univoco della richiesta, utile ad eventuali correlazioni tra chiamate diverse.

3.1.6 Attori e Interazioni

Come anticipato in «Presentazione del Modello di Interoperabilità», l'obiettivo a tendere è quello di una PA in cui le singole amministrazioni offrono interfacce di servizio, in corrispondenza ai servizi digitali che erogano, e possono a loro volta cooperare attraverso l'invocazione di interfacce di servizio offerte da altre PA.

L'EIF riprende la classificazione delle interazioni possibili in generale in Administration-to-Citizen (A2C), Administration-to-Business (A2B) e Administration-to-Administration (A2A), ulteriormente distinguendo se il fruitore del servizio è un soggetto umano od un modulo software, arrivando quindi a definire le seguenti possibili interazioni:

1. A2A in modalità *human-to-machine*;
2. A2A in modalità *machine-to-machine*;
3. A2B in modalità *human-to-machine*;
4. A2B in modalità *machine-to-machine*;
5. A2C in modalità *human-to-machine*.

[119] https://tools.ietf.org/html/rfc3339#section-5.6
[11] Cf. https://tools.ietf.org/html/rfc3339#section-5.6

In base al precedente confronto tra servizio digitale e interfaccia di servizio, la classificazione suddetta deve essere meglio specificata, al fine di individuare i giusti contesti di intervento.

A2A in modalità human-to-machine. In questo caso c'è una interazione tra due amministrazioni, di cui una offre un servizio digitale e l'altra, per il tramite di un suo operatore umano, ne fruisce al fine di espletare le proprie procedure. Ad es., un operatore di un Comune accede ad un servizio digitale dell'Agenzia delle Entrate per verificare la correttezza del codice fiscale. In questo caso, l'interfaccia di servizio viene sollecitata dalla logica di presentazione che l'erogatore offre agli operatori delle altre amministrazioni, ma non c'è un'invocazione diretta (si ricordi che un'interfaccia di servizio viene invocata solamente da altri moduli applicativi client, non è fruibile direttamente da utenti umani)

A2A in modalità machine-to-machine. In questo caso c'è una interazione tra due amministrazioni, in cui una offre un servizio digitale, ed espone una interfaccia di servizio, e l'altra realizza una propria applicazione/sistema/procedura digitale il cui software ha bisogno di invocare l'interfaccia offerta. Ad es., in un Comune viene realizzato un software (che utilizzano gli operatori allo sportello anagrafico) che durante la sua esecuzione invoca l'interfaccia di servizio dell'Agenzia delle Entrate per la verifica del codice fiscale. In questo caso l'interfaccia di servizio dell'erogatore è invocata direttamente dal modulo software del fruitore.

Va notata una differenza tra le due modalità. Nel primo caso, una esigenza operativa che richieda l'utilizzo di più servizi digitali per essere espletata, prevede l'utilizzo da parte degli operatori di più servizi digitali, e gli utenti hanno il compito di coordinare i vari servizi digitali, eventualmente muovere i dati/risultati da uno all'altro, ecc. Ovvero la composizione dei servizi digitali non può essere automatizzata, ma rimane in carico all'utente che utilizza i servizi digitali. Nel secondo caso, la composizione di servizi digitali può essere invece facilmente realizzata andando a sviluppare un nuovo servizio digitale, che compone le interfacce applicative degli erogatori e realizza la logica di coordinamento, a sua volta possibilmente offerta come interfaccia di servizio composta, al di sopra della quale offrire la logica di presentazione.

A2B in modalità human-to-machine. In questo caso c'è una interazione tra un'impresa ed un'Amministrazione che offre un servizio digitale. L'impresa sfrutta il servizio digitale per il tramite di un suo addetto umano che interagisce con il servizio. Ad es., un addetto di un'azienda accede ad un servizio digitale dell'Agenzia delle Entrate per verificare la correttezza dei codici fiscali.

A2B in modalità machine-to-machine. In questo caso c'è una interazione tra un'impresa ed un'Amministrazione a livello applicativo, ovvero una procedura software di un'impresa richiama le funzionalità offerte da un'interfaccia di servizio erogata da un'Amministrazione.

Tutte le considerazioni fatte sulle interazioni A2A human-to-machine e machine-to-machine si applicano anche a questi casi, fatta salva la trasposizione operatore di un'Amministrazione con addetto di un'azienda.

L'ultimo caso **A2C in modalità human-to-machine** è quello in cui un cittadino utilizza un servizio digitale erogato da un'Amministrazione.

Un cittadino non interagirà mai con l'interfaccia di servizio erogata, ma sempre con una logica di presentazione che a sua volta invoca, nel caso auspicabile di software progettato in modo stratificato, l'interfaccia di servizio.

Dal punto di vista funzionale (cf. Sezione 1.1) tutte le modalità machine-to-machine sono analoghe: per l'interfaccia di servizio, l'essere invocata da un modulo software è funzionalmente indipendente dalla natura dell'utente che siede di fronte alla logica di presentazione che si attesta su quel modulo (sia esso un operatore di un'altra Amministrazione o di un'azienda). La differenza è negli aspetti non funzionali, in particolare QoS e sicurezza, in quanto a seconda di chi è l'organizzazione fruitrice, l'erogatore potrebbe offrire differenti livelli di servizio, autorizzazioni, garanzie di sicurezza, ecc. L'utilizzo che il fruitore farà dell'interfaccia di servizio ha un impatto, soprattutto in termini di responsabilità, framework legale, ecc.; ad esempio, nel caso A2B, il caso in cui l'azienda fruitrice utilizza l'interfaccia all'interno di un proprio modulo applicativo, ovvero il caso in cui offre un servizio a valore aggiunto, devono essere differenziati; ma questo non ha impatti sugli aspetti tecnologici dell'interfaccia di servizio, bensì su quelli di governance, e verranno ripresi in «Governance del Modello di Interoperabilità». Tutti i casi human-to-machine sono analoghi: in questo caso non c'è interazione diretta con l'interfaccia di servizio, ma sempre per il tramite di una qualche logica di presentazione e la differenza è nella natura dell'utente umano che siede di fronte al modulo software che realizza tale logica di presentazione.

Emerge come la modalità di progettazione dei servizi digitali che stratifica chiaramente le interfacce di servizio separandole dalle logiche di presentazione, è la modalità corretta per supportare le possibili interazioni offerte da un'Amministrazione: a seconda della modalità diventa agevole stratificare la corretta logica di presentazione, ovvero moduli client, al di sopra della stessa interfaccia di servizio.

La tabella seguente riassume le considerazioni presentate.

Interazione	servizio digitale	interfaccia di servizio	richiede logica di presentazione	composizione di più servizi[12]
A2A human-to-machine	✓		✓	-
A2A machine-to-machine		✓		+
A2B human-to-machine	✓		✓	-
A2B machine-to-machine		✓		+
A2C	✓		✓	-

3.1.7 Uniformità dei dati

Uno degli aspetti maggiormente critici quando si espongono interfacce di servizio è la modellazione dei dati. Come anticipato nella Sezione 1.1, l'information model sottostante ad un servizio (e quindi anche ad un servizio digitale e interfaccia di servizio) serve a rappresentare sia il modello dei dati relativo ai cambiamenti di stato che il servizio opera, sia i dati che «transitano» (input/output) attraverso il servizio. Nel seguito ci soffermiamo sul caso delle interfacce di servizio. Facendo un parallelo con la programmazione orientata agli oggetti, oltre a definire i metodi offerti dalle classi del programma (nel parallelo corrispondenti alle operazioni dell'interfaccia di servizio), bisogna definire correttamente il numero e soprattutto il tipo dei parametri di input ed output. Non a caso, l'aspetto metodologico cruciale su cui si soffermano tutte le metodologie di progettazione e programmazione basate sul design-by-contract[13] è la definizione della segnatura dei metodi, al giusto livello di granularità, che comprende sia il nome del metodo che i parametri.

Il livello di granularità dipende da vari aspetti dell'interfaccia di servizio, in particolare se questa è atomica o composta, se il servizio a cui corrisponde è informativo o transazionale (cf. Sezione 1.1). Nella tabella seguente si forniscono delle indicazioni qualitative, da utilizzare come linee guida nella definizione delle interfacce di servizio. In «Profili e pattern di interoperabilità», esse saranno utilizzate nella definizione di vari possibili pattern che rispondono ad esigenze specifiche.

Tipo di interfaccia	Granularità[14]
Elementare	*fine-grained*
Composta	*coarse-grained*
Informativa	*fine-grained*
Transazionale	*coarse-grained*

Per quanto riguarda gli aspetti di formato dei dati delle interfacce di servizio, è importante

- omologare ove possibile i nomi delle variabili alle consuetudini europee abilitando l'interoperabilità con i servizi erogati dagli altri paesi;
- associare ai nomi dei campi dei metadati utili alla classificazione dei servizi;

[12] L'uso del +/- nell'ultima colonna dà un'indicazione qualitativa di quanto sia agevole comporre elementi nella specifica interazione. Come discusso, nel caso di servizi digitali la composizione è a cura dell'utente finale, che agisce da *human-ware* (ovvero deve farsi carico di realizzare, attraverso l'interazione stessa, la logica di composizione ed il passaggio di dati), mentre la composizione di interfacce di servizio è più semplice da automatizzare, e soprattutto può poi essere riusata più volte esponendo a sua volta come interfaccia di servizio composta. In quest'ultimo caso va però realizzata una logica di presentazione per il servizio digitale composta, se si vuole offrirlo agli utenti umani.

[13] Cf.

Meyer, Bertrand: *Design by Contract*, Technical Report TR-EI-12/CO, Interactive Software Engineering Inc., 1986

Meyer, Bertrand: *Design by Contract*, in *Advances in Object-Oriented Software Engineering*, eds. D. Mandrioli and B. Meyer, Prentice Hall, 1991, pp. 1–50

Meyer, Bertrand: *Applying «Design by Contract»*, in Computer (IEEE), 25, 10, October 1992

Meyer, Bertrand (1997). Object-Oriented Software Construction, second edition. Prentice Hall. ISBN 0-13-629155-4.

[14] La granularità è il livello di dettaglio con cui i dati sono esposti e scambiati. *Coarse-grained* significa un livello di dettaglio «basso», in quanto molti dettagli possono o devono rimanere interni all'implementazione dell'interfaccia di servizio. *Fine-grained* significa invece che il dato deve essere specificato ad un dettaglio massimo, poiché il fruitore ha bisogno di una visione puntuale del dato stesso.

- facilitare la validazione automatica delle specifiche dei vari servizi[15].

Inoltre è auspicabile che la specifica del formato sia coerente, od addirittura la stessa, tra varie tecnologie di esposizione delle interfacce di servizio[16].

Le indicazioni generali sono:

- per gli schemi dei dati, utilizzo di nomi basati su riferimenti europei (ad es., Core Vocabularies/Dizionari Controllati, Direttiva Europea INSPIRE 2007/2/CE[12017]) e standard de facto e de iure eventualmente disponibili sulla specifica tematica;
- UTF-8 come codifica di default[18];
- URI come identificatore del servizio e dell'erogatore[19];
- per i formati di serializzazione, semplicità di integrazione con strumenti di validazione (ad es. parsing);
- paesi, lingue e monete[20]: ISO 3166-1-alpha2 country[12121], ISO 4217 currency codes[12222];
- data e ora in RFC3339[12323], un sottoinsieme dell'ISO8601 ottimizzato per il web;
- aree amministrative NUTS 1 e successive: nomenclature NUTS[12424] (per il livello NUTS 0 - entità nazionali si fa riferimento ai codici ISO).

3.2 Concetti di Sicurezza

La sicurezza dei sistemi informatici è l'insieme di pratiche messe in atto al fine di impedire l'accesso non autorizzato, l'uso, la divulgazione, l'interruzione dell'accesso, la modifica, l'ispezione e la distruzione delle informazioni.

Questa sezione si concentra sui meccanismi di sicurezza che vadano oltre il semplice filtraggio di pacchetti basato su indirizzi IP, tipo di protocollo (anche detto circuit-level filtering) o contenuto del dato applicativo (application-level gateway o antivirus)[25]. In particolare la sezione si concentra sull'utilizzo di protocolli e tecniche di sicurezza basate sulla manipolazione dei messaggi di rete. La sezione farà inoltre riferimento a come i requisiti di sicurezza possano essere variabili a seconda dello scenario applicativo e del caso d'uso.

3.2.1 Meccanismi di base

Diversi sono i concetti chiave dietro al mondo della sicurezza. In origine il termine faceva riferimento al concetto di triade CIA (Confidenzialità, Integrità e Availability - Disponibilità). Nel tempo altri concetti si sono aggiunti quali l'autenticazione e il non ripudio.

Questa sezione descrive questi concetti introducendo le principali tecniche impiegate per assicurarli.

[15] Come anticipato in "Presentazione del Modello di Interoperabilità» ed approfondito in "Governance del Modello di Interoperabilità", la modellazione e specifica dei dati avviene nei Gruppi di Lavoro interni agli Ecosistemi, che indirizzano il lavoro di standardizzazione.

[16] Ad esempio, la serializzazione in JSON di un dato dovrebbe essere la medesima sia se viene esposto esternamente tramite REST API sia se transita da un messaging system interno all'amministrazione. Una rappresentazione opportuna permette quindi la fruizione del dato da sistemi diversi limitando il ricorso alle conversioni.

[120] https://joinup.ec.europa.eu/page/core-vocabularies

[17] Cf. https://joinup.ec.europa.eu/page/core-vocabularies e http://eur-lex.europa.eu/legal-content/IT/ALL/?uri=CELEX:32007L0002

[18] Vedi Linee Guida Patrimonio Pubblico. Architettura dell'Informazione del Settore Pubblico, http://lg-patrimonio-pubblico.readthedocs.io/it/latest/arch.html#formati-aperti-per-i-dati-e-documenti

[19] Gli URI vengono utilizzati anche dal gruppo DAF-Semantic per la nomenclatura delle ontologie e dei dataset

[20] Si noti che questi standard sono già usati nelle specifiche AgID sulle firme elettroniche e sul formato della fattura PA.

[121] https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2

[21] Cf. https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2

[122] https://en.wikipedia.org/wiki/ISO_4217

[22] Cf. https://en.wikipedia.org/wiki/ISO_4217

[123] https://tools.ietf.org/html/rfc3339#section-5.6

[23] Cf. https://tools.ietf.org/html/rfc3339#section-5.6

[124] https://it.wikipedia.org/wiki/Nomenclatura_delle_unit%C3%A0_territoriali_statistiche

[24] Cf. https://it.wikipedia.org/wiki/Nomenclatura_delle_unit%C3%A0_territoriali_statistiche

[25] Per questi si faccia riferimento alla letteratura, ad es., William Stallings (2017): Cryptography And Network Security, 7th edition.

Disponibilità

Il concetto di disponibilità è stato introdotto nella Sezione 1.3 parlando della QoS. Il concetto di disponibilità è legato strettamente anche a quello di sicurezza, poiché la disponibilità di una interfaccia di servizio può essere legata non solo a cause di natura tecnica ma anche a specifici tipi di attacco (ad es., denial of service).

Confidenzialità

Con il termine confidenzialità si intende la protezione dei dati e delle informazioni scambiate tra un mittente e un destinatario. La confidenzialità, declinata per il canale di comunicazione, è la proprietà di assicurare che l'informazione scambiata tra due entità colloquianti in rete non possa essere acceduta da soggetti terzi.

La confidenzialità è ottenuta tramite la cifratura dei dati e delle informazioni (sicurezza di messaggio) o del canale di comunicazione (sicurezza del canale).

In un metodo di cifratura, un messaggio in chiaro (anche chiamato plain text) viene trasformato in un messaggio codificato e viceversa. Gli algoritmi di cifratura si distinguono in meccanismi a chiave simmetrica (o privata o condivisa) e chiave asimmetrica (o pubblica). In entrambi i casi la lunghezza delle chiavi influenza la sicurezza della comunicazioni (chiavi più lunghe sono più sicure) perché proteggono maggiormente da attacchi a forza bruta. Si suppone infatti che ogni meccanismo di cifratura possa essere rotto tramite enumerazione a patto che il tempo necessario (esponenziale nella lunghezza della chiave) non sia troppo lungo rispetto agli scopi dell'attaccante. Un'altra tipologia di attacco ai metodi di cifratura (che si applica in particolar modo ai metodi a chiave simmetrica in cui le password sono generate da umani) sono quelli di tipo dizionario, basati sull'uso di parole di uso comune.

Nei meccanismi di cifratura a chiave privata, entrambe le parti (il mittente ed il destinatario) nel canale di comunicazione condividono la stessa chiave di cifratura che viene impiegata sia per cifrare che per decifrare il messaggio. La cifratura a chiave simmetrica è molto efficiente e viene utilizzata per la riservatezza di grandi quantità di dati (ad es., interi file). È necessario che le due parti abbiano condiviso la chiave privata con un metodo sicuro (ad es., scambiandola fisicamente di persona oppure tramite un meccanismo di cifratura a chiave pubblica, come si vedrà nella Sezione 2.4). Algoritmi noti di cifratura a chiave simmetrica sono RC4, DES, Triple DES, AES, IDEA e Camellia.

Nei meccanismi di cifratura a chiave pubblica, vengono utilizzate due chiavi diverse per la cifratura e la decifratura dei messaggi. In particolare si supponga che il destinatario abbia una coppia di chiavi di cui una è privata (conosciuta solo al destinatario) ed una è pubblica (conosciuta a tutti e liberamente inviata sulla rete anche in chiaro). Al fine di inviare un messaggio su di un canale sicuro, il mittente cifra il messaggio utilizzando la chiave pubblica del destinatario, ma questo potrà essere decifrato solo dal destinatario utilizzando la chiave privata. Per il destinatario infatti chiave pubblica e chiave privata sono state generate in modo da essere complementari. Il meccanismo a chiave pubblica risolve il problema della condivisione delle chiavi poiché la chiave pubblica può essere inviata su Internet senza pericolo (non può essere utilizzata per decifrare il messaggio). Come difetto, la crittografia a chiave pubblica soffre di basse prestazioni e per questo motivo viene utilizzata o nelle fasi preliminari necessarie a concordare una chiave privata di sessione condivisa (come nel caso di TLS) oppure per i meccanismi di firma digitale (quindi non a scopo di cifratura). L'algoritmo più diffuso per la cifratura a chiave pubblica è RSA (dai nomi degli inventori Rivest, Shamir e Adleman).

Integrità e Firma Digitale

Un messaggio in transito su una rete informatica può subire delle modifiche (ad esempio tramite attacchi di tipo man-in-the-middle). I meccanismi a chiave pubblica possono essere utilizzati ai fini di produrre delle prove, dette firme digitali, utili a verificare che il messaggio ricevuto sia uguale a quello inviato.

Il meccanismo di firma digitale prevede di inviare assieme al messaggio, un secondo messaggio (detto firma digitale) ottenuto dal primo:

- calcolando un riassunto (digest) del messaggio tramite tecniche cosiddette di hashing;
- cifrando il riassunto utilizzando la chiave privata del mittente.

Le tecniche di hashing utilizzate per la firma digitale sono progettate secondo diversi criteri. Tra cui:

- devono essere funzioni cosiddette one-way. Deve cioè essere facile calcolare il riassunto ma difficile risalire dal riassunto al testo originale. Questo viene anche facilitato dal fatto che i riassunti hanno solitamente lunghezza fissa;
- devono fare sì che piccolissime modifiche al messaggio in input generino significative differenze nel riassunto.

La tecnica di hashing più utilizzata per la firma digitale è Secure Hash Algorithm - SHA (disponibile in diverse versioni). Nel momento in cui un messaggio viene ricevuto, il destinatario utilizza la chiave pubblica del mittente per decifrare la firma digitale e verificare che essa corrisponda al riassunto del messaggio. La combinazione di tecniche di hashing e di cifratura a chiave pubblica assicura che un attaccante non possa modificare il messaggio e generare una firma valida per lo stesso, assicurando quindi l'integrità del messaggio stesso.

Non Ripudio e Public Key Infrastructure - PKI

Il meccanismo di firma digitale descritto in Sezione 2.1.3 assicura l'integrità del messaggio ma non ne assicura l'autenticità della fonte. In pratica, chi riceve un messaggio è sicuro che esso non ha subito modifiche durante il transito ma non è sicuro dell'identità del mittente. Il messaggio ricevuto non potrà quindi essere utilizzato ai fini del non ripudio, cioè come prova che uno specifico soggetto è il vero mittente del messaggio. Il problema principale risiede nella maniera in cui la chiave pubblica di un soggetto viene distribuita. Essa, come detto, viene posta pubblicamente su Internet ma niente vieta ad un attaccante di creare una coppia chiave pubblica / chiave privata e distribuire quest'ultima fingendosi un altro soggetto ed inviare per conto di questo, in maniera fraudolenta, dei messaggi. In altre parole chi riceve il messaggio non ha modo di verificare l'autenticità della chiave pubblica che sta utilizzando. A tal fine il meccanismo introdotto è quello della Public Key Infrastructure - PKI.

Nella PKI oltre al mittente ed al destinatario del messaggio, viene aggiunta una terza parte detta Certification Authority (Autorità di Certificazione) la quale emette dei certificati. Un certificato è un documento in chiaro contenente informazioni riguardanti l'identità dell'intestatario del certificato e la sua chiave pubblica e viene firmato dalla certification authority utilizzando la propria chiave privata.

La chiave pubblica della certification authority è installata nei sistemi operativi (e distribuita solitamente tramite gli aggiornamenti degli stessi), viene utilizzata per verificare che la chiave pubblica del mittente sia autentica. Il mittente invia assieme al messaggio firmato il suo certificato che viene validato utilizzando la chiave pubblica della certification authority che ha emesso il certificato stesso.

Il meccanismo PKI è sicuro fino a quando un attaccante non è in grado di installare sulle macchine del destinatario una public key fasulla per le certification authority. Per ovviare a questi problemi sono necessari dei meccanismi di sicurezza a livello di macchina che sono fuori dal perimetro di questo documento. Lo standard comunemente usato per i certificati è X.509.

Nel Modello di Interoperabilità, le amministrazioni dovranno acquistare certificati commerciali. Negli ultimi anni alternative all'approccio PKI sono state proposte (ad es., Web of Trust) ma il Modello attualmente ne vieta l'utilizzo.

Autenticazione

In un ambiente di calcolo distribuito, l'autenticazione è il meccanismo tramite il quale client e erogatore accertano le identità degli specifici utenti e sistemi per conto dei quali stanno operando. Quando la prova di autenticazione è bidirezionale si parla di mutua autenticazione.

L'autenticazione è spesso ottenuta in due fasi:

1. Si definisce un contesto di autenticazione effettuando una chiamata ad una entità di autenticazione diversa dall'erogatore;
2. Il contesto di autenticazione è impiegato per autenticarsi con l'altra parte della comunicazione.

Si noti come il meccanismo di non ripudio basato su PKI e firma digitale presentato in Sezione 2.1.4 sia un metodo di autenticazione ed in tal modo è usato in protocolli di strato di trasporto quali TLS (vedi Sezione 2.4) al fine di garantire non ripudio. Esistono poi dei protocolli di autenticazione a livello applicativo che forniscono dei vantaggi rispetto all'autenticazione basata su PKI:

- L'autenticazione basata su PKI solitamente non autentica solo i soggetti ma anche le macchine coinvolte (ad es., il certificato di un sito Internet contiene anche i nomi DNS su cui il sito risponderà);
- Possibilità di Single-Sign On - SSO. Il contesto di autenticazione definito con protocolli di strato applicativo può essere riutilizzato nell'interazione con diverse interfacce di servizio. Questo è dovuto al fatto che il client assume l'identità della persona o del soggetto per cui è stato creato il contesto di autenticazione;
- L'utilizzo di certificati è scomodo per l'utente finale e questo rende la mutua autenticazione basata su firma digitale meno adatta ai casi in cui siano utenti umani ad autenticarsi;
- Non sempre la funzionalità di non ripudio è richiesta e l'uso di certificati lato client risulta costoso.

A seconda dell'interfaccia di servizio utilizzata, l'autenticazione può essere debole o forte. Per autenticazione forte si intende una autenticazione che richiede almeno due fattori (ad es., nome utente/password e one-time password - OTP). I protocolli per autenticazione ed autorizzazione a livello applicativo più diffusi sono oggetto della Sezione 2.3.

Autorizzazione

I meccanismi di autorizzazione in ambienti distribuiti definiscono quali risorse possono essere accedute da uno specifico utente. Tipiche politiche di autorizzazione permettono l'accesso a specifiche collezioni a specifici gruppi di utenti autenticati sulla base di ruoli, gruppi e privilegi. L'autenticazione degli utenti è quindi una componente fondamentale nell'autorizzazione anche se i requisiti di autenticazione (forte o debole) possono cambiare a seconda del protocollo. Le politiche di autorizzazione sono le più svariate e possono interessare ad esempio l'ora del giorno in cui specifici utenti possono accedere a specifiche risorse oppure il rate massimo di chiamate concesse ad un utente.

3.2.2 Minacce alla sicurezza dei sistemi informatici

Nelle sezioni precedenti alcune minacce alla sicurezza sono state accennate. In questa sezione approfondiamo le diverse tipologie di attacchi. Non ci soffermeremo sugli attacchi basati su malware, ma ci limiteremo agli attacchi basati sull'uso dei protocolli di rete. I tipi di attacchi più comuni sono i seguenti:

- *Eavesdropping*. È un tipo di attacco passivo (senza modifica dei dati) in cui un attaccante riesce a rubare informazioni leggendo dati da una connessione non cifrata. I protocolli che assicurano confidenzialità difendono da questo tipo di attacco.
- *Modifica dei dati*. Un attaccante potrebbe riuscire a modificare i pacchetti in transito nella rete. I meccanismi di firma digitale difendono da questo tipo di attacco.
- *Identity spoofing*. In questo tipo di attacco, l'attaccante finge di essere un altro utente. Questo tipo di attacco è risolto mediante meccanismi di autenticazione.
- *Attacchi su base password*. In questo caso l'attaccante cerca di ottenere delle password, utilizzate ad esempio ai fini di autenticazione ed autorizzazione. Come già anticipato, gli attacchi basati su password si basano o su forza bruta oppure su metodi di tipo dizionario. Questo tipo di attacchi si evitano impostando politiche forti riguardo alle password utilizzate e metodi di autenticazione forte (a più fattori).
- *Denial of service - DoS*. In questo tipo di attacco l'attaccante mira a rendere non operativa una interfaccia di servizio inondandola di richieste e minandone quindi l'accessibilità. Difendersi da questi tipi di attacchi è in genere molto difficile (specialmente nella variante distribuita DDoS).
- *Attacchi man-in-the-middle*. In questo caso un attaccante si intromette come terza parte in una conversazione tra mittente e destinatario modificando i messaggi scambiati. Gli attacchi man-in-the-middle si combattono tramite tecniche di cifratura ed integrità degli scambi.

In alcuni casi, gli attaccanti possono sfruttare delle falle scoperte nei protocolli o nelle implementazioni. È quindi di fondamentale importanza tenere aggiornati i sistemi ed utilizzare quando possibile versioni aggiornate dei protocolli.

3.2.3 Protocolli per autenticazione e autorizzazione

Nel caso di autenticazione ed autorizzazione, occorre distinguere gli approcci utilizzati nello scenario human-to-machine e quelli utilizzati nello scenario machine-to-machine. I protocolli più comuni in ambito Web per autenticazione ed autorizzazione nel caso human-to-machine sono:

- OAuth2[126][26] è uno standard per l'autorizzazione;
- OpenID[127][27] è uno standard pensato per la sola autenticazione. L'ultima versione, denominata OpenID Connect[128][28], è costruita su OAuth2 in termini di scambio di messaggi;
- Security Assertion Markup Language - SAML[129][29] (la versione corrente è la 2) è il protocollo più vecchio in circolazione e copre l'autenticazione e in parte l'autorizzazione;
- eXtensible Access Control Markup Language - XACML[130][30] complementare a SAML per la gestione esaustiva degli aspetti di autorizzazione.

Nei protocolli human-to-machine, un client riceve autorizzazioni ad usare un certo tipo di risorsa per conto di un utente umano tramite le credenziali di quest'ultimo. La richiesta del token/assertion è effettuata per mezzo di uno user-agent (cioè un browser o una app mobile) che funge da intermediario.

Il ModI obbliga all'utilizzo di SPID per l'autenticazione human-to-machine o degli altri metodi indicati nell'art. 64 del Codice per l'Amministrazione Digitale[131][31] che includono anche la Carta d'Identità Elettronica - CIE e la Carta Nazionale dei Servizi - CNS.

SPID[132][32] è attualmente basato su SAML ma il supporto per OpenID Connect è in fase di definizione al fine di supportare in maniera più semplice l'autenticazione da piattaforme mobili.

In questo senso vale la pena esplorare le differenze principali tra SAML ed OpenID Connect (in breve Connect). Dal punto di vista della terminologia i due protocolli utilizzano termini differenti per gli stessi componenti:

- Identity Provider (SAML) o OpenID Provider (Connect) sono le entità che certificano l'identità dell'utente;
- Service Provider (SAML) o Relying Party (Connect) sono le interfacce di servizio, le app mobili o i siti presso cui l'utente vuole autenticarsi;
- Asserzioni (SAML) o Token (Connect) sono dei documenti firmati dall'Identity Provider (SAML) o dall'OpenID Provider (Connect) che contengono le informazioni circa l'utente identificato e le autorizzazioni che possiede.

La tabella seguente riassume le caratteristiche dei protocolli per l'interazione human-to-machine:

	OpenID Connect	**SAML + XACML**
Formato token/assertion	JSON	XML
Autorizzazione		✓
Autenticazione	✓	✓
Rischi per la sicurezza	Phishing[33]	XML Signature Wrapping[34]

[126] https://tools.ietf.org/html/rfc6749
[26] Cf. https://tools.ietf.org/html/rfc6749
[127] http://openid.net/developers/specs/
[27] Cf. http://openid.net/developers/specs/
[128] http://openid.net/connect/
[28] Cf. http://openid.net/connect/
[129] http://saml.xml.org/saml-specifications
[29] Cf. http://saml.xml.org/saml-specifications
[130] http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html
[30] Cf. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html
[131] http://www.agid.gov.it/cad/art-64-sistema-pubblico-gestione-identita-digitali-modalita-accesso-ai-servizi-erogati-rete
[31] Cf. http://www.agid.gov.it/cad/art-64-sistema-pubblico-gestione-identita-digitali-modalita-accesso-ai-servizi-erogati-rete
[132] http://spid-regole-tecniche.readthedocs.io/en/latest/
[32] Cf. http://spid-regole-tecniche.readthedocs.io/en/latest/

Uno scenario interessante nell'ambito dell'integrazione A2A e A2B è quello legato alla federazione di domini (ad es., due diverse amministrazioni) in cui alcuni utenti di un dominio devono essere autenticati ed autorizzati per accedere a risorse dell'altro dominio (una federazione può includere anche più di due domini). In ambito SOAP, gli standard più utilizzati sono WS-Federation[133][35] & WS-Trust[134][36] (vedi Sezione 3 per l'inquadramento nello stack WS-*). Soluzioni su altre tecnologie vengono sviluppate ad hoc.

Per quanto riguarda lo scenario machine-to-machine invece, come si vedrà nella sezione 2.4, l'autenticazione può avvenire a livello di trasporto utilizzando TLS.

Per quanto riguarda l'autorizzazione machine-to-machine invece è possibile utilizzare il protocollo OAuth2 nello specifico del flusso Client Credential Grant[135][37]. Tale flusso a differenza di quello standard non richiede la presenza di uno user-agent. Il client possiede invece delle proprie credenziali che vengono utilizzate per richiedere il token all'authorization server.

3.2.4 Protocolli per integrità e confidenzialità

Per ragioni storiche lo stack TCP/IP non ha di base funzionalità di sicurezza. I messaggi viaggiano in chiaro sulla rete. Poiché le tecnologie per l'integrazione che verranno introdotte utilizzano HTTP come principale protocollo di trasporto o applicativo[38], è importante che il canale di comunicazione sia protetto. La IETF definisce come standard per la securizzazione di TCP il protocollo Transport Layer Security - TLS. Con il termine HTTPS si definisce l'utilizzo di HTTP su canale TLS. Tutte le interfacce di servizio esposte nel ModI devono essere basate su HTTPS. Il protocollo TLS (ed il suo predecessore deprecato Secure Sockets Layer - SSL) assicurano su TCP confidenzialità (tramite cifratura) ed integrità (tramite firma digitale e PKI). Come introdotto in Sezione 2.1.5, il meccanismo di firma digitale assicura anche autenticazione ma questa è fatta machine-to-machine.

Il protocollo TLS (versione stabile corrente 1.2, draft 1.3 presentato a marzo) si basa come detto sull'utilizzo della firma digitale per lo scambio di una chiave di sessione da utilizzare come chiave simmetrica.

Per quanto riguarda i singoli algoritmi utilizzati:

- Per lo scambio della chiave di sessione, TLS supporta numerose tecniche. Tra quelle proposte, si impone l'uso di tecniche che evitano attacchi man-in-the-middle e forniscono la cosiddetta forward secrecy (cioè che la scoperta di una chiave privata usata nello scambio non permette di scoprire la chiave di sessione). Gli algoritmi di scambio delle chiavi permessi sono quindi ephemeral Diffie–Hellman - DHE ed ephemeral Elliptic Curve Diffie–Hellman - ECDHE.
- Per la cifratura TLS supporta numerosi algoritmi. Si suggeriscono i protocolli attualmente supportati nello standard TLS 1.3 e che sono considerati sicuri: Advanced Encryption Standard - AES (nella versioni GCM e CCM).
- Per l'integrità si suggerisce l'uso di SHA almeno a 256 bit (quindi a partire dal cosiddetto SHA-2).

[33] Per phishing si intende il tentativo di un attaccante di fingersi qualcun altro. Nel caso di OpenID Connect, in particolare, sia per quanto riguarda OpenID che OAuth2, diversi attacchi sono stati rivelati che permettono ad una relying party di redirezionare l'utente verso un identity provider falso.

[34] L'XML Signature Wrapping è una vulnerabilità non legata direttamente al protocollo ma presente in alcune implementazioni ed in diverse forme (cf., https://blog.netspi.com/attacking-sso-common-saml-vulnerabilities-ways-find/). Il tool SAML Raider può essere utilizzato per verificare la presenza della vulnerabilità.

[133] http://docs.oasis-open.org/wsfed/federation/v1.2/os/ws-federation-1.2-spec-os.html

[35] Cf. http://docs.oasis-open.org/wsfed/federation/v1.2/os/ws-federation-1.2-spec-os.html

[134] http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/ws-trust.html

[36] Cf. http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/ws-trust.html

[135] https://tools.ietf.org/html/rfc6749#section-4.4

[37] Cf. https://tools.ietf.org/html/rfc6749#section-4.4

[38] Ai fini dell'interoperabilità su Internet, la scelta di HTTP permette integrazione senza necessitare di regole particolari di inoltro o di definire Virtual Private Network - VPN.

Nel Modello di Interoperabilità, a prescindere dal profilo di autenticazione ed autorizzazione scelta (che dipende dal caso d'uso), il protocollo di trasmissione:

- DEVE essere basato su HTTP >= 1.1;
- DEVE essere cifrato tramite TLS >= 1.2;
- DEVE essere firmato con SHA-256 o superiore
- DEVE essere conforme alle misure minime AgID Basic Security Controls[41];
- Gli erogatori di interfacce di servizio DEVONO utilizzare l'header HSTS (HTTP Strict Transport Security) per evitare attacchi di tipo SSL Strip (tipo di attacco Man-in-the-middle).

Inoltre, ogni certificato TLS utilizzato per erogare interfacce di servizio:

- NON DEVE essere self-signed (ad es., CA:true);
- DEVE contenere i seguenti elementi: Subject, Key Identifier, Serial Number ed Issuer;
- DEVE avere il parametro keyUsage[136] con i seguenti bit: *digitalSignature, keyEncipherment*[42];
- DOVREBBE contenere i riferimenti al DNS dei domini serviti;
- Un certificato usato ai fini di non ripudio DEVE avere inoltre il parametro keyUsage con il bit nonRepudiation settato.

Numerose sono le minacce alla sicurezza a cui è esposto TLS (in special modo con vecchie versioni del protocollo accoppiate ad algoritmi per cifratura ed integrità vulnerabili). L'IETF nel 2015 ha rilasciato a riguardo una RFC informativa[43]. Per questo motivo, in determinati scenari che richiedono elevati standard di sicurezza, si aggiunge talvolta un ulteriore strato di sicurezza a livello applicativo.

Nel modello SPCoop si richiedeva che in ogni caso HTTPS fosse utilizzato con autenticazione mutual-TLS (vedi Sezione 2.3). Nel tempo sono emersi scenari di interazione con requisiti di sicurezza inferiori (ad es., solo HTTPS non-mutual-TLS), che non giustificano la complessità di un sistema a mutua autenticazione (ad es., accessi in sola consultazione, applicazioni Web o sistemi IoT[44]) a livello di trasporto. Fermo l'obbligo di usare HTTPS, nasce l'esigenza di venire incontro a diversi scenari e definire per essi modelli di autenticazione e di trust differenziati. Questi aspetti verranno definiti in «Pattern e Profili di Interoperabilità».

3.3 SOAP

Il protocollo SOAP (Simple Object Access Protocol) è stato sviluppato per superare le limitazioni imposte dai protocolli precedenti per l'interazione distribuita basata su oggetti (CORBA, Java/RMI, DCOM) relative alla distribuzione a livello Internet delle macchine interessate ed ai vincoli imposti dal punto di vista delle tecnologie di implementazione.

La versione corrente della specifica SOAP è la 1.2 del 27 aprile 2007[137][45]. La specifica definisce due stili di comunicazione (communication modes):

- quello basato su chiamata a procedura (RPC-like),
- e quello basato su scambio di documenti (document style).

In combinazione ad essi, il protocollo definisce delle modalità di scambio dell'informazione:

- interazioni one-way (dal client al server),
- interazioni request/response,

[41] Circolare AgID 18 aprile 2017, n.2/2017 http://www.gazzettaufficiale.it/eli/id/2017/05/05/17A03060/sg

[136] https://tools.ietf.org/html/rfc5280#section-4.2.1.3

[42] Cf. https://tools.ietf.org/html/rfc5280#section-4.2.1.3

[43] Cf. https://tools.ietf.org/html/rfc7457

[44] Un esempio potrebbe essere una interfaccia di servizio di un comune che permette di avere in tempo reale la situazione dei posti liberi nei parcheggi comunali. Un sistema di trasporto integrato regionale accede al dato su tutti i parcheggi dei comuni della regione e mostra in tempo reale la situazione aggregata dei parcheggi disponibili. In questo scenario, l'informazione scambiata (numero posti liberi) è poco sensibile e eventuali apparati installati presso i parcheggi non giustificano il costo necessario di una configurazione a prova di non ripudio ed una mutua autenticazione TLS. Esempi di tali scenari (con standard diversi da SPCoop) sono emersi in E015, sviluppato in occasione di Expo nella Regione Lombardia.

[137] https://www.w3.org/TR/soap12-part1/

[45] Cf. https://www.w3.org/TR/soap12-part1/

- invio di notifiche (interazione one-way dal server al client)
- e solicit/response (interazione request/response in cui la request è inviata dal server).

Le ultime due modalità sono poco utilizzate in pratica e fuori dai profili di interoperabilità standard, quindi il loro utilizzo è vietato.

Il protocollo SOAP definisce tre componenti fondamentali:

- una envelope (letteralmente «busta da lettere») che definisce la struttura del messaggio e come processarlo;
- un insieme di regole di codifica per esprimere istanze di tipi di dato definiti a livello applicativo;
- una convenzione per rappresentare lo stile di interazione RPC.

La definizione del protocollo è pensata per essere indipendente dal protocollo sottostante. In particolare, SOAP può operare (tramite i cosiddetti binding) su diversi protocolli di trasporto inclusi HTTP, SMTP, TCP, UDP o JMS. Sebbene implementazioni sono state proposte per ognuno di questi casi (in special modo JMS per interazioni asincrone), il mercato ha premiato principalmente soluzioni sincrone basate su HTTP.

Una delle caratteristiche che contraddistinguono il protocollo SOAP è la sua estensibilità. In particolare si indica con WS-* lo stack di estensioni costruite su SOAP, molte delle quali hanno avuto grande successo in termini di implementazioni disponibili. Queste estensioni permettono di avere su SOAP una serie di funzionalità che su altri protocolli devono essere costruite ad hoc. Lo svantaggio di questa soluzione è che il protocollo introduce un overhead di processamento che fa preferire altre soluzioni in determinati contesti.

Tra le estensioni supportate dai framework più diffusi abbiamo:

- WS-Addressing è un modo standard per includere informazioni circa l'instradamento dei messaggi (ad es., l'interfaccia di servizio a cui inviare la risposta o da contattare in caso di errore).
- WS-Security è la specifica che descrive le politiche di sicurezza implementate a livello applicativo dalle interfacce di servizio. In particolare, WS-Security include meccanismi per autenticazione e autorizzazione, confidenzialità, integrità e firma digitale.
- WS-Trust è una estensione a WS-Security che permette di richiedere, rinnovare e validare token di sicurezza. Permette inoltre di verificare la relazione di mutua fiducia su un canale sicuro.
- WS-Federation è una estensione che permette a differenti domini di sicurezza di scambiare informazioni circa identità, attributi di autorizzazione ed autenticazione.
- WS-ReliableMessaging permette di consegnare in maniera affidabile (ad es., nell'ordine corretto) messaggi SOAP in presenza di problemi di rete e di inattività di componenti software e di sistema.
- WS-AtomicTransaction è una estensione che permette di ottenere la proprietà tutto o niente per un gruppo di operazioni. Essa definisce tre protocolli (completamento, two-phase commit volatile e two-phase commit durevole) che sono implementati dal framework
- WS-Coordination.
- WS-Choreography è la specifica per la definizione di coreografie. Una coreografia specifica i passi relativi allo scambio di messaggi tra diversi soggetti che si integrano.
- WS-BPEL è la specifica per la definizione di orchestrazioni.
- WS-Coordination è un framework estensibile per il coordinamento di web service (corrispondenti alle interfacce di servizio). In particolare esso spiega come implementare (e quindi è preso a riferimento dalle varie implementazioni dello stack WS-*) i protocolli di coordinamento inclusi quelli descritti da WS-AtomicTransaction.

La specifica delle interfacce di servizio SOAP è effettuata tramite Web Services Description Language - WSDL[138][46]. Oltre ad indicare le funzionalità offerte dall'interfaccia di servizio dal punto di vista funzionale, esso permette anche di definire le caratteristiche non funzionali tramite le estensioni WS-Policy[139][47] che permettono di specificare le varie componenti della QoS. L'uso di WSDL evidenzia il tentativo di adattare al Web l'approccio

[138] https://www.w3.org/TR/wsdl20-primer/
[46] Cf. https://www.w3.org/TR/wsdl20-primer/
[139] https://www.w3.org/TR/ws-policy/
[47] Cf. https://www.w3.org/TR/ws-policy/

all'interoperabilità basato su chiamate remote; il WSDL non è altro che un IDL (Interface Description Language) per un componente software; l'esistenza di WSDL favorisce l'uso di tool per creare automaticamente client in un determinato linguaggio di programmazione.

3.3.1 Indicazioni di utilizzo

La specifica SOAP permette la definizione di specifici profili di interoperabilità, imponendo alcune restrizioni circa i tipi ed i formati scambiati. Il profilo di interoperabilità secondo il quale interfacce di servizio di tipo SOAP andranno implementate è la versione 2.0 del Basic Profile[14048] (nel seguito BP2) definito dal WS-I (Web Services Interoperability Organization) ed ora confluito in OASIS. BP2 è basato su SOAP 1.2 e WS-Addressing (per il dispatching dei messaggi a livello applicativo, in particolare nel caso di interazioni asincrone). Tra le molte indicazioni, BP2 definisce anche la modalità di gestione degli errori. In particolare, oltre all'utilizzo dei codici di errore HTTP si richiede che il ricevente sia in grado di gestire le SOAP fault[49] che quindi devono, obbligatoriamente, essere emesse dall'erogatore a fronte di errori.

3.3.2 Sicurezza

Per quanto riguarda la sicurezza, l'ultimo profilo standard definito da OASIS è il Basic Security Profile 1.1[14150]. Il profilo è datato ma le considerazioni sono ancora valide. Per quanto riguarda le versioni dei protocolli, si devono rispettare i vincoli imposti dal Modello di Interoperabilità in questo documento.

È importante, nel caso si richiedessero funzionalità di autorizzazione, autenticazione e non ripudio, oltre che di riservatezza (coperta dall'utilizzo obbligatorio di HTTPS[51]) fare affidamento alle tecnologie di autenticazione ed autorizzazione a livello applicativo. Il Basic Security Profile 1.1, basato sull'estensione WS-Security, suggerisce l'uso di SAML 2.0. Come detto, rispetto alle tecnologie di autenticazione ed autorizzazione, ci sono alcuni domini applicativi per i quali OAuth2 o OpenID sono più appropriati. In questi ultimi casi, fermo restando l'utilizzo della XML Signature definita in WS-Security per quanto riguarda il non ripudio, l'utilizzo di token di autorizzazione ed autenticazione non SAML richiede la definizione di request header custom[14252].

3.3.3 Uniformità e naming

Non esistono standard riguardanti il naming in ambito SOAP. Le best-practice prevedono l'utilizzo di CamelCase[14353] (con prima lettera maiuscola, anche noto come PascalCase) per endpoint, porte, operazioni e parametri.

Quando le risorse contengono link e riferimenti a risorse esterne, si dovrebbero usare le specifiche indicate in IANA registered link relations[14454] trasformando il Kebab Case[14555] utilizzato con il CamelCase.

[140] http://docs.oasis-open.org/ws-brsp/BasicProfile/v2.0/cs01/BasicProfile-v2.0-cs01.html
[48] Cf. http://docs.oasis-open.org/ws-brsp/BasicProfile/v2.0/cs01/BasicProfile-v2.0-cs01.html
[49] Le SOAP fault devono essere accompagnate anch'esse da un appropriato codice di errore HTTP. Per SOAP fault comuni si può fare riferimento a "https://www.w3.org/TR/2007/REC-soap12-part2-20070427/#tabresstatereccodes" <https://www.w3.org/TR/2007/REC-soap12-part2-20070427/#tabresstatereccodes>_.
[141] http://www.ws-i.org/Profiles/BasicSecurityProfile-1.1.html
[50] Cf. http://www.ws-i.org/Profiles/BasicSecurityProfile-1.1.html
[51] HTTPS è richiesto dal modello di interoperabilità ma non da BP2.
[142] https://developers.google.com/adwords/api/docs/guides/call-structure
[52] Cf. https://developers.google.com/adwords/api/docs/guides/call-structure
[143] https://it.wikipedia.org/wiki/Notazione_a_cammello
[53] Cf. https://it.wikipedia.org/wiki/Notazione_a_cammello
[144] http://www.iana.org/assignments/link-relations/link-relations.xml
[54] Cf. http://www.iana.org/assignments/link-relations/link-relations.xml
[145] https://it.wikipedia.org/wiki/Kebab_case
[55] Cf. https://it.wikipedia.org/wiki/Kebab_case

3.4 REST

REpresentational State Transfer (REST) è uno stile architetturale, proposto da Roy Fielding[14655], che consente di accedere e manipolare rappresentazioni testuali di risorse web usando un insieme predefinito di operazioni stateless. Le interfacce di servizio che seguono lo stile architetturale REST sono dette RESTful o semplicemente REST. Con il termine «risorsa web» si intendevano inizialmente documenti e file identificati da una URL sul World Wide Web. Oggi il termine ha un'accezione molto più generica ed astratta, andando ad indicare ogni cosa o entità che possa essere identificata tramite una URI (si noti il passaggio da URL ad URI che indica l'indipendenza dal protocollo di recupero dei dati). Nel caso dell'applicazione di questo stile architetturale ad HTTP, le operazioni stateless a cui si fa riferimento sono GET, POST, PUT, DELETE a cui corrispondono operazioni di tipo Create-Read-Update-Delete - CRUD sulla risorsa. Questo approccio favorisce l'uniformità delle interfacce di servizio.

Il termine «state transfer» indica che è il client a dovere riportare tutte le informazioni necessarie al soddisfacimento di una richiesta, e il server non memorizza alcun tipo di informazione circa la sessione; quindi le interfacce di servizio sono, per definizione, stateless. Questo tipo di approccio favorisce l'introduzione di meccanismi di caching. In particolare, le risposte del server devono contenere una indicazione sul fatto che le risposte possano essere messe in cache o meno. Opzionalmente, inoltre, è possibile per il server richiedere l'esecuzione di alcune funzionalità al client tramite il passaggio di codice da eseguire (ad es., codice JavaScript da eseguire nel browser).

Talvolta, il termine Resource Oriented Architecture - ROA è usato per denotare l'architettura REST in opposizione alle Service Oriented Architecture - SOA, indicando la predilezione della prima per l'accesso basato su risorsa più che sulla chiamate ad operazioni di tipo RPC. Il dibattito sulla correttezza o meno di implementare operazioni RPC utilizzando REST è molto acceso, ma come dato di fatto numerose iniziative di API commerciali e non, utilizzano interfacce di servizio REST anche per effettuare RPC. Il concetto di REST è inoltre molto spesso legato, anche se non per definizione, alle architetture dette a microservizi[56], caratterizzate da elevata modularità, per via della leggerezza del protocollo.

A differenza delle interfacce di servizio SOAP, per cui una serie di standard è definita e mantenuta da OASIS (cf. stack WS-*), per le interfacce REST sono disponibili singoli standard e best-practice.

Per la specifica delle interfacce REST esistono due grandi iniziative: OpenAPI e RAML. Sebbene simili dal punto di vista dello sviluppatore di interfacce di servizio, la specifica RAML è più indirizzata alla creazione automatica di server e di client per API, mentre OpenAPI (attualmente nella versione OpenAPI v3) contiene elementi più descrittivi per la documentazione e la catalogazione (che invece sono disponibili in RAML come estensioni ad hoc) e si sta imponendo come standard de facto. Altri standard proposti in passato, quali Web Application Description Language - WADL, hanno avuto scarso successo e nei framework in cui sono stati utilizzati si sta optando per il passaggio ad OpenAPI v3[14757]. Per queste ragioni il ModI impone l'uso di OpenAPI v3.

Un file OpenAPI consente di descrivere l'intera interfaccia (endpoint disponibili e operazioni su ciascun endpoint, parametri di input e output per ogni operazione, metodi di autenticazione, informazioni di contatto, licenza, termini di utilizzo, ecc.). Le specifiche possono essere scritte in YAML o JSON. Swagger[164] è un insieme di strumenti open source che, basandosi sulle specifiche OpenAPI, permette di supportare il progetto, costruzione e documentazione di REST Web service.

È possibile assicurare la conversione tra le differenti rappresentazioni delle interfacce REST tramite tool automatici.

Legato al concetto di specifica nel mondo REST è quello di *Hypermedia As The Engine Of Application State - HATEOAS*. Secondo questo approccio, accedendo ad una risorsa, la risposta del server contiene hyperlink ad altre azioni che possono essere eseguite sulla risorsa[58]. HATEOAS permette di scoprire dinamicamente le operazioni

[146] http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[55] Cf. http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[56] Cf. Sam Newman (2015): Building Microservices.

[147] https://www.openapis.org/

[57] Cf. https://www.openapis.org/

[164] Originariamente Swagger (della società SmartBear Software) era un insieme di tool sia per la descrizione delle interfacce che per il loro sviluppo. Nel 2015 un gruppo di aziende, sotto la sponsorship della Linux Foundation, ha dato vita all'iniziativa OpenAPI, a cui SmartBear ha donato il formato di specifica che è stato rinominato da Swagger Specification in OpenAPI Specification. OpenAPI 3.0 è l'ultima versione della specifica. Gli strumenti Swagger, che sono ancora supportati da SmartBear Software, sono tra gli strumenti più popolari per implementare la specifica OpenAPI e continueranno a mantenere il nome Swagger. Esistono molti altri strumenti open source e proprietari, non correlati a Swagger, che supportano la specifica OpenAPI.

[58] Si supponga ad esempio una operazione HTTP GET http://api.domain.com/management/departments che restituisce informazioni circa i reparti. Il singolo reparto può contenere link relativi ad altre operazioni come quella per ottenere gli impiegati del repar-

presenti in una interfaccia di servizio e può essere utilizzato come approccio complementare (non sostitutivo) alla specifica.

3.4.1 Indicazioni di utilizzo

L'interfaccia di servizio REST deve utilizzare l'HTTP verb più adatto all'operazione come indicato in RFC 7231[14859]. In particolare i metodi:

- GET, HEAD, DELETE: non devono avere un payload.
- GET, HEAD: devono essere «safe», cioè devono essere essenzialmente read-only. Il client in questo caso non si aspetta e non richiede un cambiamento dello stato della risorsa.
- GET, HEAD, PUT, DELETE: devono essere idempotenti, cioè chiamate multiple con richieste identiche si comportano come singole richieste.
- POST: dovrebbe implementare un meccanismo di idempotenza per evitare di duplicare eventuali entry.

Ove necessario, specialmente ai fini del caching e l'accesso concorrente alle risorse[60], occorre fare leva sugli ETag[14961] (degli identificatori univoci di versione delle risorse). Infine l'utilizzo di eventuali header HTTP non deve sostituire i parametri da passare in una GET.

3.4.2 Sicurezza

Lo standard di riferimento per la firma e la crittografia in ambito JSON/REST è Javascript Object Signing and Encryption[15062] (di seguito JOSE), menzionato nelle Linee Guida AgID[15163] ed in «European Telecommunications Standards Institute - Security of the mission critical service»[15264]. JOSE è un framework per la sicurezza comprendente diverse componenti tra cui centrale è il JSON Web Token[15365] (di seguito JWT). JWT è uno standard per la definizione di token di accesso basato su JSON Web Signature[15466] (di seguito JWS) e JSON Web Encryption[15567] (si seguito JWE) di cui eredita ed estende gli header. Il token JWT è passato in REST tramite l'header HTTP Authorization utilizzando lo schema Bearer[68]. Il token in OpenID Connect è espresso per esempio direttamente come JWT.

Per ulteriori dettagli sulla sicurezza, si vedano anche:

- OWASP REST Security Cheat-Sheet[15669];
- OWASP API Security Project[15770];

to:{«departmentId»: 10,»departmentName»: «Administration»,»links»: [{«href»: «[[http://api.domain.com/management/departments/10/employees]{.underline}](http://api.domain.com/management/departments/10/employees)»,»rel»: «employees», «type» : «GET» }]}

[148] https://tools.ietf.org/html/rfc7231#section-4.3

[59] Cf. https://tools.ietf.org/html/rfc7231#section-4.3

[60] C.f. https://en.wikipedia.org/wiki/Optimistic_concurrency_contro

[149] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/ETag

[61] Cf. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/ETag

[150] http://www.etsi.org/deliver/etsi_ts/118100_118199/118103/02.04.01_60/ts_118103v020401p.pdf

[62] Cf. http://www.etsi.org/deliver/etsi_ts/118100_118199/118103/02.04.01_60/ts_118103v020401p.pdf

[151] http://www.agid.gov.it/agenda-digitale/infrastrutture-architetture/cert-pa/linee-guida-sviluppo-sicuro

[63] Cf. http://www.agid.gov.it/agenda-digitale/infrastrutture-architetture/cert-pa/linee-guida-sviluppo-sicuro

[152] http://www.etsi.org/deliver/etsi_ts/133100_133199/133180/14.02.00_60/ts_133180v140200p.pdf

[64] Cf. http://www.etsi.org/deliver/etsi_ts/133100_133199/133180/14.02.00_60/ts_133180v140200p.pdf

[153] https://tools.ietf.org/html/rfc7519

[65] Cf. https://tools.ietf.org/html/rfc7519

[154] https://tools.ietf.org/html/rfc7515

[66] Cf. https://tools.ietf.org/html/rfc7515

[155] https://tools.ietf.org/html/rfc7516

[67] Cf. https://tools.ietf.org/html/rfc7516

[68] Lo schema Bearer, inizialmente introdotto nella specifica OAuth2 ma poi utilizzato in altri contesti, ha la forma «Authorization: Bearer <token>» dove il token JWT è codificato in base64.

[156] https://www.owasp.org/index.php/REST_Security_Cheat_Sheet

[69] Cf. https://www.owasp.org/index.php/REST_Security_Cheat_Sheet

[157] https://www.owasp.org/index.php/OWASP_API_Security_Project

[70] Cf. https://www.owasp.org/index.php/OWASP_API_Security_Project

- JWS - Security Considerations[15871].

3.4.3 Uniformità e Naming

In questa sezione introduciamo le best practice da utilizzare per interfacce di servizio REST. In prima istanza, ogni endpoint deve essere univocamente associato alle componenti Scheme, Authority e Path di un URL[15972].

La componente Authority dell'URL:

- dovrebbe essere associata al dominio del sito Istituzionale dell'erogatore presente su IndicePA, anche tramite il prefisso «api», ad esempio un erogatore con sito istituzionale «erogatore.gov.it», potrebbe usare come authority «api.erogatore.gov.it»;
- può essere associata al dominio di un ente che l'erogatore ha delegato (ad es., una società in-house, un consorzio di comuni).

Per quanto riguarda la componente Path, i nomi utilizzati non devono usare abbreviazioni e acronimi non universalmente riconosciuti[73]. Inoltre, il Path dovrebbe essere semplice, intuitivo e coerente[74].

Il campo Query dovrebbe:

- essere in snake_case minuscolo;
- non essere in camelCase;
- utilizzare ove possibile dei nomi comuni per le funzionalità di paginazione, ricerca ed embedding/resource-expansion (ad es., limit, offset, q, sort).

Le risposte in formato JSON[16075], devono restituire sempre oggetti strutturati con attributi chiave-valore, non semplici liste. Questo permette di estendere facilmente le risposte introducendo in versioni successive ulteriori attributi (ad es., di paginazione).

In caso di errore, le risposte devono usare schemi standard come quello definito nella RFC 7807 - Problem Details for HTTP APIs - IETF Tools[16176] in particolare utilizzando il content type application/problem+json nella risposta.

Quando le risorse contengono link e riferimenti a risorse esterne, si dovrebbero usare le specifiche indicate in IANA registered link relations[16277].

Tutti i riferimenti dovrebbero contenere URL comprensivi di schema.

[158] https://tools.ietf.org/html/rfc7515#section-10
[71] Cf. https://tools.ietf.org/html/rfc7515#section-10
[159] https://tools.ietf.org/html/rfc3986
[72] Cf. https://tools.ietf.org/html/rfc3986
[73] Cf. https://linee-guida-cataloghi-dati-profilo-dcat-ap-it.readthedocs.io/it/latest/catalogo_elementi_obbligatori.html#titolo-dct-title Ad esempio,

- sono ammesse stringhe come «id», «args» o «stdin» ed abbreviazioni come «tcp» ed «udp»;
- stringhe come «codice fiscale» andrebbero espresse per esteso con «codice_fiscale» o «tax_code», e non con «cod_fiscale», «cod_fisc» o «cf».

[74] Alcune indicazioni in questo senso:

- usare parole minuscole separate da trattino «-«;
- usare nomi al plurale per le risorse e al singolare per l'accesso alla singola risorsa;
- ispirarsi alle convenzioni utilizzate a livello europeo (ad es., Core Vocabularies/Dizionari Controllati, Direttiva Europea INSPIRE 2007/2/CE);
- non contenere verbi (ad es., api.example.com/ospedale/prenota/);
- uniformarsi a quello di altre interfacce di servizio a livello Europeo quando ciò vada nella direzione dell'interoperabilità e della semplicità.
- In generale tutte le stringhe in inglese dovrebbero utilizzare la dizione US per evitare ambiguità (come ad es., «color» vs «colour», «flavor» vs «flavour»).

[160] https://tools.ietf.org/html/rfc7159
[75] Cf. https://tools.ietf.org/html/rfc7159
[161] https://tools.ietf.org/html/rfc7807
[76] Cf. https://tools.ietf.org/html/rfc7807
[162] http://www.iana.org/assignments/link-relations/link-relations.xml
[77] Cf. http://www.iana.org/assignments/link-relations/link-relations.xml

3.4.4 Throttling ed indisponibilità del servizio

Nelle API basate su REST, meccanismi di throttling vengono implementati al fine di garantire l'accessibilità delle interfacce di servizio ed evitare in alcuni casi la raccolta non autorizzata (web-harvesting) dei dati.

Poiché l'RFC 6585 prevede per la gestione del throttling il solo status code 429, nel Modl si richiede di notificare al fruitore lo stato del throttling ed eventuali limiti come segue:

- restituire in ogni risposta valida i valori globali di throttling tramite i seguenti header HTTP:
  - X-RateLimit-Limit: limite massimo di richieste per un endpoint;
  - X-RateLimit-Remaining: numero di richieste rimanenti fino al prossimo reset;
  - X-RateLimit-Reset: il numero di secondi mancanti al momento in cui il limite verrà reimpostato.
- utilizzare gli HTTP status code nelle risposte:
  - HTTP 429 (too many requests), insieme ai rate limit di cui al punto precedente, se il rate limit viene superato;
  - HTTP 503 (service unavailable) se l'infrastruttura non può erogare le operazioni offerte nei tempi attesi (definiti dalla SLA associata all'interfaccia di servizio).
- nei casi 429 e 503 gli erogatori dovrebbero notificare al client dopo quanti secondi ripresentarsi tramite l'header Retry-After[163][78] (pratica anche detta "circuit breaker"), anche implementando meccanismi di exponential back-off. L'RFC prevede che questo header possa essere utilizzato sia in forma di data che di secondi, ma il Modl vieta l'utilizzo del formato data poiché se non implementato correttamente potrebbe aggravare lo stato dei sistemi.

I fruitori dell'interfaccia di servizio devono impegnarsi a rispettare le indicazioni provenienti dagli header e dagli status code di cui sopra.

3.5 Message Broker

Un message broker è un modulo software che permette l'integrazione asincrona tramite scambio di messaggi. Questo tipo di interazione è fortemente disaccoppiata perché l'invio del messaggio avviene su un canale in cui è responsabilità del message broker consegnare il messaggio ai soggetti interessati. Il compito del message broker non è però solo quello di passare dati, in quanto esso si occupa anche di aspetti legati alla sicurezza, priorità dei messaggi, inoltro ordinato.

I middleware focalizzati sul fornire integrazione basata su messaggi vengono detti Message Oriented Middleware - MOM.

Un message broker supporta solitamente diverse modalità di interazione:

- Publish/Subscribe. In questo scenario un publisher invia dei messaggi sul canale ed il message broker li invia a diversi ricevitori sulla base di sottoscrizioni. Questo tipo di interazione supporta diversi scenari tra cui uno a molti o molti a molti;
- Queuing. In questo caso un fruitore invia una richiesta su una coda specifica (corrispondente all'erogatore) e l'erogatore invia la risposta sulla medesima coda; di fatto è una realizzazione asincrona della modalità request/reply;
- Store/Forward. In questo caso il broker memorizza i messaggi e quindi inoltra agli interessati.

Un caso particolare di message broker è costituito dagli integration broker. Rispetto ad un message broker, questi si occupano anche della trasformazione di messaggi dai formati sorgente a quelli manipolabili dai riceventi/sottoscrittori.

L'utilizzo di message broker è consigliato in alcuni casi d'uso in cui l'interazione è asincrona o di tipo publish/subscribe (ad es., Internet-of-Things - IoT, aggregatori di dati pubblici).

[163] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Retry-Afte
[78] Cf. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Retry-After

Varie tecnologie e realizzazioni di message broker hanno storicamente supportato svariati protocolli quali STOMP[16580], XMPP[16681], MQTT[16782], OpenWire[16883] e AMPQ[16984]. Oggigiorno, sebbene in determinati contesti essi vengano attualmente ancora utilizzati (ad es., in contesti intra-dominio o in casi particolari quali l'IoT in cui si preferiscono protocolli binari efficienti come MQTT), si preferiscono, in ambito di integrazione di sistemi, approcci in cui l'interfacciamento con i message broker avviene tramite interfacce di servizio REST. In particolare sono disponibili sia soluzioni native che wrapper per implementazioni di altri protocolli.

I vantaggi di questo approccio includono la possibilità di utilizzare le modalità di autenticazione, autorizzazione, throttling ed accounting già discussi riguardo alla tecnologia REST, e la risoluzione di possibili problematiche legate all'attraversamento di firewall e proxy.

Sebbene, a seconda delle implementazioni, le diverse interfacce di servizio REST per l'accesso a message broker differiscano per funzionalità offerte e modi di modellare code, topic/sottoscrizioni, si possono astrarre i seguenti comportamenti dei metodi HTTP:

- il metodo POST viene utilizzato per l'invio di messaggi e la creazione di topic/sottoscrizioni e code;
- il metodo GET viene utilizzato per consumare messaggi da code e topic/sottoscrizioni;
- il metodo DELETE viene utilizzato per l'eliminazione di topic/sottoscrizioni e code ed in alcuni casi per segnalare il fatto che un messaggio è stato consumato.

Il metodo PUT viene di solito utilizzato per modificare le proprietà di topic/sottoscrizioni e code.

3.6 Considerazioni comparative

Un primo criterio per orientarsi tra le tecnologie di integrazione presentate nelle Sezioni 3, 4 e 5 è quella di distinguere le tecnologie adatte per una interazione sincrona da quelle adatte ad una interazione asincrona. Riguardo a questa distinzione si può fare riferimento alla seguente tabella:

	SOAP	**REST**	**Message Broker**
Interazione Sincrona	✓	✓	
Interazione Asincrona	✓*	✓*	✓

SOAP (inteso come stack WS-*), come si evince dalla tabella, può essere utilizzato sia per interazioni sincrone che per interazioni asincrone.

In particolare, in SOAP, l'interazione asincrona può essere ottenuta sia su protocolli di trasporto sincroni che su protocolli di trasporto asincroni. Nonostante la specifica supporti questo genere di interazioni, il supporto di middleware e framework di sviluppo a queste funzionalità è limitato. Per quanto riguarda REST invece, nonostante non originariamente previsto dalla specifica, si è visto in Sezione 5 come esso venga utilizzato come interfaccia di servizio per message broker.

Per quanto riguarda l'interazione sincrona (stile chiamata a procedura o accesso CRUD a risorsa), diverse considerazioni tecnologiche devono essere effettuate. SOAP e REST utilizzano HTTP in due modi differenti.

Mentre SOAP lo utilizza come un protocollo di trasporto, REST lo utilizza come un protocollo applicativo. La diffusione dell'accesso alla rete ha aumentato il carico sulle infrastrutture IT, inoltre reti migliori hanno aumentato le aspettative in termini di latenza. L'IETF nel tempo ha risposto a queste esigenze:

- migliorando la semantica di HTTP, facendo leva sui campi Header, Status e Method RFC7230[17085],

[165] https://stomp.github.io/
[80] Cf. https://stomp.github.io/
[166] https://xmpp.org/
[81] Cf. https://xmpp.org/
[167] http://mqtt.org/
[82] Cf. http://mqtt.org/
[168] http://activemq.apache.org/openwire.html
[83] Cf. http://activemq.apache.org/openwire.html
[169] https://www.amqp.org/
[84] Cf. https://www.amqp.org/
[170] https://tools.ietf.org/html/rfc7230
[85] Cf. https://tools.ietf.org/html/rfc7230

RFC7231[171][86];

- codificando le semantiche di caching RFC7234[172][87] e controllo della concorrenza RFC7232[173][88] per facilitare l'implementazione di interfacce di servizio stateless, che possano scalare senza che i bilanciatori conoscano la logica applicativa;
- orientandosi verso formati più leggeri (ad es., JSON).

Queste innovazioni fanno preferire l'approccio REST:

- Quando è possibile esprimere la logica applicativa tramite la semantica HTTP, poiché si guadagna:
  - espressività (ad es., il contesto d'utilizzo è chiarito da Method e Status);
  - mobile-ready (esporre un'API in un'app con un http-wrapper);
  - performance e scalabilità (ad es., possibilità di ruotare le chiamate in base al Method, senza inspection applicativa).
- Quando le API devono essere fruibili anche da mobile/web;
- L'accesso avviene in maniera prevalente con operazioni di tipo CRUD sui dati.

Quindi rispetto a quanto discusso in "Presentazione del Modello di Interoperabilità" sui paradigmi di cooperazione, questo suggerisce l'uso di REST nei casi di condivisione di dati e di composizione applicativa, quando le operazioni componenti sono prevalentemente orientate a fornire dati. Il servizio digitale corrispondente all'interfaccia di servizio è prevalentemente informativo (cf. Sezione 1).

L'utilizzo di SOAP è suggerito:

- Quando la semantica HTTP non è sufficiente ad esprimere la logica applicativa ed è necessario usare un protocollo di messaging ulteriore con dei propri header;
- Se la specifica applicazione richiede la creazione di interfacce di servizio principalmente *stateful*, cioè l'accesso ad informazioni di contesto o la gestione dello stato della conversazione[89]. SOAP prevede estensioni (eg. relative al concetto di transazione) che con altri approcci (ad es., REST) devono essere costruite ad hoc per la specifica applicazione.
- Nel caso si necessiti di processamento asincrono che non sia possibile implementare con semantiche HTTP;
- Quando servono specifiche assicurazioni circa la QoS (quali quelle fornite dall'estensione WS-ReliableMessaging).

Quindi rispetto a quanto discusso in "Presentazione del Modello di Interoperabilità" sui paradigmi di cooperazione, questo suggerisce l'uso di SOAP nei casi di processo inter-PA e di composizione applicativa quando le operazioni componenti offrono delle logiche complesse.

La tabella seguente riporta alcuni aspetti tecnologici che devono essere tenuti in considerazione (le celle in cui è presente «-» indicano che l'aspetto in questione non è considerato e standardizzato, e quindi è a cura dello specifico progetto/applicazione indirizzarlo attraverso sviluppi ad hoc)

[171] https://tools.ietf.org/html/rfc7231
[86] Cf. https://tools.ietf.org/html/rfc7231
[172] https://tools.ietf.org/html/rfc7234
[87] Cf. https://tools.ietf.org/html/rfc7234
[173] https://tools.ietf.org/html/rfc7232
[88] Cf. https://tools.ietf.org/html/rfc7232
[89] Come nel caso di processi amministrativi sia completamente automatizzati (short-running) sia con intervento umano o comunque long-running.

	SOAP (WS-*)	REST
Formato Payload	XML	Tutti (JSON nella maggior parte dei casi)
Identificazione delle operazioni	URI, WS-Addressing	URI
Descrizione delle interfacce di servizio	WSDL	RAML, OpenAPI
Affidabilità	WS-ReliableMessaging	-
Sicurezza	HTTPS, WS-Security	HTTPS, JWT
Transazioni	WS-AtomicTransaction, WS-BusinessActivity	-
Composizione di interfacce di servizio	WS-Choreography WS-BPEL	-

In letteratura, talvolta si indica con contract-first una metodologia di progetto che parte dalla specifica dell'interfaccia senza considerare possibili vincoli di implementazione, e successivamente si occupa di come realizzare tale interfaccia al di sopra di eventuali realizzazioni esistenti. In alternativa, si parla di contract-last (che potremmo anche indicare come implementation-first) quando invece eventuali vincoli di realizzazione guidano la specifica dell'interfaccia. SOAP supporta naturalmente entrambi gli approcci, in quanto lo sviluppo di un'interfaccia di servizio origina dalla definizione dell'interfaccia o dalle segnature dei metodi utilizzati nello sviluppo, mentre in REST l'interfaccia è definita dagli http verb associati alle operazioni CRUD, riportando il contratto alla definizione delle risorse. La differenza appare ininfluente nel caso di progettazione e realizzazione di sistemi nuovi, ma non in presenza di sistemi legacy. Quando l'interfaccia di servizio è vincolata dalla presenza di un sistema esistente o legacy, essa è definita a posteriori rispetto all'implementazione. In questo caso non essere limitati dai verb http (eg. usando SOAP) appare semplificare il lavoro di modellazione e realizzazione dell'interfaccia di servizio evitando di mappare risorse su procedure legacy.

Talvolta si parla di REST indicandolo come contract-less (REST)[90], proprio ad indicare il fatto che l'interfaccia è definita dagli http verb; a rigore però vanno comunque progettate le giuste risorse da esporre su cui effettuare operazioni CRUD, e quindi più che essere senza contratto, è il contratto che ha una forma differente.

Nel modo REST, il principio secondo cui l'interfaccia di servizio (in questo caso l'API) deve essere il primo artefatto di progettazione, viene recentemente indicato come API-first[17491] ed è largamente adottato da molte organizzazioni private, ed anche framework di interoperabilità nazionali come quello inglese[17592].

Nel caso invece di nuovi sistemi, la progettazione dell'interfaccia può essere effettuata sia in un'ottica contract-first che contract-less. In un'ottica contract-first, la specifica dell'interfaccia viene effettuata a tavolino a partire dalle macro-operazioni che si vogliono offerte dal sistema finale. Nel caso di accesso basato su risorsa (in ottica ROA), essendo in realtà le operazioni da effettuare già predefinite (operazioni CRUD), il tipo di progettazione è contract-less. Vanno però definite le risorse che il sistema deve esporre, quindi una qualche forma di progettazione preventiva all'implementazione è comunque prevista (cioè, la specifica delle risorse).

Nel progetto di interfacce di servizio SOAP occorre definire, oltre alle macro-operazioni, anche le strutture XML per la rappresentazione dei dati. Le operazioni possono essere raggruppate in base a caratteristiche quali: area funzionale (o area di business), requisiti di sicurezza (ad es. meccanismi di autenticazione ed autorizzazione), oppure in base a fattori organizzativi quali la frequenza dei cambiamenti o pattern di gestione delle versioni. Il

[90] Cf. Cesare Pautasso, Olaf Zimmermann, Frank Leymann: Restful web services vs. «big» web services: making the right architectural decision. WWW 2008: 805-814.

[174] https://www.programmableweb.com/api-university/understanding-api-first-design

[91] Cf. https://www.programmableweb.com/api-university/understanding-api-first-design In termini colloquiali, il principio può essere parafrasato in questi termini:

- L'API è la prima interfaccia dell'applicazione
- L'API viene prima dell'implementazione
- L'API deve essere descritta (ed addirittura essere auto-descrittiva, se possibile e fattibile)

[175] https://www.programmableweb.com/news/why-uks-government-data-service-takes-api-first-approach-to-datagovuk/elsewhere-web/2016/09/02

[92] Cf. https://www.programmableweb.com/news/why-uks-government-data-service-takes-api-first-approach-to-datagovuk/elsewhere-web/2016/09/02

principio alla base di questo raggruppamento è quello di impattare il minor numero di fruitori quando avviene un cambiamento.

Nel progetto di interfacce di servizio REST invece occorre:

- Identificare le risorse che l'interfaccia di servizio manipolerà. Queste risorse sono solitamente i concetti base che stanno dietro ad un processo (ad es., un ordine di acquisto).
- Progettare gli URI seguendo i principi introdotti nella sezione relativa alla tecnologia REST.
- Scegliere il tipo di operazione disponibile per ognuna delle URI.
- Scegliere i collegamenti tra risorse da fornire nelle risposte. In quest'ottica l'approccio HATEOAS può risultare utile.
- Progettare le strutture JSON per la rappresentazione dei dati.

Il ModI, come discusso nella Sezione 1, prevede che la progettazione parta dalla definizione delle interfacce di servizio, indipendentemente dalla tecnologia di realizzazione sia SOAP che REST, anche se con accorgimenti tecnici differenti nella sua realizzazione.

3.7 Altri approcci e tecnologie di integrazione

Nelle precedenti sezioni, sono state introdotte le principali tecnologie di integrazione. Accanto a queste, stanno emergendo altre modalità di integrazione che potrebbero essere proposte in futuro in affiancamento in casi d'uso molto specifici.

3.7.1 Datastore distribuiti

L'applicazione di tecnologie per datastore distribuiti è strettamente connessa, in ambito integrazione di sistemi, al mantenimento di database multi-tenant in cui, ad esempio, si richiede data locality per basi di dati di grandi dimensioni. In questo contesto, vanno considerati principalmente i file system ed i database distribuiti.

I file system distribuiti offrono interfacce basate su API per la memorizzazione di file e di oggetti e sono oggigiorno disponibili sia in soluzioni cloud pubbliche sia private. La sicurezza di queste soluzioni è soggetta agli stessi vincoli visti per l'utilizzo di interfacce di servizio nelle sezioni precedenti.

Tra i database distribuiti, grande interesse è stato suscitato da quelli basati su blockchain[176][93]. L'obiettivo di una blockchain è il mantenimento di un *libro mastro distribuito* (distributed ledger) mediante una rete peer-to-peer di nodi[94]. L'obiettivo è quello di avere un datastore capace di certificare transazioni e vincoli contrattuali, in cui il meccanismo di distribuzione certifica la validità degli stessi. In particolare, è possibile appurare la validità di *smart contract* (contratti intelligenti), certificando le precondizioni degli stessi. Il termine contratto spazia dal semplice scambio di denaro, ad es., la piattaforma Bitcoin in cui la precondizione all'invio di denaro è il possesso del denaro stesso, a contratti complessi dove le precondizioni possono assumere una qualunque forma. L'integrità dei dati memorizzati è certificata da meccanismi basati su chiave pubblica. La maggior parte dei protocolli disponibili per la realizzazione di blockchain sono basati su scambio di messaggi su TCP/TLS o HTTPS.

In Estonia, il modello X Road[177][95] (equivalente al ModI) ha promosso l'utilizzo di un ledger distribuito nell'ambito della Pubblica Amministrazione, anche se più a scopo di marketing[178][96] che per l'utilizzo degli aspetti precipui di una blockchain. Quello che è interessante è l'idea di un tracciamento distribuito delle decisioni prese da una Pubblica Amministrazione.

[176] https://it.wikipedia.org/wiki/Blockchain
[93] Cf. https://it.wikipedia.org/wiki/Blockchain
[94] Una rete di calcolatori si definisce peer-to-peer, quando le macchine componenti (i nodi) non sono organizzati gerarchicamente ma svolgono delle funzionalità paritarie.
[177] https://e-estonia.com/solutions/security-and-safety/ksi-blockchain
[95] Cf. https://e-estonia.com/solutions/security-and-safety/ksi-blockchain
[178] https://techcrunch.com/2018/04/19/do-you-need-a-blockchain/
[96] Cf. https://techcrunch.com/2018/04/19/do-you-need-a-blockchain/

La tecnologia blockchain non è esente da rischi in quanto diversi tipi di attacco sono stati formulati che permettono la modifica dei contenuti e la creazione di ramificazioni della catena di transazioni alla base del libro mastro[17997].

In conclusione, sebbene si tratti di una tecnologia che sta suscitando interesse, attualmente blockchain non sono considerate abbastanza mature per l'utilizzo nella Pubblica Amministrazione in settori strategici e il ModI ne sconsiglia al momento l'utilizzo. Inoltre deve ancora essere definito il modo di integrare ed interoperare tra PA utilizzando smart contract come interfacce di servizio, e le tipologie di transazioni che effettivamente hanno bisogno di requisiti tali per cui la blockchain sia la giusta soluzione.

L'utilizzo di datastore distribuiti potrebbe in futuro affiancare l'integrazione basata su altre tecnologie più consolidate. Le future linee guida dovranno tenere in considerazione per queste tecnologie:

- requisiti di latenza e consistenza (ad es., eventual consistency, autoritatività);
- le modalità di logging e auditing associate alla trasmissione dei dati;
- le modalità operative di manutenzione;
- la standardizzazione delle interfacce di accesso.

3.7.2 Esposizione di open data

Una modalità di integrazione, importante specialmente negli scenari A2B e A2C, è quella basata sull'esposizione da parte delle PA di *open data*. Gli open data devono essere fruibili, ed essere inseriti ove possibile nel contesto dei Base Register definiti nell"EIF[18098], standardizzando gli schemi e le modalità di fruizione.

Vista la progressiva crescita dei dataset, gli open data dovrebbero essere erogati in modo da ridurre gli impatti infrastrutturali sull'erogatore.

Come indicato nelle linee guida nazionali per la valorizzazione del patrimonio informativo pubblico[18199] pubblicate da AgID nel 2014, l'obiettivo è quello di mettere a disposizione i dati aperti in formato Linked Open Data - LOD ai fini dell'integrazione, il che prevede l'esposizione di dati in formato W3C RDF e SPARQL (secondo il cosiddetto modello del *Semantic Web*). A tal fine gli SPARQL endpoint costituiscono le interfacce di servizio. Le query in formato SPARQL vengono inviate su endpoint HTTP. Un altro approccio possibile, sempre nel rispetto dei dizionari comuni, è quello di utilizzare un approccio ROA basato su interfacce REST[100].

Un'interessante evoluzione dell'approccio REST (di cui eredita molti dei vantaggi, quali ad esempio la leggerezza e l'utilizzo dei verbi HTTP) che può risultare utile nell'esposizione di open data è quello basato su GraphQL[182101]. In particolare, mentre per l'estrazione di dati complessi l'approccio basato su interfacce di servizio REST richiede diverse chiamate, GraphQL introduce un linguaggio che permette l'esecuzione di interrogazioni complesse sulle risorse.

In tutti i casi presentati, restano valide le indicazioni contenute nelle sezioni precedenti circa la sicurezza nell'esposizione delle interfacce di servizio.

[179] https://www.multichain.com/blog/2017/05/blockchain-immutability-myth/
[97] Cf. https://www.multichain.com/blog/2017/05/blockchain-immutability-myth/
[180] https://joinup.ec.europa.eu/asset/eia/description
[98] Cf. https://joinup.ec.europa.eu/asset/eia/description
[181] http://lg-patrimonio-pubblico.readthedocs.io/it/latest/index.html
[99] Cf. http://lg-patrimonio-pubblico.readthedocs.io/it/latest/index.html
[100] Cf. Massimo Mecella, Francesco Leotta (2017): Migliorare l'accesso agli open data pubblici: tecnologie e approcci, https://www.agendadigitale.eu/cittadinanza-digitale/pa-tecnologie-e-approcci-per-migliorare-laccesso-ai-dati-aperti/
[182] https://graphql.org/
[101] Cf. https://graphql.org/

CAPITOLO 4

Profili e Raccomandazioni

La terza sezione del Modello di Interoperabilità, così come introdotto nella sezione relativa alla visione generale, fornisce indicazioni concrete, a livello tecnico, su differenti modalità operative per realizzare l'interoperabilità tra sistemi informativi di differenti pubbliche amministrazioni, tenendo conto delle possibili tecnologie ed approcci disponibili. Data la veloce evoluzione tecnologica, questo documento verrà continuamente aggiornato ed approfondito. Questa versione si focalizza sulle modalità bloccanti e non bloccanti e sulle tecnologie SOAP e REST.

La sezione è organizzata nel seguente modo:

- *3.1 - Paradigmi per la progettazione delle interfacce di servizio* presenta una discussione sulla progettazione di interfacce di servizio in modalità cosiddetta *RPC-like* ed in modalità cosiddetta *resource-oriented*, e su altri aspetti legati alla relazione tra sistemi informativi delle pubbliche amministrazioni e sistemi esterni, incluse alcune considerazioni su come le modalità di dispiegamento delle interfacce di servizio siano correlate alla tipologia di client e di dispositivo che le fruisce.
- *3.2 - Profili di interazione* descrive i profili di interoperabilità proposti, adottando uno stile uniforme di presentazione a scheda, in modo da rendere agevole e di immediata fruibilità la consultazione.
- *3.3 - Concetti di base* presenta ulteriori concetti architetturali che vengono richiamati nel documento, insieme alla bibliografia di riferimento, sia per motivi di autocontenimento che di disambiguazione.
- *3.4 - Riferimenti Bibliografici*

NOTA BENE

Le parole chiave «DEVE», «DEVONO», «NON DEVE», «NON DEVONO», «E" RICHIESTO», «DOVREBBE», «NON DOVREBBE», «RACCOMANDATO», «NON RACCOMANDATO» «PUO"» e «OPZIONALE» nel testo del documento debbono essere interpretate come descritto nel seguito, in conformità alle corrispondenti traduzioni contenute nel documento IETF **RFC 2119**[183].

[183] https://tools.ietf.org/html/rfc2119.html

4.1 Paradigmi per la progettazione delle interfacce di servizio

Una trattazione completa dei paradigmi per la progettazione e realizzazione delle interfacce di servizio esula dagli scopi del presente documento. Il lettore interessato ad approfondire gli aspetti metodologici può riferirsi a[1],[2] . La breve discussione che viene presentata in questa sezione, condotta anche attraverso esempi, vuole ricordare al lettore la problematica dell'*impedance mismatch* tra differenti tecnologie per la realizzazione di interfacce di servizio ed il modello architetturale sottostante (RPC-like vs. resource-oriented).

Le interfacce di servizio, cf. il secondo documento delle linee guida, possono essere progettate secondo due paradigmi:

1. **RPC-like**. In questo paradigma, un'interfaccia di servizio espone una serie di operazioni (metodi) che permettono l'invocazione delle operazioni offerte dall'interfaccia. Il significato dell'operazione è informalmente espresso dal nome dell'operazione, dal numero e tipo dei suoi parametri, dal tipo del valore di ritorno (il tutto viene detto tecnicamente *segnatura*). Approcci formali prevedono che tale significato venga eventualmente anche descritto in opportuni documenti di accompagnamento e/o attraverso specifiche formali dell'interfaccia di servizio. Ogni operazione può quindi rappresentare sia semplici operazioni di computazione, che operazioni *long-running*, che operazioni di accesso a dati, ecc. Il paradigma è detto RPC-like in quanto le interfacce di servizio ricordano le librerie di chiamata a procedura (RPC sta per *remote procedure call*) e quindi l'interfaccia, metaforicamente, è di fatto una libreria di funzioni.
2. **Resource-oriented**. In questo paradigma, l'interfaccia di servizio offre operazioni CRUD di accesso a risorse. CRUD - Create, Read, Update e Delete sono le 4 operazioni fondamentali con cui manipolare risorse. Una risorsa è un qualsiasi oggetto informativo che possiede uno stato. In questo paradigma, l'interfaccia di servizio, metaforicamente, è un accesso diretto ad una base informativa, e le uniche operazioni possibili sono appunto le modifiche a tali risorse.

Parallelamente, esistono delle tecnologie con cui poter naturalmente realizzare interfacce di servizio, che sono *(i)* SOAP ed il cosiddetto stack WS-*[1], e *(ii)* lo stile architetturale REST basato su HTTP. Entrambe sono indicate come modi per realizzare i cosiddetti *Web service* (servizi richiamabili sul Web).

- Un Web service basato su SOAP espone un insieme di metodi richiamabili da remoto da parte di un client. SOAP definisce una struttura dati per lo scambio di messaggi tra applicazioni, codificata in XML; di fatto SOAP utilizza HTTP come protocollo di trasporto, ma non è limitato né vincolato ad esso, dal momento che può benissimo usare altri protocolli di trasporto.
- Un Web service RESTful adotta il modello basato su risorse secondo le seguenti caratteristiche:
  - individuazione delle risorse mediante il formalismo delle URI;
  - operazioni sulle risorse effettuate sulla rappresentazione del loro stato;
  - CRUD sulle risorse mediante HTTP utilizzando i metodi nella semantica prevista dal protocollo stesso.

La prima differenza tra i due tipi di Web service è la visione del Web come piattaforma di elaborazione che viene implicitamente proposta: REST propone una visione del Web incentrata sul concetto di risorsa mentre i SOAP Web service mettono in risalto il concetto di servizio.

L'approccio REST evidenzia le caratteristiche del Web come piattaforma leggera per l'elaborazione distribuita. Non è, in prima istanza, necessario aggiungere nulla a quanto è già esistente sul Web per consentire ad applicazioni remote di interagire.

Un Web service SOAP è adatto a realizzare sia interfacce di servizio RPC-like che resource-oriented, mentre un Web service REST è finalizzato a realizzare interfacce di servizio resource oriented. Questo è dovuto al fatto che SOAP è di fatto la specifica di un middleware, e quindi neutro rispetto al paradigma, mentre REST,

[1] SOAP - Simple Object Access Protocol è il protocollo originariamente proposto, e standardizzato dal W3C, per lo sviluppo e dispiegamento di Web service. Al di sopra di esso, sono stati nel tempo proposti vari standard per Web service, ad es., WS-Addressing, WS-Discovery, WS-Federation, WS-Policy, WS-Security, and WS-Trust solo per nominarne alcuni, che oramai vengono comunemente indicati con l'acronimo WS-*.

[2] Originariamente Swagger (della società SmartBear Software) era un insieme di tool sia per la descrizione delle interfacce che per il loro sviluppo. Nel 2015 un gruppo di aziende, sotto la sponsorizzazione della Linux Foundation, ha dato vita all'iniziativa OpenAPI, a cui SmartBear ha donato il formato di specifica che è stato rinominato da Swagger Specification in OpenAPI Specification. Gli strumenti Swagger, che sono ancora supportati da SmartBear Software, sono tra gli strumenti più popolari per implementare la specifica OpenAPI e continueranno a mantenere il nome Swagger. Esistono molti altri strumenti open source e proprietari, non correlati a Swagger, che supportano la specifica OpenAPI.

in quanto stile architetturale, è maggiormente caratterizzato. Gli esempi successivi evidenziano l'*impedance mismatch* (differenza tra i modelli realizzativi e concettuali, cf. Capitolo 3) che si origina nelle varie casistiche partendo da una interfaccia di servizio secondo ciascuno dei due paradigmi (RPC-like e resource oriented) e cercando di realizzarla utilizzando le due diverse tecnologie SOAP e REST.

Esempio 1. Si immagini di voler realizzare un'interfaccia di servizio resource-oriented che permette di accedere ad informazioni su ordini e voci (oggetti acquistati, *item*) inseriti negli ordini. Utilizzando SOAP/WSDL, si realizza un'interfaccia di servizio che offre operazioni quali submitOrder() e getOrderDetails(), come mostrato in Figura 3.1. L'interfaccia di servizio utilizza Order come oggetto serializzabile (in XML) per i dettagli dell'ordine da ritornare come tipo di ritorno nelle varie operazioni ovvero come parametro per la sua creazione.

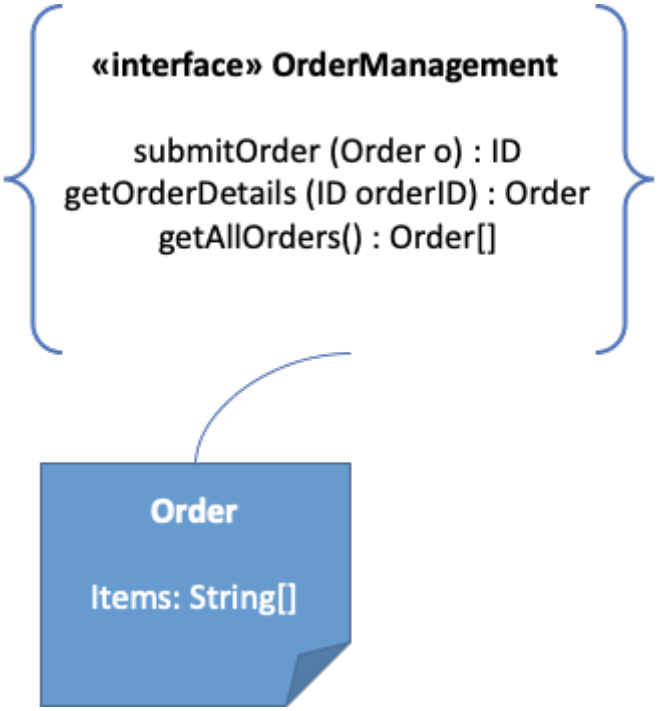


Fig. 4.1: Interfaccia di servizio usando SOAP/WSDL.

Per semplicità nell'esempio si suppone che il singolo item nell'Order sia rappresentabile con una semplice stringa (che codifica il suo codice, ad esempio) e sempre in quantità singola (per cui un Order non fa altro che aggregare differenti Items).

Utilizzando REST, l'interfaccia di servizio corrisponde a due risorse, Order e Item ed i verbi HTTP (ovvero GET, PUT, POST e DELETE) si mappano esattamente sulle operazioni CRUD, come mostrato in Figura 3.2.

Esempio 2. Si immagini di voler realizzare un'interfaccia di servizio che permette di avviare un task di lunga durata, ad esempio una complessa procedura computazionale. Utilizzando SOAP/WSDL, si realizza un'interfaccia di servizio TaskManagement con operazioni startTask() e getResult(), che ritorna un valore che codifica che il task è ancora in esecuzione (fintanto che non è completato), oppure il risultato finale una volta che la procedure/task sia stata completata ed il risultato è stato prodotto. Figura 3.3 schematizza l'interfaccia di servizio.

Per realizzare una interfaccia di servizio che fornisce la stessa logica in tecnologia REST occorre ridefinire l'interfaccia utilizzando il paradigma a risorse. Si introducono le risorse Task e Result utilizzando le quali è possibile implementare le stesse funzionalità. La POST serve quindi a far partire il task.

Listato 4.1: Request

```
POST /tasks HTTP/1.1
Content-Type: application/json

{ .. request payload }
```

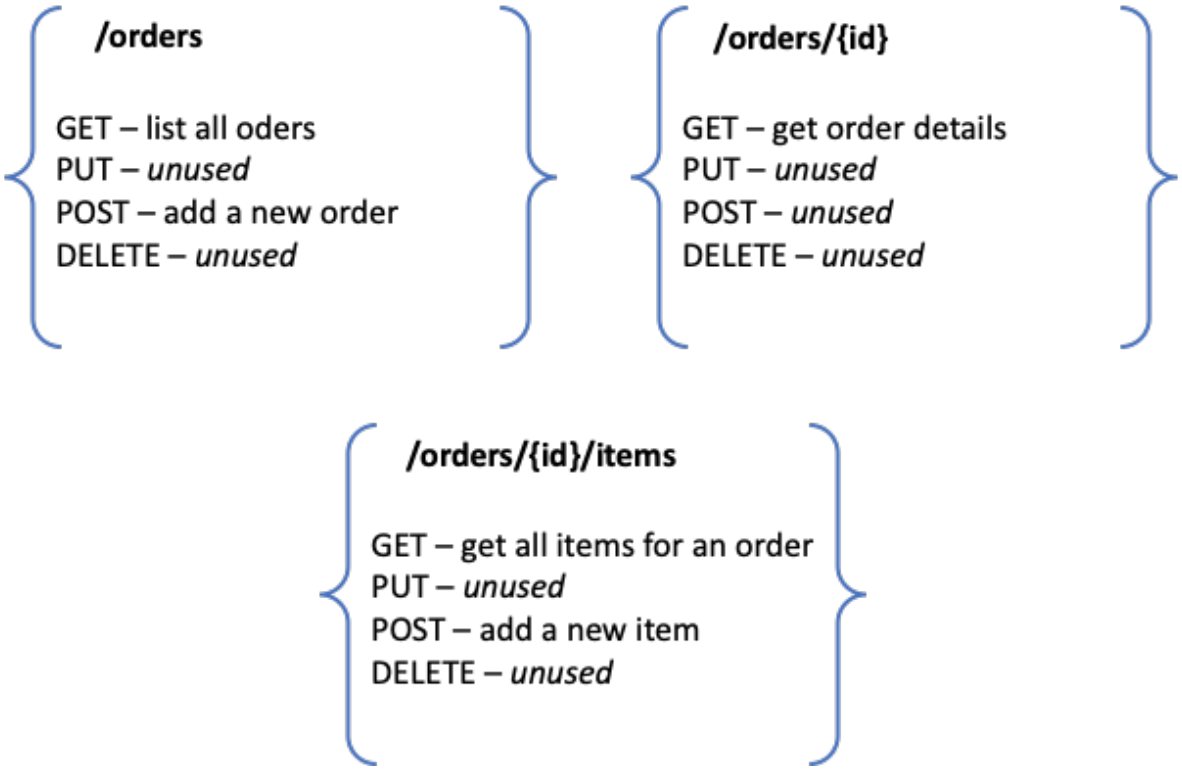


Fig. 4.2: Interfaccia di servizio usando REST.

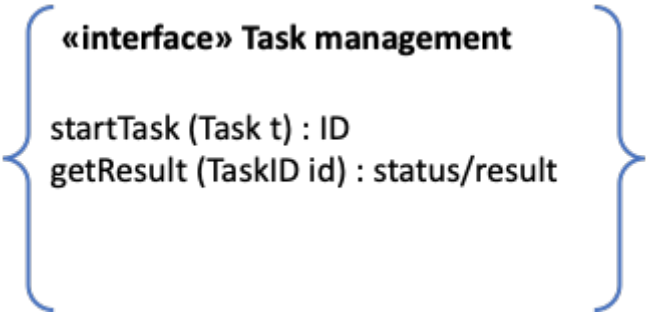


Fig. 4.3: Interfacce di servizio usando SOAP/WSDL

Listato 4.2: Response

```
HTTP/1.1 202 Accepted
Content-Type: application/json
Location: /tasks/20181231

{
   "task": {
   "status": "accepted",
   "message": "Your task has been queued for processing",
   "ping_time": "2018-12-31T19:43:37+0100"
   }
}
```

Il **HTTP status 202 Accepted**[184] indica che l'interfaccia di servizio ha verificato l'input della richiesta e lo ha accettato, ma non è fornita una risposta immediata.

Il client deve seguire il collegamento fornito nell'intestazione Location per informarsi (con GET) sullo stato della richiesta in sospeso.

Listato 4.3: Request

```
GET /task/20181231 HTTP/1.1
```

Listato 4.4: Response

```
HTTP/1.1 200 OK

{
   "task": {
   "status": "processing",
   "message": "Your task is being processed",
   "ping_time": "2018-12-31T19:52:45+0100"
   }
}
```

I client possono inviare richieste GET in qualsiasi momento per tenere traccia dei progressi. Oltre allo stato, la risposta contiene anche un suggerimento (nell'elemento ping-time) su quando deve essere eseguita la successiva richiesta di polling per ridurre il traffico di rete e il carico di servizio a causa di un polling eccessivo.

Una volta completato il lavoro, la risposta alla richiesta di polling reindirizza il client a un'altra risorsa da cui è possibile recuperare il risultato finale.

Listato 4.5: Request

```
GET /task/20181231 HTTP/1.1
```

Listato 4.6: Response

```
HTTP/1.1 303 See Other
Location: /result/20181232

{
   "task": {
   "status": "done",
   "message": "Your task is completed",
   }
}
```

[184] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/202

Il client può quindi seguire il collegamento trovato nell'intestazione Location per recuperare (con GET) il risultato della computazione completata. Il collegamento potrebbe anche essere condiviso tra diversi client interessati a leggere l'output della richiesta POST originale.

Listato 4.7: Request

```
GET /result/20181232 HTTP/1.1
```

Listato 4.8: Response

```
HTTP/1.1 200 OK
Content-Type: application/json

[Output data payload]
```

Questo secondo esempio mostra invece che se l'interfaccia di servizio è orientata alle funzionalità (come appunto nell'esempio, in cui di fatto si vogliono eseguire operazioni remote) allora l'*impedance mismatch* con SOAP è minimo, e nel caso invece si voglia utilizzare REST, occorre convertire l'interfaccia originariamente RPC-like in una resource oriented.

Si noti infine che nell'applicazione pratica di REST si assiste al suo uso in modalità non del tutto canoniche. Ogni deviazione rispetto alle caratteristiche previste da REST porta alla realizzazione di architetture ibride tra il paradigma RESTful Web service e quello dei Web service RPC-like. In merito ai modelli ibridi che si possono presentare, esiste una classificazione, il cosiddetto Richardson Maturity Model[3] che prevede quattro livelli, da 0 a 3, in accordo al grado di aderenza ai dettami REST. In particolare, si possono presentare i casi seguenti:

- Livello 0, per servizi che semplicemente usano HTTP come protocollo di trasporto applicativo (tunnel HTTP). In questo caso il sistema non ha niente del modello REST.
- Livello 1, per i servizi che operano sulle risorse definite secondo la sintassi e la semantica previste per le URI, sulle quali si opera invocando delle operazioni (metodi) che agiscono su di esse.
- Livello 2, per i servizi che operano su risorse definite secondo la sintassi e la semantica previste per le URI, sulle quali si opera sulla rappresentazione del loro stato per mezzo del protocollo HTTP usando la semantica dei metodi (verbi) come previsti dal protocollo.
- Livello 3, come per il livello 2, con in aggiunta la possibile presenza di controlli ipermediali nella rappresentazione delle risorse.

Si anticipa che nel Capitolo 2 dedicato ai profili, si adotteranno, nel caso dei profili non bloccanti realizzati in tecnologia REST, delle interfacce di servizio classificabili di livello 1 del Richardson Maturity Model.

4.1.1 Perimetro delle interfacce di servizio

Un aspetto che si vuole qui richiamare è la relazione tra l'interno e l'esterno del sistema informativo di una pubblica amministrazione, e come questo confine abbia impatti sulle interfacce di servizio in termini di funzionalità e sicurezza. Nel precedente modello di interoperabilità (il cosiddetto SPCoop del 2005) era stato definito il concetto di *dominio* di un'amministrazione, o *dominio di cooperazione* tra più amministrazioni, ad indicare l'insieme delle risorse - tra cui procedure, dati e servizi - e delle politiche di una determinata amministrazione o gruppo di amministrazioni, e rappresentava il confine di responsabilità, in particolar modo per quanto riguardava le politiche relative al sistema informativo della stessa (o gruppo di amministrazioni). Uno specifico elemento architetturale, la Porta di Dominio, istanziava fisicamente tale confine.

Nel nuovo framework di interoperabilità, l'istanziazione della Porta di Dominio come punto unico di interfaccia viene meno, tuttavia concettualmente il confine del *dominio* dell'amministrazione continua ad esistere ed è importante considerarlo nella progettazione delle interfacce di servizio, soprattutto relativamente agli aspetti di sicurezza. Le interfacce di servizio vengono offerte da qualsiasi server applicativo, senza essere vincolate ad essere raggiungibili attraverso un unico gateway.

Le figure **3.4** e **3.5** illustrano schematicamente la differenza tra i due framework.

[3] Cf. https://www.crummy.com/writing/speaking/2008-QCon/act3.html

Quindi ogni server applicativo offre interfacce di servizio, tuttavia è comunque significativo distinguere se l'interfaccia di servizio viene offerta per interoperare:

- all'interno del dominio (da parte di clienti applicativi offerti dalla stessa amministrazione, ad es., un'applicazione Web od una mobile)
- verso altre amministrazioni o altri soggetti con cui è stabilità una relazione di fiducia
- esternamente, da parte di moduli applicativi completamente esterni alle pubbliche amministrazioni, e per i quali non esiste a priori nessuna relazione, né organizzativa né di fiducia.

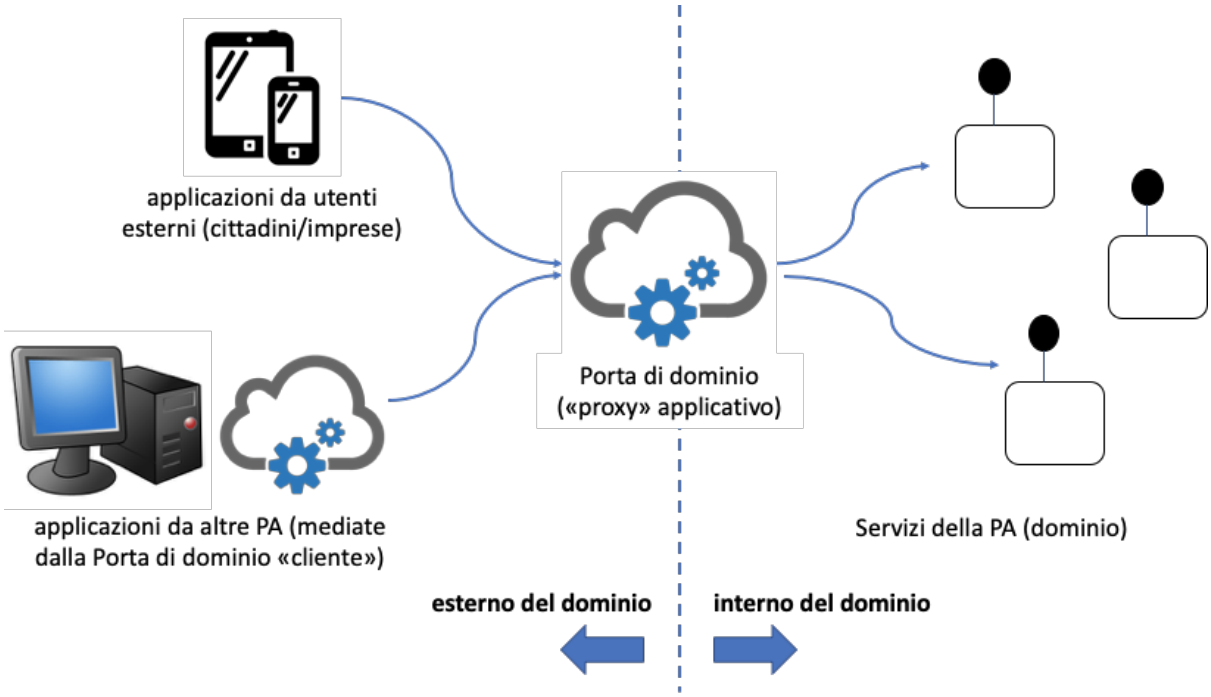


Fig. 4.4: Perimetro delle interfacce in SPCoop

4.2 Profili di Interazione

Il *profilo di interazione* definisce le modalità secondo le quali un fruitore ed un erogatore possono interagire, l'interazione avviene definendo le interfacce di servizio. I *profili di interazione* quali riferimenti concettuali coniugano, fatto salvo per l'accesso CRUD, message exchange pattern (MEP) per le tecnologie SOAP e REST proposte dal ModI. Di seguito l'attenzione è rivolta agli aspetti funzionali rimandando per gli aspetti di sicurezza all'apposito documento delle linee guida.

Per i concetti di bloccante e non bloccante si rimanda alla lettura di «Concetti di base».

Ogni interfaccia di servizio deve essere rappresentata mediante un IDL - Interface Description Language standard - che il ModI fissa:

- per le interfacce di servizio REST, OpenAPI 3.0 e successive
- per le interfacce di servizio SOAP, WSDL 1.1 e successive

L'endpoint deve indicare in modo esplicito la tecnologia utilizzata (REST o SOAP) e la versione (versioning su URL).

4.2.1 Modalità di descrizione dei profili

Nel seguito, per gli esempi proposti per i diversi profili si fa riferimento ad un'amministrazione denominata `amministrazioneesempio` che offre un'interfaccia di servizio secondo le due diverse tecnologie REST o

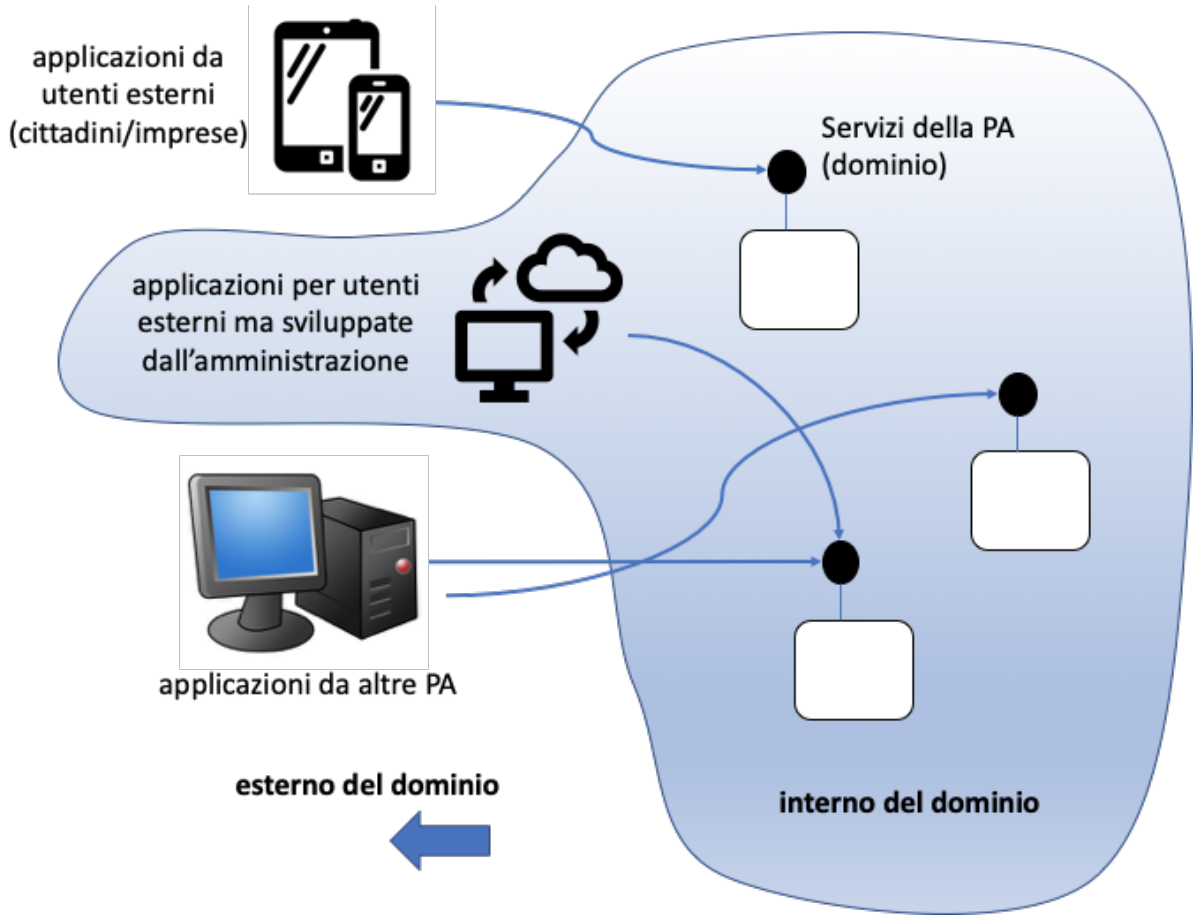


Fig. 4.5: Perimetro delle interfacce in ModI

SOAP. Inoltre, per quanto riguarda i profili relativi a chiamata a procedura remota (bloccante e non bloccante), si farà riferimento ad un metodo `M` che accetta come parametri:

- `a`, un oggetto contenente a sua volta un array `a1` di interi ed una stringa `a2`;
- `b`, una stringa.

e restituisce una stringa `c` come output.

Per ognuno dei profili verrà introdotto:

- lo scenario di applicazione (sottosezione Scenario), che indica in che casi vada utilizzato lo specifico profilo;
- la descrizione (sottosezione Descrizione) dello scambio di messaggi a livello applicativo, indipendentemente quindi dalla tecnologia utilizzata;
- una sezione che riguarda l'utilizzo della tecnologia REST (sottosezione REST) per lo specifico profilo, che include una sezione con le regole generali da seguire e l'implementazione dell'esempio proposto;
- una sezione che riguarda l'utilizzo della tecnologia SOAP (sottosezione SOAP) per lo specifico profilo, che include una sezione con le regole generali da seguire e l'implementazione dell'esempio proposto.

Le implementazioni degli esempi sono corredate dalla specifica dell'interfaccia e da uno scambio di messaggi esemplificativo.

4.2.2 Profilo bloccante RPC

Scenario

Lo sviluppo di una interfaccia bloccante di tipo RPC si richiede nei casi in cui:

- L'esecuzione del metodo `M` è poco onerosa computazionalmente e può essere portata immediatamente a termine dall'erogatore.
- Il contesto rende complessa l'implementazione delle modalità non bloccanti. Ad esempio, non è possibile per il fruitore esporre una propria interfaccia di servizio ed il fruitore non può farsi carico di mantenere il contesto necessario ad effettuare attesa attiva.

Descrizione

Fig. 4.6: Interazione bloccante RPC

In questo profilo si ha una risposta da parte dell'erogatore contestuale alla richiesta del fruitore. La figura mostra lo schema di questa interazione.

Interfaccia REST

Nel caso di implementazione tramite tecnologia REST, DEVONO essere seguite almeno le seguenti indicazioni:

- La specifica dell'interfaccia DEVE dichiarare tutti i codici di stato HTTP restituiti con relativo schema della risposta, oltre che ad eventuali header HTTP restituiti;
- La specifica dell'interfaccia DEVE dichiarare lo schema della richiesta insieme ad eventuali header HTTP richiesti;
- Al passo (1) il fruitore DEVE utilizzare come verbo HTTP per l'esecuzione della chiamata a procedura il verbo **HTTP method POST** [185] su un URL contenente gli ID interessati ed il nome del metodo;
- Al passo (2) il fruitore DEVE utilizzare **HTTP status 200 OK**[186] a meno che non si verifichino errori.

[185] https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST
[186] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/200

Regole di processamento

Al termine del processamento della richiesta, l'erogatore DEVE fare uso dei codici di stato HTTP rispettandone la semantica[1]. In particolare, al ricevimento della richiesta da parte del fruitore, l'erogatore:

- DEVE verificare la validità sintattica e semantica dei dati in ingresso;
- DEVE, in caso di dati errati, restituire **HTTP status 400 Bad Request**[187] fornendo nel body di risposta dettagli circa l'errore;
- DEVE, in caso di representation semanticamente non corretta, ritornare **HTTP status 422** [188];
- DEVE, se qualcuno degli ID nel path o nel body non esiste, restituire **HTTP status 404 Not Found**[189], indicando nel body di risposta quale degli ID è mancante;
- PUO", se ipotizza che la richiesta sia malevola, ritornare **HTTP status 400 Bad Request**[190] o **HTTP status 404 Not Found**[191]
- DEVE, in caso di errori non dipendenti dalla richiesta, restituire HTTP status 5XX rispettando la semantica degli stessi;
- DEVE, in caso di successo, restituire **HTTP status 200 OK**[192] inviando il risultato dell'operazione nel payload body.

NB: I messaggi di errore devono essere utili al client ma NON DEVONO rivelare dettagli tecnici e/o informazioni riservate.

Esempio

Specifica Servizio https://api.amministrazioneesempio.it/rest/nomeinterfacciaservizio/v1/RESTblocking.yaml

```
openapi: 3.0.1
info:
 title: RESTblocking
 version: "1.0"
 description: |-
   Questo file descrive semplicemente i metodi di un'API
   e non indica tutte le informazioni di metadatazione che
   normalmente vanno inserite.
 license:
   name: Apache 2.0 License
   url: http://www.apache.org/licenses/LICENSE-2.0.html
paths:
 /resources/{id_resource}/M:
   post:
     description: Esegue M
     operationId: M
     parameters:
     - name: id_resource
       in: path
       required: true
       schema:
         type: integer
         format: int32
     requestBody:
       content:
```

(continues on next page)

[1] http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml
[187] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/400
[188] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/422
[189] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/404
[190] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/400
[191] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/404
[192] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/200

(continua dalla pagina precedente)

```
        application/json:
          schema:
            $ref: '#/components/schemas/MType'
    responses:
      200:
        description: Esecuzione di M avvenuta con successo
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/MResponseType'
      400:
        description: Richiesta non valida
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ErrorMessage'
      404:
        description: Identificativo non trovato
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ErrorMessage'
      default:
        description: |-
          Errore inatteso. Non ritornare informazioni
          sulla logica interna e/o non pertinenti all'interfaccia.
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ErrorMessage'

components:
 schemas:
   MType:
     type: object
     properties:
       a:
         $ref: '#/components/schemas/AComplexType'
       b:
         type: string
   MResponseType:
     type: object
     properties:
       c:
         type: string
   AComplexType:
     type: object
     properties:
       a1s:
         type: array
         items:
           type: integer
           format: int32
       a2:
         type: string
   ErrorMessage:
      type: object
      properties:
        detail:
          description: |
            A human readable explanation specific to this occurrence of the
```

(continues on next page)

(continua dalla pagina precedente)

```
            problem.
          type: string
        instance:
          description: |
            An absolute URI that identifies the specific occurrence of the problem.
            It may or may not yield further information if dereferenced.
          format: uri
          type: string
        status:
          description: |
            The HTTP status code generated by the origin server for this occurrence
            of the problem.
          exclusiveMaximum: true
          format: int32
          maximum: 600
          minimum: 100
          type: integer
        title:
          description: |
            A short, summary of the problem type. Written in english and readable
            for engineers (usually not suited for non technical stakeholders and
            not localized); example: Service Unavailable
          type: string
        type:
          default: about:blank
          description: |
            An absolute URI that identifies the problem type.  When dereferenced,
            it SHOULD provide human-readable documentation for the problem type
            (e.g., using HTML).
          format: uri
          type: string
```

Di seguito un esempio di chiamata al metodo `M`.

Http Operation POST

Endpoint https://api.amministrazioneesempio.it/rest/nomeinterfacciaservizio/v1/resources/1234/M

1. Request Body

```
{
 "a": {
   "a1s": [1,2],
   "a2": "RGFuJ3MgVG9vbHMgYXJlIGNvb2wh"
   },
   "b": "Stringa di esempio"
}
```

2. Response Body (HTTP Status Code 200 OK)

```
{
  "c" : "risultato"
}
```

2. Response Body (HTTP Status Code 500 Internal Server Error)

```
{
    "type": "https://apidoc.example.com/probs/operation-too-long",
    "status": 500,
    "title": "L'operazione dura troppo.",
    "detail": "Il sistema non e' riuscito a completare in tempo l'operazione␣
↪prevista.",
}
```

2. Response Body (HTTP Status Code 400 Bad Request)

```
{
    "type": "https://apidoc.example.com/probs/invalid-a",
    "status": 400,
    "title": "L'attributo `b` ha un valore non valido.",
    "detail": "L'attributo `b` dev'essere una stringa di lunghezza inferiore a 32␣
↪caratteri.",
}
```

2. Response Body (HTTP Status Code 404 Not Found)

```
{
    "status": 404,
    "title": "Risorsa non trovata.",
}
```

Interfaccia SOAP

Se il profilo viene implementato con tecnologia SOAP, a differenza del caso REST, il metodo invocato non è specificato nell'endpoint chiamato, poichè viene identificato all'interno del body. Inoltre tutti gli ID coinvolti DEVONO essere riportati all'interno del body. DEVE essere rispettata le seguente regola:

- La specifica dell'interfaccia dell'erogatore DEVE dichiarare tutti i metodi esposti con relativi schemi dei messaggi di richiesta e di ritorno. Inoltre le interfacce devono specificare eventuali header SOAP richiesti;

Regole di processamento

Nel caso di errore il WS-I Basic Profile Version 2.0 richiede l'utilizzo del meccanismo della SOAP fault per descrivere i dettagli dell'errore. Al ricevimento della richiesta da parte del fruitore, l'erogatore:

- DEVE verificare la validità sintattica dei dati in ingresso. In caso di dati errati DEVE restituire **HTTP status 500 Internal Server Error**[193] fornendo dettagli circa l'errore utilizzando il meccanismo della SOAP fault;
- Se l'erogatore ipotizza che la richiesta sia malevola PUO" ritornare **HTTP status 400 Bad Request**[194] o **HTTP status 404 Not Found**[195]
- In caso di errori non dipendenti dal fruitore, DEVE restituire i codici HTTP 5XX rispettando la semantica degli stessi o restituire il codice **HTTP status 500 Internal Server Error**[196] indicando il motivo dell'errore nella SOAP fault;
- In caso di successo restituire **HTTP status 200 OK**[197], riempiendo il body di risposta con il risultato dell'operazione.

Esempio

Specifica Servizio https://api.amministrazioneesempio.it/soap/nomeinterfacciaservizio/v1?wsdl

```
<?xml version='1.0' encoding='UTF-8'?>
<wsdl:definitions
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns:tns="http://amministrazioneesempio.it/nomeinterfacciaservizio"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap12/"
```

(continues on next page)

[193] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/500
[194] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/400
[195] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/404
[196] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/500
[197] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/200

(continua dalla pagina precedente)

```
        name="SOAPBlockingImplService"
        targetNamespace="http://amministrazioneesempio.it/nomeinterfacciaservizio">
        <wsdl:types>
                <xs:schema
                        xmlns:xs="http://www.w3.org/2001/XMLSchema"
                        xmlns:tns="http://amministrazioneesempio.it/
↪nomeinterfacciaservizio"
                        attributeFormDefault="unqualified" elementFormDefault=
↪"unqualified"
                        targetNamespace="http://amministrazioneesempio.it/
↪nomeinterfacciaservizio">
                        <xs:element name="MRequest" type="tns:MRequest"/>
                        <xs:element name="MRequestResponse" type=
↪"tns:MRequestResponse"/>
                        <xs:complexType name="MRequest">
                                <xs:sequence>
                                        <xs:element minOccurs="0" name="M" type=
↪"tns:mType"/>
                                </xs:sequence>
                        </xs:complexType>
                        <xs:complexType name="mType">
                                <xs:sequence>
                                        <xs:element minOccurs="0" name="oId" type=
↪"xs:int"/>
                                        <xs:element minOccurs="0" name="a" type=
↪"tns:aComplexType"/>
                                        <xs:element minOccurs="0" name="b" type=
↪"xs:string"/>
                                </xs:sequence>
                        </xs:complexType>
                        <xs:complexType name="aComplexType">
                                <xs:sequence>
                                        <xs:element minOccurs="0" name="a1s" type=
↪"tns:a1ComplexType"/>
                                        <xs:element minOccurs="0" name="a2" type=
↪"xs:string"/>
                                </xs:sequence>
                        </xs:complexType>
                        <xs:complexType name="a1ComplexType">
                                <xs:sequence>
                                        <xs:element maxOccurs="unbounded"␣
↪minOccurs="0" name="a1" nillable="true" type="xs:string"/>
                                </xs:sequence>
                        </xs:complexType>
                        <xs:complexType name="MRequestResponse">
                                <xs:sequence>
                                        <xs:element minOccurs="0" name="return"␣
↪type="tns:mResponseType"/>
                                </xs:sequence>
                        </xs:complexType>
                        <xs:complexType name="mResponseType">
                                <xs:sequence>
                                        <xs:element minOccurs="0" name="c" type=
↪"xs:string"/>
                                </xs:sequence>
                        </xs:complexType>
                        <xs:complexType name="errorMessageFault">
                                <xs:sequence>
                                        <xs:element minOccurs="0" name=
↪"customFaultCode" type="xs:string"/>
                                </xs:sequence>
```

(continues on next page)

(continua dalla pagina precedente)

```
                        </xs:complexType>
                        <xs:element name="ErrorMessageFault" nillable="true" type=
↪"tns:errorMessageFault"/>
                </xs:schema>
        </wsdl:types>
        <wsdl:message name="MRequest">
                <wsdl:part element="tns:MRequest" name="parameters"/>
        </wsdl:message>
        <wsdl:message name="MRequestResponse">
                <wsdl:part element="tns:MRequestResponse" name="parameters"/>
        </wsdl:message>
        <wsdl:message name="ErrorMessageException">
                <wsdl:part element="tns:ErrorMessageFault" name=
↪"ErrorMessageException"/>
        </wsdl:message>
        <wsdl:portType name="SOAPBlockingImpl">
                <wsdl:operation name="MRequest">
                        <wsdl:input message="tns:MRequest" name="MRequest"/>
                        <wsdl:output message="tns:MRequestResponse" name=
↪"MRequestResponse"/>
                        <wsdl:fault message="tns:ErrorMessageException" name=
↪"ErrorMessageException"/>
                </wsdl:operation>
        </wsdl:portType>
        <wsdl:binding name="SOAPBlockingImplServiceSoapBinding" type=
↪"tns:SOAPBlockingImpl">
                <soap:binding style="document" transport="http://schemas.xmlsoap.
↪org/soap/http"/>
                <wsdl:operation name="MRequest">
                        <soap:operation soapAction="" style="document"/>
                        <wsdl:input name="MRequest">
                                <soap:body use="literal"/>
                        </wsdl:input>
                        <wsdl:output name="MRequestResponse">
                                <soap:body use="literal"/>
                        </wsdl:output>
                        <wsdl:fault name="ErrorMessageException">
                                <soap:fault name="ErrorMessageException" use=
↪"literal"/>
                        </wsdl:fault>
                </wsdl:operation>
        </wsdl:binding>
        <wsdl:service name="SOAPBlockingImplService">
                <wsdl:port binding="tns:SOAPBlockingImplServiceSoapBinding" name=
↪"SOAPBlockingImplPort">
                        <soap:address location="https://api.amministrazioneesempio.
↪it/soap/nomeinterfacciaservizio/v1"/>
                </wsdl:port>
        </wsdl:service>
</wsdl:definitions>
```

A seguire un esempio di chiamata al metodo `M`.

Endpoint https://api.amministrazioneesempio.it/soap/nomeinterfacciaservizio/v1

Method M

1. Request Body

```
<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:m="http://amministrazioneesempio.it/nomeinterfacciaservizio" >
```

(continues on next page)

(continua dalla pagina precedente)

```
    <soap:Header>
        <!--Autenticazione-->
    </soap:Header>
    <soap:Body>
    <m:MRequest>
        <M>
            <oId>1234</oId>
            <a>
                <a1s><a1>1</a1>...<a1>2</a1></a1s>
                <a2>RGFuJ3MgVG9vbHMgYXJlIGNvb2wh</a2>
            </a>
            <b>Stringa di esempio</b>
        </M>
    </m:MRequest>
    </soap:Body>
</soap:Envelope>
```

2. Response Body (HTTP status code 200 OK)

```
<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:m="http://amministrazioneesempio.it/nomeinterfacciaservizio" >

    <soap:Body>
    <m:MRequestResponse>
      <return>
        <m:c>OK</m:c>
      </return>
    </m:MRequestResponse>
</soap:Body>

</soap:Envelope>
```

2. Response Body (HTTP status code 500 Internal Server Error)

```
<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:m="http://amministrazioneesempio.it/nomeinterfacciaservizio" >
    <soap:Body>
      <soap:Fault>
         <soap:Code>
            <soap:Value>soap:Receiver</soap:Value>
         </soap:Code>
         <soap:Reason>
            <soap:Text xml:lang="en">Error</soap:Text>
         </soap:Reason>
         <soap:Detail>
            <m:ErrorMessageFault>
               <customFaultCode>1234</customFaultCode>
            </m:ErrorMessageFault>
         </soap:Detail>
      </soap:Fault>
    </soap:Body>
</soap:Envelope>
```

4.2.3 Profili non bloccanti

Nel seguito del capitolo verranno mostrati i profili di interazione non bloccante, in cui un fruitore sottomette una richiesta e questa viene solo presa in carico immediatamente, mentre il suo soddisfacimento può avvenire in maniera differita.

Gli approcci non bloccanti vengono utilizzati nei casi in cui i tempi per l'erogazione di una risposta da parte del fruitore sono lunghi perché

- la richiesta è onerosa in termini temporali;
- il fruitore non può farsi immediatamente carico dell'erogazione del servizio.

Al fine di collegare le richieste con le risposte si farà uso, sia nelle implementazioni SOAP che in quelle REST, di meta-informazioni specifiche (quali il correlation ID ed l'endpoint per le callback). Queste sono estranee solitamente alla business logic del servizio, ma è necessario definirle a livello di interfaccia di servizio ai fini dell'interoperabilità. A tal fine verranno definiti header(HTTP nel caso REST ed envelope nel caso SOAP) utili a contenere queste informazioni.

In alcuni casi, una interfaccia di servizio viene creata al fine di automatizzare o semplificare un servizio già offerto dalla pubblica amministrazione. In una moltitudine di casi questi servizi sono asincroni (non bloccanti) per natura, e consistono di richieste a cui vengono allegati degli identificativi (es. numeri di protocollo) che accompagnano la richiesta. In questi casi, il correlation ID può essere sostituito da questi identificativi già previsti dal servizio.

Profili non bloccante RPC PUSH (basato su callback)

Scenario

Questo caso particolare, denominato RPC PUSH, è utilizzabile nel caso in cui il fruitore abbia a sua volta la possibilità di esporre una interfaccia di servizio per la ricezione delle risposte.

Descrizione

Fig. 4.7: Interazione non bloccante tramite callback

In questo scenario (vedi figura), la richiesta del fruitore contiene un riferimento al servizio da chiamare al momento della risposta. Si suppone infatti che i fruitori espongano a loro volta delle interfacce con segnatura standard. Al momento del ricevimento della richiesta (passo (1)), l'erogatore risponde (passo (2)) riconoscendo l'avvenuta ricezione (acknowledgement o in breve ack) ed indica un correlation ID (ID di correlazione), generato lato erogatore, che permetta al fruitore, una volta ricevuta la risposta, di accoppiarla alla richiesta originale. Quando il processamento è terminato infatti, l'erogatore (passo (3)) chiama il fruitore (invertendo quindi i ruoli chiamato/chiamante) riportando l'esito ed indicando il correlation ID. Il fruitore riconosce (sempre mediante un messaggio di ack) la ricezione della risposta (passo (4)).

Interfaccia REST

Nel caso in cui il profilo venga implementato con tecnologia REST, DEVONO essere rispettate le seguenti indicazioni:

- Le specifiche delle interfacce del fruitore e dell'erogatore DEVONO dichiarare tutti i codici di stato HTTP restituiti con relativo schema della risposta, oltre che ad eventuali header HTTP restituiti;
- Le specifiche delle interfacce del fruitore e dell'erogatore DEVONO dichiarare gli schemi delle richieste insieme ad eventuali header HTTP richiesti;
- La specifica dell'interfaccia dell'erogatore deve dichiarare tramite il formalismo specifico[1] il formato delle callback; questa specifica deve essere rispettata dall'interfaccia esposta dal fruitore, e quindi nella rispettiva specifica;
- Al passo (1), il fruitore DEVE indicare l'endpoint della callback utilizzando l'header HTTP custom `X-ReplyTo` ed usando **HTTP method POST** [198];

[1] Cf. https://swagger.io/docs/specification/callbacks/

[198] https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST

- Al passo (2), l'erogatore DEVE fornire insieme all'acknowledgement della richiesta nel body, il correlation ID utilizzando l'header HTTP custom `X-Correlation-ID`; Il codice HTTP di stato DEVE essere **HTTP status 202 Accepted**[199] a meno che non si verifichino errori;
- Al passo (3), l'erogatore DEVE riutilizzare lo stesso correlation ID fornito al passo (2) sempre utilizzando l'header HTTP custom `X-Correlation-ID`; Il verbo HTTP utilizzato deve essere POST;
- Al passo (4), il fruitore DEVE riconoscere tramite un messaggio di acknowledgement il ricevimento della risposta; Il codice HTTP di stato DEVE essere **HTTP status 200 OK**[200] a meno che non si verifichino errori.

Regole di processamento

Al termine del processamento delle richieste, l'erogatore ed il fruitore DEVONO fare uso dei codici di stato HTTP rispettandone la semantica[2]. Friutore ed erogatore:

- DEVONO verificare la validità sintattica dei dati in ingresso. In caso di dati errati DEVONO restituire HTTP status 500 Internal Server Error fornendo dettagli circa l'errore utilizzando il meccanismo della SOAP fault;
- In caso di dati errati DEVONO restituire **HTTP status 400 Bad Request**[201] fornendo nel body di risposta dettagli circa l'errore;
- In caso di representation semanticamente non corretta DEVONO ritornare **HTTP status 422** [202];
- Se qualcuno degli ID nel path o nel body non esiste, DEVONO restituire **HTTP status 404 Not Found**[203], indicando nel body di risposta quale degli ID è mancante;
- Se si ipotizza che la richiesta sia malevola, PUO" ritornare **HTTP status 400 Bad Request**[204] o **HTTP status 404 Not Found**[205]
- In caso di errori non dipendenti dalla richiesta, DEVONO restituire HTTP status 5XX rispettando la semantica degli stessi;
- Al momento della ricezione della richiesta, l'erogatore DEVE restituire **HTTP status 202 Accepted**[206]. In caso di ricezione corretta della risposta, il fruitore DEVE restituire **HTTP status 200 OK**[207], ritornando nel body di risposta un acknowledgement dell'avvenuta ricezione. In caso di errore di ricezione della risposta da parte del fruitore, è possibile utilizzare meccanismi specifici per la ritrasmissione della risposta o della richiesta.

NB: I messaggi di errore devono essere utili al client ma NON DEVONO rivelare dettagli tecnici e/o informazioni riservate.

Esempio

Specifica Servizio Server https://api.amministrazioneesempio.it/rest/nomeinterfacciaservizio/v1/RESTCallbackServer.yaml

```
openapi: 3.0.1
info:
 title: RESTCallbackServer
 version: "1.0"
 description: |-
```

(continues on next page)

[199] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/202
[200] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/200
[2] http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml
[201] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/400
[202] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/422
[203] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/404
[204] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/400
[205] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/404
[206] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/202
[207] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/200

(continua dalla pagina precedente)

```
    Questo file descrive semplicemente i metodi di un'API
    e non indica tutte le informazioni di metadatazione che
    normalmente vanno inserite.
  license:
    name: Apache 2.0 License
    url: http://www.apache.org/licenses/LICENSE-2.0.html
paths:
  /resources/{id_resource}/M:
    post:
      description: M
      operationId: PushMessage
      parameters:
      - name: X-ReplyTo
        in: header
        schema:
          type: string
      - name: id_resource
        in: path
        required: true
        schema:
          type: integer
          format: int32
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/MType'
      responses:
        202:
          description: Preso carico correttamente di M
          headers:
            X-Correlation-ID:
              required: true
              schema:
                type: string
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ACKMessage'
        400:
          description: Richiesta non valida
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorMessage'
        404:
          description: Identificativo non trovato
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorMessage'
        default:
          $ref: '#/components/responses/default'
      callbacks:
        completionCallback:
          '{$request.header#/X-ReplyTo}':
            post:
              requestBody:
                content:
                  application/json:
                    schema:
```

(continues on next page)

(continua dalla pagina precedente)

```
                    $ref: '#/components/schemas/MResponseType'
            responses:
              200:
                description: Risposta correttamente ricevuta
                content:
                  application/json:
                    schema:
                      $ref: '#/components/schemas/ACKMessage'

              default:
                $ref: '#/components/responses/default'
components:
 responses:

   default:
     description: |-
       Errore inatteso. Non ritornare informazioni
       sulla logica interna e/o non pertinenti all'interfaccia.
     content:
       application/json:
         schema:
           $ref: '#/components/schemas/ErrorMessage'
 schemas:
   MType:
     type: object
     properties:
       a:
         $ref: '#/components/schemas/AComplexType'
       b:
         type: string
   ACKMessage:
     type: object
     properties:
       outcome:
         type: string
   MResponseType:
     type: object
     properties:
       c:
         type: string
   AComplexType:
     type: object
     properties:
       a1s:
         type: array
         items:
           type: integer
           format: int32
       a2:
         type: string
   ErrorMessage:
      type: object
      properties:
        detail:
          description: |
            A human readable explanation specific to this occurrence of␣
↪the
            problem.
          type: string
        instance:
          description: |
```

(continues on next page)

(continua dalla pagina precedente)

```
            An absolute URI that identifies the specific occurrence of␣
↪the problem.
            It may or may not yield further information if dereferenced.
          format: uri
          type: string
        status:
          description: |
            The HTTP status code generated by the origin server for this␣
↪occurrence
            of the problem.
          exclusiveMaximum: true
          format: int32
          maximum: 600
          minimum: 100
          type: integer
        title:
          description: |
            A short, summary of the problem type. Written in english and␣
↪readable
            for engineers (usually not suited for non technical␣
↪stakeholders and
            not localized); example: Service Unavailable
          type: string
        type:
          default: about:blank
          description: |
            An absolute URI that identifies the problem type.  When␣
↪dereferenced,
            it SHOULD provide human-readable documentation for the␣
↪problem type
            (e.g., using HTML).
          format: uri
          type: string
```

Specifica Servizio Client https://api.indirizzoclient.it/rest/nomeinterfacciaservizio/v1/RESTCallbackClient.yaml

```
openapi: 3.0.1
info:
 title: RESTCallbackClient
 version: "1.0"
 description: |-
   Questo file descrive semplicemente i metodi di un'API
   e non indica tutte le informazioni di metadatazione che
   normalmente vanno inserite.
 license:
   name: Apache 2.0 License
   url: http://www.apache.org/licenses/LICENSE-2.0.html
paths:
 /MResponse:
   post:
     description: M
     operationId: PushResponseMessage
     parameters:
     - name: X-Correlation-ID
       in: header
       schema:
         type: string
     requestBody:
       content:
```

(continues on next page)

(continua dalla pagina precedente)

```
        application/json:
          schema:
            $ref: '#/components/schemas/MResponseType'
    responses:
      200:
        description: Risposta correttamente ricevuta
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ACKMessage'
      400:
        description: Richiesta non valida
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ErrorMessage'
      404:
        description: Identificativo non trovato
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ErrorMessage'
      default:
        description: |-
          Errore inatteso. Non ritornare informazioni
          sulla logica interna e/o non pertinenti all'interfaccia.
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ErrorMessage'

components:
 schemas:
   ACKMessage:
     type: object
     properties:
       outcome:
         type: string
   MResponseType:
     type: object
     properties:
       c:
         type: string
   ErrorMessage:
      type: object
      properties:
        detail:
          description: |
            A human readable explanation specific to this occurrence of␣
↪the
            problem.
          type: string
        instance:
          description: |
            An absolute URI that identifies the specific occurrence of␣
↪the problem.
            It may or may not yield further information if dereferenced.
          format: uri
          type: string
        status:
          description: |
```

(continues on next page)

(continua dalla pagina precedente)

```
            The HTTP status code generated by the origin server for this␣
↪occurrence
            of the problem.
          exclusiveMaximum: true
          format: int32
          maximum: 600
          minimum: 100
          type: integer
        title:
          description: |
            A short, summary of the problem type. Written in english and␣
↪readable
            for engineers (usually not suited for non technical␣
↪stakeholders and
            not localized); example: Service Unavailable
          type: string
        type:
          default: about:blank
          description: |
            An absolute URI that identifies the problem type.  When␣
↪dereferenced,
            it SHOULD provide human-readable documentation for the␣
↪problem type
            (e.g., using HTML).
          format: uri
          type: string
```

Di seguito un esempio di chiamata al metodo M con la presa in carico da parte dell'erogatore.

HTTP Operation POST

Endpoint https://api.amministrazioneesempio.it/rest/nomeinterfacciaservizio/v1/resources/1234/M

1) Request Header & Body

```
POST /rest/nomeinterfacciaservizio/v1/resources/1234/M HTTP/1.1
Content-Type: application/json
X-ReplyTo: https://api.indirizzoclient.it/rest/v1/nomeinterfacciaclient/Mresponse

{
  "a": {
    "a1": [1,...,2],
    "a2": "RGFuJ3MgVG9vbHMgYXJlIGNvb2wh"
  },
  "b": "Stringa di esempio"
}
```

2) Response Header & Body (**HTTP status 202 Accepted**[208])

```
HTTP/1.1 202 Accepted
Content-Type: application/json
X-Correlation-ID: 69a445fb-6a9f-44fe-b1c3-59c0f7fb568d

{
  "result" : "ACK"
}
```

Di seguito un esempio di risposta da parte dell'erogatore verso il fruitore.

HTTP Operation POST

Endpoint https://api.indirizzoclient.it/rest/v1/nomeinterfacciaclient/Mresponse

[208] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/202

3) Request Header & Body

```
POST /rest/v1/nomeinterfacciaclient/Mresponse HTTP/1.1
X-Correlation-ID: 69a445fb-6a9f-44fe-b1c3-59c0f7fb568d

{
   "c": "OK"
}
```

4) Response Header & Body (**HTTP status 200 OK**[209])

```
HTTP/1.1 200 Success
Content-Type: application/json

{
  "result" : "ACK"
}
```

Interfaccia SOAP

Nel caso di implementazione mediante tecnologia SOAP, l'endpoint di callback ed il correlation ID, vengono inseriti all'interno dell'header SOAP come campi custom. Erogatore e fruitore DEVONO inoltre seguire le seguenti regole:

- Le specifiche delle interfacce del fruitore e dell'erogatore DEVONO dichiarare tutti i metodi esposti con relativi schemi dei messaggi di richiesta e di ritorno. Inoltre le interfacce devono specificare eventuali header SOAP richiesti;
- La specifica dell'interfaccia del fruitore DEVE rispettare quanto richiesto dall'erogatore; in particolare si richiede che l'erogatore fornisca un WSDL descrittivo del servizio di callback che il fruitore è tenuto ad implementare;
- Al passo (1), il fruitore DEVE indicare l'endpoint della callback utilizzando l'header SOAP custom X-ReplyTo;
- Al passo (2), l'erogatore DEVE fornire insieme all'acknowledgement della richiesta nel body, il correlation ID utilizzando l'header SOAP custom X-Correlation-ID;
- Al passo (3), l'erogatore DEVE riutilizzare lo stesso correlation ID fornito al passo (2) sempre utilizzando l'header SOAP custom X-Correlation-ID;
- Al passo (4), il fruitore DEVE riconoscere tramite un messaggio di acknowledgement il ricevimento della risposta.

Regole di processamento

Nel caso di errore il WS-I Basic Profile Version 2.0 richiede l'utilizzo del meccanismo della SOAP fault per descrivere i dettagli dell'errore. In particolare, al ricevimento della richiesta, fruitore ed erogatore:

- DEVONO verificare la validità sintattica dei dati in ingresso. In caso di dati errati DEVONO restituire **HTTP status 500 Internal Server Error**[210] fornendo dettagli circa l'errore, utilizzando il meccanismo della SOAP falut;
- Nel caso in cui qualcuno degli ID nel path o nel body non esista, DEVONO restituire **HTTP status 500 Internal Server Error**[211], indicando nel body di risposta quale degli ID è mancante;

[209] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/200
[210] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/500
[211] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/500

- Se ipotizzano che la richiesta sia malevola POSSONO ritornare **HTTP status 400 Bad Request**[212] o **HTTP status 404 Not Found**[213]
- In caso di errori non dipendenti dal fruitore, DEVE restituire i codici HTTP 5XX rispettando la semantica degli stessi o restituire il codice HTTP status 500 indicando il motivo dell'errore nella SOAP fault;
- Al momento della ricezione della richiesta, DEVONO restituire un codice 2XX, nel dettaglio:
  - **HTTP status 200 OK**[214] in caso di presenza della payload SOAP, riempiendo il body di risposta con il risultato relativo alla richiesta.
  - **HTTP status 200 OK**[215] o **HTTP status 202 Accepted**[216] in caso di assenza della payload SOAP
- Nel caso di errore al momento di ricezione della risposta da parte del richiedente (fruitore o erogatore), è possibile definire meccanismi specifici per la ritrasmissione delle richieste.

Esempio

Specifica Servizio Server https://api.amministrazioneesempio.it/soap/nomeinterfacciaservizio/v1?wsdl

```
<?xml version="1.0"?>
<wsdl:definitions
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns:tns="http://amministrazioneesempio.it/
↪nomeinterfacciaservizio"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap12/"
        name="SOAPCallbackServerService"
        targetNamespace="http://amministrazioneesempio.it/
↪nomeinterfacciaservizio">
        <wsdl:types>
                <xs:schema
                        xmlns:xs="http://www.w3.org/2001/XMLSchema"
                        xmlns:tns="http://amministrazioneesempio.it/
↪nomeinterfacciaservizio"
                        attributeFormDefault="unqualified"␣
↪elementFormDefault="unqualified"
                        targetNamespace="http://amministrazioneesempio.it/
↪nomeinterfacciaservizio">

                        <xs:element name="MRequest" type="tns:MRequest"/>
                        <xs:element name="MRequestResponse" type=
↪"tns:MRequestResponse"/>

                        <xs:element name="ErrorMessageFault" nillable=
↪"true" type="tns:errorMessageFault"/>
                        <xs:element name="X-ReplyTo" nillable="true" type=
↪"xs:string"/>
                        <xs:element name="X-Correlation-ID" nillable="true
↪" type="xs:string"/>

                        <xs:complexType name="MRequest">
                                <xs:sequence>
                                        <xs:element minOccurs="0" name="M
↪" type="tns:mType"/>
                                </xs:sequence>
                        </xs:complexType>
```

(continues on next page)

[212] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/400
[213] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/404
[214] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/200
[215] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/200
[216] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/202

(continua dalla pagina precedente)

```
                        <xs:complexType name="mType">
                                <xs:sequence>
                                        <xs:element minOccurs="0" name="o_
↪id" type="xs:int"/>
                                        <xs:element minOccurs="0" name="a
↪" type="tns:aComplexType"/>
                                        <xs:element minOccurs="0" name="b
↪" type="xs:string"/>
                                </xs:sequence>
                        </xs:complexType>

                        <xs:complexType name="aComplexType">
                                <xs:sequence>
                                        <xs:element maxOccurs="unbounded"␣
↪minOccurs="0" name="a1s" nillable="true" type="xs:string"/>
                                        <xs:element minOccurs="0" name="a2
↪" type="xs:string"/>
                                </xs:sequence>
                        </xs:complexType>

                        <xs:complexType name="MRequestResponse">
                                <xs:sequence>
                                        <xs:element minOccurs="0" name=
↪"return" type="tns:ackMessage"/>
                                </xs:sequence>
                        </xs:complexType>

                        <xs:complexType name="ackMessage">
                                <xs:sequence>
                                        <xs:element minOccurs="0" name=
↪"outcome" type="xs:string"/>
                                </xs:sequence>
                        </xs:complexType>

                        <xs:complexType name="errorMessageFault">
                                <xs:sequence>
                                        <xs:element minOccurs="0" name=
↪"customFaultCode" type="xs:string"/>
                                </xs:sequence>
                        </xs:complexType>


                </xs:schema>
        </wsdl:types>

        <wsdl:message name="MRequest">
                <wsdl:part element="tns:MRequest" name="parameters"/>
                <wsdl:part element="tns:X-ReplyTo" name="X-ReplyTo"/>
        </wsdl:message>

        <wsdl:message name="MRequestResponse">
                <wsdl:part element="tns:MRequestResponse" name="result"/>
                <wsdl:part element="tns:X-Correlation-ID" name="X-
↪Correlation-ID"/>
        </wsdl:message>

        <wsdl:message name="ErrorMessageException">
                <wsdl:part element="tns:ErrorMessageFault" name=
↪"ErrorMessageException"/>
        </wsdl:message>
```

(continues on next page)

(continua dalla pagina precedente)

```
        <wsdl:portType name="SOAPCallback">
                <wsdl:operation name="MRequest">
                        <wsdl:input message="tns:MRequest" name="MRequest
↪"/>
                        <wsdl:output message="tns:MRequestResponse" name=
↪"MRequestResponse"/>
                        <wsdl:fault message="tns:ErrorMessageException"␣
↪name="ErrorMessageException"/>
                </wsdl:operation>
        </wsdl:portType>

        <wsdl:binding name="SOAPCallbackServiceSoapBinding" type=
↪"tns:SOAPCallback">
                <soap:binding style="document" transport="http://schemas.
↪xmlsoap.org/soap/http"/>
                <wsdl:operation name="MRequest">
                        <soap:operation soapAction="" style="document"/>
                        <wsdl:input name="MRequest">
                                <soap:header message="tns:MRequest" part=
↪"X-ReplyTo" use="literal"/>
                                <soap:body parts="parameters" use="literal
↪"/>
                        </wsdl:input>
                        <wsdl:output name="MRequestResponse">
                                <soap:header message="tns:MRequestResponse
↪" part="X-Correlation-ID" use="literal"/>
                                <soap:body parts="result" use="literal"/>
                        </wsdl:output>
                        <wsdl:fault name="ErrorMessageException">
                                <soap:fault name="ErrorMessageException"␣
↪use="literal"/>
                        </wsdl:fault>
                </wsdl:operation>
        </wsdl:binding>

        <wsdl:service name="SOAPCallbackService">
                <wsdl:port name="SOAPCallbackPort" binding=
↪"tns:SOAPCallbackServiceSoapBinding" >
                        <soap:address location="https://api.
↪amministrazioneesempio.it/soap/nomeinterfacciaservizio/v1"/>
                </wsdl:port>
        </wsdl:service>

</wsdl:definitions>
```

Specifica Servizio Callback https://api.indirizzoclient.it/soap/nomeinterfacciaservizio/v1?wsdl

```
<?xml version='1.0' encoding='UTF-8'?>
<wsdl:definitions
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns:tns="http://amministrazioneesempio.it/
↪nomeinterfacciaservizio"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap12/"
        name="SOAPCallbackClientInterfaceService"
        targetNamespace="http://amministrazioneesempio.it/
↪nomeinterfacciaservizio">
        <wsdl:types>
                <xs:schema
                        xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

(continues on next page)

(continua dalla pagina precedente)

```
                        xmlns:tns="http://amministrazioneesempio.it/
↪nomeinterfacciaservizio"
                        attributeFormDefault="unqualified"␣
↪elementFormDefault="unqualified"
                        targetNamespace="http://amministrazioneesempio.it/
↪nomeinterfacciaservizio">

                        <xs:element name="MRequestResponse" type=
↪"tns:MRequestResponse"/>
                        <xs:element name="MRequestResponseResponse" type=
↪"tns:MRequestResponseResponse"/>
                        <xs:element name="X-Correlation-ID" nillable="true
↪" type="xs:string"/>

                        <xs:complexType name="MRequestResponse">
                                <xs:sequence>
                                        <xs:element minOccurs="0" name=
↪"return" type="tns:mResponseType"/>
                                </xs:sequence>
                        </xs:complexType>

                        <xs:complexType name="mResponseType">
                                <xs:sequence>
                                        <xs:element minOccurs="0" name="c
↪" type="xs:string"/>
                                </xs:sequence>
                        </xs:complexType>

                        <xs:complexType name="MRequestResponseResponse">
                                <xs:sequence>
                                        <xs:element minOccurs="0" name=
↪"return" type="tns:ackMessage"/>
                                </xs:sequence>
                        </xs:complexType>

                        <xs:complexType name="ackMessage">
                                <xs:sequence>
                                        <xs:element minOccurs="0" name=
↪"outcome" type="xs:string"/>
                                </xs:sequence>
                        </xs:complexType>

                </xs:schema>
        </wsdl:types>

        <wsdl:message name="MRequestResponse">
                <wsdl:part element="tns:MRequestResponse" name="parameters
↪"/>
                <wsdl:part element="tns:X-Correlation-ID" name="X-
↪Correlation-ID"/>
        </wsdl:message>

        <wsdl:message name="MRequestResponseResponse">
                <wsdl:part element="tns:MRequestResponseResponse" name=
↪"result"/>
        </wsdl:message>

        <wsdl:portType name="SOAPCallbackClient">
                <wsdl:operation name="MRequestResponse">
                        <wsdl:input message="tns:MRequestResponse" name=
↪"MRequestResponse"/>
```

(continues on next page)

(continua dalla pagina precedente)

```
                        <wsdl:output message="tns:MRequestResponseResponse
↪" name="MRequestResponseResponse"/>
                </wsdl:operation>
        </wsdl:portType>

        <wsdl:binding name="SOAPCallbackClientServiceSoapBinding" type=
↪"tns:SOAPCallbackClient">
                <soap:binding style="document" transport="http://schemas.
↪xmlsoap.org/soap/http"/>
                <wsdl:operation name="MRequestResponse">
                        <soap:operation soapAction="" style="document"/>
                        <wsdl:input name="MRequestResponse">
                                <soap:header message="tns:MRequestResponse
↪" part="X-Correlation-ID" use="literal"/>
                                <soap:body   parts="parameters" use=
↪"literal"/>
                        </wsdl:input>
                        <wsdl:output name="MRequestResponseResponse">
                                <soap:body parts="result" use="literal" />
                        </wsdl:output>
                </wsdl:operation>
        </wsdl:binding>
        <wsdl:service name="SOAPCallbackClientService">
                <wsdl:port binding=
↪"tns:SOAPCallbackClientServiceSoapBinding" name="SOAPCallbackClientPort
↪">
                        <soap:address location="https://api.
↪indirizzoclient.it/soap/nomeinterfacciaservizio/v1"/>
                </wsdl:port>
        </wsdl:service>
</wsdl:definitions>
```

Segue un esempio di chiamata al metodo M in cui l'erogatore conferma di essersi preso carico della richiesta.

Endpoint https://api.amministrazioneesempio.it/soap/nomeinterfacciaservizio/v1

Method MRequest

1) Request Body

```
<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:m="http://amministrazioneesempio.it/nomeinterfacciaservizio">
    <soap:Header>
        <m:X-ReplyTo>https://api.indirizzoclient.it/soap/nomeinterfacciaservizio/v1
↪</m:X-ReplyTo>
    </soap:Header>
    <soap:Body>
        <m:MRequest>
            <M>
                <o_id>1234</o_id>
                <a>
                    <a1s>1</a1s>
                    <a2>prova</a2>
                </a>
                <b>prova</b>
            </M>
        </m:MRequest>
    </soap:Body>
</soap:Envelope>
```

2) Response Body

```
<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:m="http://amministrazioneesempio.it/nomeinterfacciaservizio">>
    <soap:Header>
        <m:X-Correlation-ID>b8268033-de67-4fa0-bf06-caebbfa5117a</m:X-Correlation-
↪ID>
    </soap:Header>
    <soap:Body>
        <m:MRequestResponse>
            <return>
                <outcome>ACCEPTED</outcome>
            </return>
        </m:MRequestResponse>
    </soap:Body>
</soap:Envelope>
```

Endpoint https://api.indirizzoclient.it/soap/nomeinterfacciaclient/v1

Method MRequestResponse

3. Request Body

```
<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:m="http://amministrazioneesempio.it/nomeinterfacciaservizio">
    <soap:Header>
        <m:X-Correlation-ID>b8268033-de67-4fa0-bf06-caebbfa5117a</m:X-Correlation-
↪ID>
    </soap:Header>
    <soap:Body>
        <m:MRequestResponse>
            <return>
                <c>OK</c>
            </return>
        </m:MRequestResponse>
    </soap:Body>
</soap:Envelope>
```

4. Response Body

```
<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:m="http://amministrazioneesempio.it/nomeinterfacciaservizio">
    <soap:Body>
        <m:MRequestResponseResponse>
            <return>
                <outcome>OK</outcome>
            </return>
        </m:MRequestResponseResponse>
    </soap:Body>
</soap:Envelope>
```

Profilo non bloccante RPC PULL (busy waiting)

Scenario

Questo scenario è simile al precedente, di cui eredita le motivazioni, ma in questo caso si decide che il fruitore non fornisce un indirizzo di callback, mentre l'erogatore fornisce un indirizzo interrogabile per verificare lo stato di processamento di una richiesta e, al fine dell'elaborazione della stessa, il risultato.

Descrizione

Questo scenario prevede due possibili workflow, uno per REST ed uno per SOAP.

Il fruitore invia una richiesta (passo (1)) e riceve immediatamente un acknowledge (passo (2)) insieme ad:

- un indirizzo dove verificare lo stato del processamento o della risorsa (REST);
- oppure un correlation ID (SOAP).

D'ora in poi il fruitore, periodicamente, verifica lo stato della richiesta utilizzando (passo (3)):

- l'url indicato (REST)
- oppure il correlation ID (SOAP)

fin quando la risposta alla richiesta sarà pronta (passo (4)).

Gli intervalli di polling possono essere definiti tramite i meccanismi definiti in «Robustezza».

A questo punto il fruitore può accedere al risultato o alla risorsa finale (passi (5) e (6)).

Interfaccia REST

Fig. 4.8: Interazione non bloccante tramite busy waiting

Nel caso in cui il profilo venga implementato con tecnologia REST, DEVONO essere rispettate le seguenti regole:

- La specifica dell'interfaccia dell'erogatore DEVE dichiarare tutti i codici di stato HTTP restituiti con relativo schema della risposta, oltre che ad eventuali header HTTP restituiti;
- La specifica dell'interfaccia DEVE dichiarare gli schemi delle richieste insieme ad eventuali header HTTP richiesti;
- Al passo (1), il fruitore DEVE utilizzare il verbo HTTP POST;
- Al passo (2), l'erogatore DEVE fornire insieme all'acknowledgement della richiesta, un percorso di risorsa per interrogare lo stato di processamento utilizzando **HTTP header Location** [217]; Il codice HTTP di stato DEVE essere **HTTP status 202 Accepted**[218] a meno che non si verifichino errori;
- Al passo (3), il fruitore DEVE utilizzare il percorso di cui al passo (2) per richiedere lo stato della risorsa; Il verbo HTTP utilizzato deve essere GET;
- Al passo (4) l'erogatore indica che la risorsa non è ancora pronta, fornendo informazioni circa lo stato della lavorazione della richiesta; il codice HTTP restituito è **HTTP status 200 OK**[219];
- Al passo (6) l'erogatore indica che la risorsa è pronta, utilizzando **HTTP header Location** [220]; per indicare il percorso dove recuperare la risorsa, il codice HTTP restituito è **HTTP status 303 See Other**[221];
- Al passo (8) l'erogatore risponde con la rappresentazione della risorsa,Il codice HTTP restituito è **HTTP status 200 OK**[222];

Regole di processamento

Al termine del processamento delle richieste, l'erogatore deve fare uso dei codici di stato HTTP rispettandone la semantica[3]. In particolare, al ricevimento della richiesta da parte del fruitore, l'erogatore:

[217] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Location
[218] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/202
[219] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/200
[220] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Location
[221] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/303
[222] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/200
[3] http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml

- DEVE verificare la validità sintattica e semantica dei dati in ingresso;
- DEVE, in caso di dati errati, restituire **HTTP status 400 Bad Request**[223] fornendo nel body di risposta dettagli circa l'errore;
- DEVE, in caso di representation semanticamente non corretta, ritornare **HTTP status 422** [224];
- DEVE, se qualcuno degli ID nel path o nel body non esiste, restituire **HTTP status 404 Not Found**[225], indicando nel body di risposta quale degli ID è mancante;
- PUO", se ipotizza che la richiesta sia malevola, ritornare **HTTP status 400 Bad Request**[226] o **HTTP status 404 Not Found**[227]
- DEVE, in caso di errori non dipendenti dalla richiesta, restituire HTTP status 5XX rispettando la semantica degli stessi;
- DEVE, ricevuta la richiesta, restituire **HTTP status 202 Accepted**[228].

  In caso di ricezione corretta della risposta, il fruitore DEVE restituire **HTTP status 200 OK**[229] , riempiendo il body di risposta con il risultato dell'operazione.

  In caso di errore al momento di ricezione della risposta da parte del fruitore, è possibile definire meccanismi specifici per la ritrasmissione della risposta o della richiesta.

NB: I messaggi di errore devono essere utili al client ma NON DEVONO rivelare dettagli tecnici e/o informazioni riservate.

Esempio

Specifica Servizio Server: https://api.amministrazioneesempio.it/rest/nomeinterfacciaservizio/v1/openapi.yaml

```
openapi: 3.0.1
info:
 title: RESTbusywaiting
 version: "1.0"
 description: |-
   Questo file descrive semplicemente i metodi di un'API
   e non indica tutte le informazioni di metadatazione che
   normalmente vanno inserite.
 license:
   name: Apache 2.0 License
   url: http://www.apache.org/licenses/LICENSE-2.0.html
paths:
  /tasks/queue:
    post:
      description: Crea in maniera asincrona un task o una risorsa.
      operationId: PushMessage
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/MType'
      responses:
        202:
          description: Preso carico correttamente
          headers:
            Location:
```

(continues on next page)

[223] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/400
[224] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/422
[225] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/404
[226] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/400
[227] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/404
[228] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/202
[229] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/200

(continua dalla pagina precedente)

```
            description: URL dove verificare lo stato
            required: true
            schema:
              type: string
              format: uri
      '400':
        $ref: '#/components/responses/400BadRequest'

      default:
        $ref: '#/components/responses/default'
/tasks/queue/{id_task}/:
  get:
    description: Verifica lo stato del task o risorsa
    operationId: CheckStatus
    parameters:
    - $ref: '#/components/parameters/id_task'
    responses:
      200:
        description: |-
          Lo stato del task o della risorsa.
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/TaskStatus'
      '303':
        description: Task Completato
        content:
            application/json:
              schema:
                $ref: '#/components/schemas/TaskStatus'
        headers:
            Location:
              description: URL dove prelevare il risultato
              required: true
              schema:
                type: string
                format: uri
      '400':
        $ref: '#/components/responses/400BadRequest'
      '404':
        $ref: '#/components/responses/404NotFound'
      default:
        $ref: '#/components/responses/default'
/tasks/result/{id_task}/:
  get:
    description: Recupera il risultato del task o la risorsa
    operationId: RetriveResource
    parameters:
    - $ref: '#/components/parameters/id_task'
    responses:
      200:
        description:  Il risultato del task o la risorsa
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/MResponseType'
      '400':
        $ref: '#/components/responses/400BadRequest'
      '404':
        $ref: '#/components/responses/404NotFound'
      default:
```

(continues on next page)

(continua dalla pagina precedente)

```
          $ref: '#/components/responses/default'
components:
  parameters:
    id_task:
      name: id_task
      in: path
      required: true
      schema:
        type: string
  responses:
    400BadRequest:
      description: Richiesta non accoglibile
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/ErrorMessage'
    404NotFound:
      description: Identificativo non trovato
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/ErrorMessage'
    default:
      description: |-
        Errore inatteso. Questo viene ritornato nel caso ci sia
        un errore inatteso. Non vanno mai esposti i dati interni
        del server.
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/ErrorMessage'
  schemas:
    TaskStatus:
      type: object
      properties:
        status:
          type: string
          enum: [pending, completed]
          example: pending
        message:
          type: string
    MType:
      type: object
      properties:
        a:
          $ref: '#/components/schemas/AComplexType'
        b:
          type: string
    MResponseType:
      type: object
      properties:
        c:
          type: string

    AComplexType:
      type: object
      properties:
        a1s:
          type: array
          items:
            type: string
```

(continues on next page)

(continua dalla pagina precedente)

```
        a2:
          type: string
    ErrorMessage:
      type: object
      properties:
        detail:
          description: |
            A human readable explanation specific to this occurrence of the
            problem.
          type: string
        instance:
          description: |
            An absolute URI that identifies the specific occurrence of the problem.
            It may or may not yield further information if dereferenced.
          format: uri
          type: string
        status:
          description: |
            The HTTP status code generated by the origin server for this occurrence
            of the problem.
          exclusiveMaximum: true
          format: int32
          maximum: 600
          minimum: 100
          type: integer
        title:
          description: |
            A short, summary of the problem type. Written in english and readable
            for engineers (usually not suited for non technical stakeholders and
            not localized); example: Service Unavailable
          type: string
        type:
          default: about:blank
          description: |
            An absolute URI that identifies the problem type.  When dereferenced,
            it SHOULD provide human-readable documentation for the problem type
            (e.g., using HTML).
          format: uri
          type: string
```

Di seguito un esempio di chiamata ad M in cui l'erogatore dichiara di essersi preso carico della richiesta.

Endpoint https://api.amministrazioneesempio.it/rest/nomeinterfacciaservizio/v1/resources/1234/M

```
POST /rest/nomeinterfacciaservizio/v1/resources/1234/M   HTTP/1.1

{
   "a": {
     "a1": [1,...,2],
     "a2": "Stringa di esempio"
   },
   "b": "Stringa di esempio"
 }
```

```
HTTP/1.1 202 Accepted
Content-Type: application/json
Location:  resources/1234/M/8131edc0-29ed-4d6e-ba43-cce978c7ea8d
```

(continues on next page)

(continua dalla pagina precedente)

```
{
  "status": "accepted",
  "message": "Preso carico della richiesta",
  "id": "8131edc0-29ed-4d6e-ba43-cce978c7ea8d"
}
```

Di seguito un esempio di chiamata con cui il fruitore verifica l'esecuzione di M nei casi di processamento ancora in atto (4) e di processamento avvenuto (5).

Endpoint https://api.amministrazioneesempio.it/rest/nomeinterfacciaservizio/v1/resources/1234/M/8131edc0-29ed-4d6e-ba43-cce978c7ea8d

```
HTTP/1.1 200 Success

{
  "status": "processing",
  "message": "Richiesta in fase di processamento"
}
```

```
HTTP/1.1 303 See Other

{
  "status": "done",
  "message": "Processamento completo"
}
```

Di seguito un esempio di chiamata con cui il fruitore richiede l'esito della sua richiesta.

Endpoint https://api.amministrazioneesempio.it/rest/nomeinterfacciaservizio/v1/resources/1234/M/8131edc0-29ed-4d6e-ba43-cce978c7ea8d/result

```
HTTP/1.1 200 Success

{
  "c": "OK"
}
```

Interfaccia SOAP

Fig. 4.9: Interazione non bloccante tramite busy waiting

Nel caso in cui il profilo venga implementato con tecnologia SOAP, DEVONO essere rispettate le seguenti regole:

- L'interfaccia di servizio dell'erogatore fornisce tre metodi differenti al fine di inoltrare una richiesta, controllarne lo stato ed ottenerne il risultato;
- La specifica dell'interfaccia dell'erogatore DEVE indicare l'header SOAP X-Correlation-ID;
- Al passo (2), l'erogatore DEVE fornire insieme all'acknowledgement della richiesta nel body, un correlation ID riportato nel header custom SOAP X-Correlation-ID;
- Al passo (3), il fruitore DEVE utilizzare i l correlation ID ottenuto al passo (2) per richiedere lo stato di processamento di una specifica richiesta;
- Al passo (4) l'erogatore, quando il processamento non si è ancora concluso fornisce informazioni circa lo stato della lavorazione della richiesta, quando invece il processamento si è concluso risponde indicando in maniera esplicita il completamento;

- Al passo (5), il fruitore utilizza il correlation ID di cui al passo (2) al fine di richiedere il risultato della richiesta;
- Al passo (6), l'erogatore fornisce il risultato del processamento.

Regole di processamento

Nel caso di errore il WS-I Basic Profile Version 2.0 richiede l'utilizzo del meccanismo della SOAP fault per descrivere i dettagli dell'errore. Al ricevimento della richiesta da parte del fruitore, l'erogatore:

- DEVE verificare la validità sintattica dei dati in ingresso. In caso di dati errati DEVE restituire **HTTP status 500 Internal Server Error**[230] fornendo dettagli circa l'errore utilizzando il meccanismo della SOAP fault;
- Se l'erogatore ipotizza che la richiesta sia malevola PUO" ritornare **HTTP status 400 Bad Request**[231] o **HTTP status 404 Not Found**[232]
- In caso di errori non dipendenti dal fruitore, DEVE restituire i codici HTTP 5XX rispettando la semantica degli stessi o restituire il codice HTTP status 500 indicando il motivo dell'errore nella SOAP fault;
- In caso di successo restituire **HTTP status 200 OK**[233], riempiendo il body di risposta con il risultato dell'operazione.

Esempio

Specifica Servizio Server: https://api.amministrazioneesempio.it/soap/nomeinterfacciaservizio/v1?wsdl

```
<?xml version='1.0' encoding='UTF-8'?>
<wsdl:definitions
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns:tns="http://amministrazioneesempio.it/nomeinterfacciaservizio"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap12/"
        name="SOAPPullService"
        targetNamespace="http://amministrazioneesempio.it/nomeinterfacciaservizio">
        <wsdl:types>
                <xs:schema
                        xmlns:xs="http://www.w3.org/2001/XMLSchema"
                        xmlns:tns="http://amministrazioneesempio.it/
↪nomeinterfacciaservizio"
                        attributeFormDefault="unqualified" elementFormDefault=
↪"unqualified"
                        targetNamespace="http://amministrazioneesempio.it/
↪nomeinterfacciaservizio">

                        <xs:element name="MProcessingStatus" type=
↪"tns:MProcessingStatus" />
                        <xs:element name="MProcessingStatusResponse" type=
↪"tns:MProcessingStatusResponse" />

                        <xs:element name="MRequest" type="tns:MRequest" />
                        <xs:element name="MRequestResponse" type=
↪"tns:MRequestResponse" />

                        <xs:element name="MResponse" type="tns:MResponse" />
                        <xs:element name="MResponseResponse" type=
↪"tns:MResponseResponse" />
```

(continues on next page)

[230] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/500
[231] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/400
[232] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/404
[233] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/200

(continua dalla pagina precedente)

```
                        <xs:element name="ErrorMessageFault" nillable="true" type=
↪"tns:errorMessageFault" />
                        <xs:element name="X-Correlation-ID" nillable="true" type=
↪"xs:string" />


                        <xs:complexType name="MProcessingStatus"/>
                        <xs:complexType name="MProcessingStatusResponse">
                                <xs:sequence>
                                        <xs:element name="return" type=
↪"tns:processingStatus" />
                                </xs:sequence>
                        </xs:complexType>

                        <xs:complexType name="MRequest">
                                <xs:sequence>
                                        <xs:element name="M" type="tns:mType" />
                                </xs:sequence>
                        </xs:complexType>
                        <xs:complexType name="MRequestResponse">
                                <xs:sequence>
                                        <xs:element minOccurs="0" name="return
↪"        type="tns:processingStatus" />
                                </xs:sequence>
                        </xs:complexType>

                        <xs:complexType name="MResponse"/>
                        <xs:complexType name="MResponseResponse">
                                <xs:sequence>
                                        <xs:element minOccurs="0" name="return"␣
↪type="tns:mResponseType" />
                                </xs:sequence>
                        </xs:complexType>

                        <xs:complexType name="mType">
                                <xs:sequence>
                                        <xs:element minOccurs="0" name="o_id" type=
↪"xs:int" />
                                        <xs:element minOccurs="0" name="a" type=
↪"tns:aComplexType" />
                                        <xs:element minOccurs="0" name="b" type=
↪"xs:string" />
                                </xs:sequence>
                        </xs:complexType>
                        <xs:complexType name="aComplexType">
                                <xs:sequence>
                                        <xs:element maxOccurs="unbounded"␣
↪minOccurs="0"        name="a1s" nillable="true" type="xs:string" />
                                        <xs:element minOccurs="0" name="a2" type=
↪"xs:string" />
                                </xs:sequence>
                        </xs:complexType>


                        <xs:complexType name="processingStatus">
                                <xs:sequence>
                                        <xs:element name="status" type="xs:string"␣
↪/>
                                        <xs:element name="message" type="xs:string
↪" />
```

(continues on next page)

(continua dalla pagina precedente)

```
                                </xs:sequence>
                        </xs:complexType>

                        <xs:complexType name="mResponseType">
                                <xs:sequence>
                                        <xs:element minOccurs="0" name="c" type=
↪"xs:string" />
                                </xs:sequence>
                        </xs:complexType>

                        <xs:complexType name="errorMessageFault">
                                <xs:sequence>
                                        <xs:element name="customFaultCode" type=
↪"xs:string" />
                                </xs:sequence>
                        </xs:complexType>

                </xs:schema>
        </wsdl:types>

        <wsdl:message name="MProcessingStatus">
                <wsdl:part element="tns:MProcessingStatus" name="parameters"/>
                <wsdl:part element="tns:X-Correlation-ID" name="X-Correlation-ID"/>
        </wsdl:message>

        <wsdl:message name="MProcessingStatusResponse">
                <wsdl:part element="tns:MProcessingStatusResponse"        name=
↪"parameters"/>
        </wsdl:message>

        <wsdl:message name="MRequest">
                <wsdl:part element="tns:MRequest" name="parameters"/>
        </wsdl:message>

        <wsdl:message name="MRequestResponse">
                <wsdl:part element="tns:MRequestResponse" name="result"/>
                <wsdl:part element="tns:X-Correlation-ID" name="X-Correlation-ID"/>
        </wsdl:message>

        <wsdl:message name="MResponse">
                <wsdl:part element="tns:MResponse" name="parameters"/>
                <wsdl:part element="tns:X-Correlation-ID" name="X-Correlation-ID"/>
        </wsdl:message>

        <wsdl:message name="MResponseResponse">
                <wsdl:part element="tns:MResponseResponse" name="parameters"/>
        </wsdl:message>

        <wsdl:message name="ErrorMessageException">
                <wsdl:part element="tns:ErrorMessageFault" name=
↪"ErrorMessageException"/>
        </wsdl:message>

        <wsdl:portType name="SOAPPull">

                <wsdl:operation name="MRequest">
                        <wsdl:input message="tns:MRequest" name="MRequest"/>
                        <wsdl:output message="tns:MRequestResponse" name=
↪"MRequestResponse"/>
                        <wsdl:fault message="tns:ErrorMessageException" name=
↪"ErrorMessageException"/>
```

(continues on next page)

(continua dalla pagina precedente)

```
                </wsdl:operation>

                <wsdl:operation name="MProcessingStatus">
                        <wsdl:input message="tns:MProcessingStatus"        name=
↪"MProcessingStatus"/>
                        <wsdl:output message="tns:MProcessingStatusResponse" name=
↪"MProcessingStatusResponse"/>
                        <wsdl:fault message="tns:ErrorMessageException
↪"        name="ErrorMessageException"/>
                </wsdl:operation>

                <wsdl:operation name="MResponse">
                        <wsdl:input message="tns:MResponse" name="MResponse"/>
                        <wsdl:output message="tns:MResponseResponse" name=
↪"MResponseResponse"/>
                        <wsdl:fault message="tns:ErrorMessageException" name=
↪"ErrorMessageException"/>
                </wsdl:operation>

        </wsdl:portType>
        <wsdl:binding name="SOAPPullServiceSoapBinding" type="tns:SOAPPull">
                <soap:binding style="document"        transport="http://schemas.
↪xmlsoap.org/soap/http" />

                <wsdl:operation name="MRequest">
                        <soap:operation soapAction="" style="document" />

                        <wsdl:input name="MRequest">
                                <soap:body use="literal" />
                        </wsdl:input>

                        <wsdl:output name="MRequestResponse">
                                <soap:header message="tns:MRequestResponse
↪"        part="X-Correlation-ID" use="literal"/>
                                <soap:body parts="result" use="literal" />
                        </wsdl:output>

                        <wsdl:fault name="ErrorMessageException">
                                <soap:fault name="ErrorMessageException" use=
↪"literal" />
                        </wsdl:fault>

                </wsdl:operation>

                <wsdl:operation name="MProcessingStatus">
                        <soap:operation soapAction="" style="document" />

                        <wsdl:input name="MProcessingStatus">
                                <soap:header message="tns:MProcessingStatus" part=
↪"X-Correlation-ID" use="literal"/>
                                <soap:body parts="parameters" use="literal" />
                        </wsdl:input>

                        <wsdl:output name="MProcessingStatusResponse">
                                <soap:body use="literal" />
                        </wsdl:output>

                        <wsdl:fault name="ErrorMessageException">
                                <soap:fault name="ErrorMessageException" use=
↪"literal" />
                        </wsdl:fault>
```

(continues on next page)

(continua dalla pagina precedente)

```
                </wsdl:operation>

                <wsdl:operation name="MResponse">
                        <soap:operation soapAction="" style="document" />

                        <wsdl:input name="MResponse">
                                <soap:header message="tns:MResponse" part="X-
↪Correlation-ID" use="literal"/>
                                <soap:body parts="parameters" use="literal" />
                        </wsdl:input>

                        <wsdl:output name="MResponseResponse">
                                <soap:body use="literal" />
                        </wsdl:output>

                        <wsdl:fault name="ErrorMessageException">
                                <soap:fault name="ErrorMessageException" use=
↪"literal" />
                        </wsdl:fault>

                </wsdl:operation>

        </wsdl:binding>

        <wsdl:service name="SOAPPullService">
                <wsdl:port binding="tns:SOAPPullServiceSoapBinding" name=
↪"SOAPPullPort">
                        <soap:address location="https://api.amministrazioneesempio.
↪it/soap/nomeinterfacciaservizio/v1" />
                </wsdl:port>
        </wsdl:service>
</wsdl:definitions>
```

Di seguito un esempio di chiamata ad M in cui l'erogatore risponde di avere preso in carico la richiesta.

Endpoint https://api.amministrazioneesempio.it/soap/nomeinterfacciaservizio/v1

Method MRequest

- 1) Richiesta

```
<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:m="http://amministrazioneesempio.it/nomeinterfacciaservizio">
    <soap:Body>
        <m:MRequest
                <M>
                    <o_id>1234</o_id>
                    <a>
                        <a1s>1</a1s>
                        <a2>prova</a2>
                    </a>
                    <b>prova</b>
                </M>
            </m:MRequest>
        </soap:Body>
</soap:Envelope>
```

- 2) Risposta

```
<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:m="http://amministrazioneesempio.it/nomeinterfacciaservizio">
        <soap:Header>
            <m:X-Correlation-ID>c8e191a8-f34f-41ed-82ea-68e096466707</m:X-
↪Correlation-ID>
        </soap:Header>
        <soap:Body>
            <m:MRequestResponse>
                <return>
                    <status>accepted</status>
                    <message>Preso carico della richiesta</message>
                </return>
            </m:MRequestResponse>
        </soap:Body>
</soap:Envelope>
```

Di seguito un esempio di chiamata con cui il fruitore verifica l'esecuzione di M nei casi di processamento ancora in atto e di processamento avvenuto (4).

Endpoint https://api.amministrazioneesempio.it/soap/nomeinterfacciaservizio/v1

Method MProcessingStatus

- 3) richiesta verifica status

```
<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:m="http://amministrazioneesempio.it/nomeinterfacciaservizio">
        <soap:Header>
            <m:X-Correlation-ID>c8e191a8-f34f-41ed-82ea-68e096466707</m:X-
↪Correlation-ID>
        </soap:Header>
        <soap:Body>
            <m:MProcessingStatus/>
        </soap:Body>
</soap:Envelope>
```

- 4) risposta verifica status in attesa

```
<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:m="http://amministrazioneesempio.it/nomeinterfacciaservizio">
        <soap:Body>
            <m:MProcessingStatusResponse>
                <return>
                    <status>processing</status>
                    <message>Richiesta in fase di processamento</message>
                </return>
            </m:MProcessingStatusResponse>
        </soap:Body>
</soap:Envelope>
```

- 4) risposta verifica status completato

```
<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:m="http://amministrazioneesempio.it/nomeinterfacciaservizio">
    <soap:Body>
        <m:MProcessingStatusResponse>
            <return>
                <status>done</status>
                <message>Richiesta completata</message>
```

(continues on next page)

(continua dalla pagina precedente)

```
            </return>
        </m:MProcessingStatusResponse>
    </soap:Body>
</soap:Envelope>
```

Di seguito un esempio di chiamata con cui il fruitore richiede l'esito della sua richiesta.

Endpoint https://api.amministrazioneesempio.it/soap/nomeinterfacciaservizio/v1

Method MProcessingStatus

- 5) richiesta recupero entry

```
<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:m="http://amministrazioneesempio.it/nomeinterfacciaservizio">
    <soap:Header>
        <m:X-Correlation-ID>c8e191a8-f34f-41ed-82ea-68e096466707</m:X-Correlation-
↪ID>
    </soap:Header>
    <soap:Body>
        <m:MResponse/>
    </soap:Body>
</soap:Envelope>
```

- 6) risposta recupero entry

```
<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:m="http://amministrazioneesempio.it/nomeinterfacciaservizio">
    <soap:Body>
        <m:MResponseResponse">
            <return>
                <c>OK</c>
            </return>
        </m:MResponseResponse>
    </soap:Body>
</soap:Envelope>
```

4.2.4 Accesso CRUD a Risorse

Scenario

In molti casi, le interfacce di servizio vengono utilizzate più che per eseguire compiti complessi, al fine di eseguire operazioni di tipo CRUD - Create, Read, Update, Delete su risorse del dominio di interesse. Ad esempio, una prenotazione è una risorsa che può essere creata (quando viene fissata), letta, modificata ed eliminata. Questo genere di operazione ha durata molto breve, e quindi deve essere implementata mediante una modalità bloccante come quella vista in Sezione 3.2.2.

Descrizione

Fig. 4.10: Interazione bloccante CRUD

Il sequence diagram è, come si nota in figura, equivalente a quello di una RPC bloccante. In questo caso la richiesta DEVE contenere:

- Una operazione da effettuare sulla risorsa da scegliere tra Create, Read, Update e Delete;

- Un percorso che indica un identificativo di risorsa (es. una specifica prenotazione) oppure di una collezione di risorse (es. un insieme di prenotazioni);
- Nel caso di una operazione di Create o Update, un pacchetto dati indicante come inizializzare o modificare (anche parzialmente) la risorsa in questione.

Per quanto sia possibile sviluppare funzionalità CRUD anche sviluppando una interfaccia di servizio SOAP, il ModI consiglia nei casi in cui occorra un approccio a risorse, su cui è naturale utilizzare operazioni CRUD, di sviluppare una interfaccia RESTful.

Interfaccia REST

Regole di processamento

L'approccio RESTful trova la sua applicazione naturale in operazioni CRUD - Create, Read, Update, Delete su risorse ed a tal fine sfrutta i verbi standard dell'HTTP per indicare tali operazioni. In particolare la seguente mappatura viene utilizzata:

CRUD	Verbo HTTP	Esito (stato HTTP) Se applicato a intera collezione (es. /clienti)	Esito (stato HTTP) Se applicato a risorsa specifica (es. /clienti/{id})
Create	POST	201 (Created), l'header "Location" nella risposta può contenere un link a /clienti/{id} che indica l'ID creato.	404 (Not Found), 409 (Conflict) se la risorsa è già esistente.
Read	GET	200 (OK), lista dei clienti. Implementare meccanismi di paginazione, ordinamento e filtraggio per navigare grandi liste.	200 (OK), 404 (Not Found), se l'ID non è stato trovato o è non valido.
Update/Replace/Create	PUT	405 (Method Not Allowed), a meno che non si voglia sostituire ogni risorsa nella collezione.	200 (OK) o 204 (No Content). 404 (Not Found), se l'ID non è stato trovato o è non valido. 201 (Created), nel caso di UPSERT.
Update/Modify	PATCH	405 (Method Not Allowed), a meno che non si voglia applicare una modifica ad ogni risorsa nella collezione.	200 (OK) o 204 (No Content). 404 (Not Found), se l'ID non è stato trovato o è non valido.
Delete	DELETE	405 (Method Not Allowed), a meno che non si voglia permettere di eliminare l'intera collezione.	200 (OK). 404 (Not Found), se l'ID non è stato trovato o è non valido.

Si noti l'uso distinto di **HTTP method PUT** [234] e **HTTP method PATCH** [235] per la sostituzione e l'applicazione di modifiche ad una risorsa rispettivamente. E" possibile utilizzare anche altri verbi HTTP a patto che si rispettino i dettami dell'approccio RESTful.

In alcuni casi il verbo **HTTP method PUT** [236] può essere utilizzato con funzionalità di UPSERT (Update o Insert). In particolare, questo è necessario nei casi in cui sia il fruitore a definire gli identificativi del sistema erogatore.

Per usare il **HTTP method PATCH** [237] bisogna usare alcuni accorgimenti, perché questo metodo non è definito nelle nuove specifiche di HTTP/1.1 del 2014 ma nel precedente **RFC 5789**[238].

Come indicato in questa considerazione https://www.rfc-editor.org/errata/eid3169 **NON SI DOVREBBE** associare un significato di patch a dei media-type che non lo prevedono (eg. `application/json` o `application/xml`) ma utilizzare dei media-type adeguati.

E" possibile ad esempio usare `application/merge-patch+json` definito in **RFC 7386**[239] facendo attenzione:

[234] https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT
[235] https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PATCH
[236] https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT
[237] https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PATCH
[238] https://tools.ietf.org/html/rfc5789.html
[239] https://tools.ietf.org/html/rfc7386.html

- che **HTTP method PATCH** [240] rifiuti richieste con media-type non adeguato con **HTTP status 415 Unsupported Media Type**[241];
- che il media-type di patching sia compatibile con gli schemi utilizzati;
- di verificare le considerazioni di sicurezza presenti in **RFC 7396#section-5**[242] e **RFC 5789#section-5**[243].

Esempio

Per illustrare l'approccio RESTful al CRUD, esemplificheremo un API per gestire le prenotazioni di un appuntamento presso un ufficio municipale. L'erogatore, verifica la compatibilità con la disponibilità nello specifico orario ed accetta o nega la creazione o l'eventuale variazione. Come da specifica seguente i metodi implementati sono **HTTP method POST** [244] (creazione), **HTTP method DELETE** [245] (eliminazione), **HTTP method PATCH** [246] (modifica) e **HTTP method GET** [247] (lettura).

Specifica Servizio Server https://api.amministrazioneesempio.it/rest/appuntamenti/v1/openapi.yaml

```
openapi: 3.0.1
info:
  title: RESTCRUD
  version: "1.0"
  description: |-
    Questo file descrive semplicemente i metodi di un'API
    e non indica tutte le informazioni di metadatazione che
    normalmente vanno inserite.
  license:
    name: Apache 2.0 License
    url: http://www.apache.org/licenses/LICENSE-2.0.html
paths:
  /municipio/{id_municipio}/ufficio/{id_ufficio}/prenotazioni:
    get:
      description: Mostra prenotazioni
      operationId: listReservations
      parameters:
      - $ref: '#/components/parameters/limit'
      - $ref: '#/components/parameters/cursor'
      - $ref: '#/components/parameters/path_id_municipio'
      - $ref: '#/components/parameters/path_id_ufficio'
      responses:
        '200':
          description: Una lista di prenotazioni.
          content:
            application/json:
              schema:
                properties:
                  prenotazioni:
                    type: array
                    items:
                      $ref: '#/components/schemas/Prenotazione'
                  count:
                    type: integer
                    format: int32
                  next:
                    type: string
```

(continues on next page)

[240] https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PATCH
[241] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/415
[242] https://tools.ietf.org/html/rfc7396.html#section-5
[243] https://tools.ietf.org/html/rfc5789.html#section-5
[244] https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST
[245] https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/DELETE
[246] https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PATCH
[247] https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET

(continua dalla pagina precedente)

```
        '400':
          $ref: '#/components/responses/400BadRequest'
        '404':
          $ref: '#/components/responses/404NotFound'
        default:
          $ref: '#/components/responses/default'

    post:
      description: Aggiungi una prenotazione
      operationId: AddReservation_1
      parameters:
      - $ref: '#/components/parameters/path_id_municipio'
      - $ref: '#/components/parameters/path_id_ufficio'
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Prenotazione'
      responses:
        '201':
          description: Prenotazione Creata.
          headers:
            Location:
              description: ID della prenotazione creata
              schema:
                type: string
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Prenotazione'
        '400':
          $ref: '#/components/responses/400BadRequest'
        '404':
          $ref: '#/components/responses/404NotFound'
        default:
          $ref: '#/components/responses/default'

  /municipio/{id_municipio}/ufficio/{id_ufficio}/prenotazioni/{id_
↪prenotazione}:
    get:
      description: LeggiPrenotazione
      operationId: GetReservation_1
      parameters:
      - $ref: '#/components/parameters/path_id_municipio'
      - $ref: '#/components/parameters/path_id_ufficio'
      - name: id_prenotazione
        in: path
        required: true
        schema:
          type: integer
          format: int32
      responses:
        '200':
          description: Prenotazione estratta correttamente
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Prenotazione'
        '400':
          $ref: '#/components/responses/400BadRequest'
        '404':
```

(continues on next page)

(continua dalla pagina precedente)

```
          $ref: '#/components/responses/404NotFound'
        default:
          $ref: '#/components/responses/default'
    delete:
      description: EliminaPrenotazione
      operationId: DeleteReservation
      parameters:
      - $ref: '#/components/parameters/path_id_municipio'
      - $ref: '#/components/parameters/path_id_ufficio'
      - name: id_prenotazione
        in: path
        required: true
        schema:
          type: integer
          format: int32
      responses:
        '200':
          description: Prenotazione eliminata correttamente
        '404':
          $ref: '#/components/responses/404NotFound'
        default:
          $ref: '#/components/responses/default'

    patch:
      description: ModificaPrenotazione
      operationId: PatchReservation
      parameters:
      - $ref: '#/components/parameters/path_id_municipio'
      - $ref: '#/components/parameters/path_id_ufficio'
      - name: id_prenotazione
        in: path
        required: true
        schema:
          type: integer
          format: int32
      requestBody:
        content:
          application/merge-patch+json:
            schema:
              $ref: '#/components/schemas/PatchPrenotazione'
      responses:
        '200':
          description: Prenotazione modificata correttamente
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Prenotazione'
        '400':
          $ref: '#/components/responses/400BadRequest'
        '404':
          $ref: '#/components/responses/404NotFound'
        default:
          $ref: '#/components/responses/default'
components:
  parameters:
    path_id_municipio:
      name: id_municipio
      in: path
      required: true
      schema:
        type: integer
```

(continues on next page)

(continua dalla pagina precedente)

```
        format: int32
    path_id_ufficio:
      name: id_ufficio
      in: path
      required: true
      schema:
        type: integer
        format: int32
    limit:
        description: How many items to return at one time (max 100)
        in: query
        name: limit
        schema:
          format: int32
          type: integer
    cursor:
        description: An opaque identifier that points to the next item in␣
↪the collection.
        example: 01BX9NSMKVXXS5PSP2FATZM123
        in: query
        name: cursor
        schema:
          type: string
  responses:
    400BadRequest:
      description: Richiesta non accoglibile
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/ErrorMessage'
    404NotFound:
      description: Identificativo non trovato
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/ErrorMessage'
    default:
      description: |-
        Errore inatteso. Questo viene ritornato nel caso ci sia
        un errore inatteso. Non vanno mai esposti i dati interni
        del server.
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/ErrorMessage'
  schemas:
    TaxCode:
        description: Il codice fiscale.
        example: RSSMRA75L01H501A
        externalDocs:
          url: https://w3id.org/italia/onto/CPV/taxCode
        pattern: /^(?:(?:[B-DF-HJ-NP-TV-Z]|[AEIOU])[AEIOU][AEIOUX]|[B-DF-
↪HJ-NP-TV-Z]{2}[A-Z]){2}[\dLMNP-V]{2}(?:[A-EHLMPR-T](?:[04LQ][1-9MNP-
↪V]|[1256LMRS][\dLMNP-V])|[DHPS][37PT][0L]|[ACELMRT][37PT][01LM])(?:[A-
↪MZ][1-9MNP-V][\dLMNP-V]{2}|[A-M][0L](?:[1-9MNP-V][\dLMNP-V]|[0L][1-9MNP-
↪V]))[A-Z]$/i
        type: string
    Prenotazione:
      type: object
      properties:
        nome:
```

(continues on next page)

(continua dalla pagina precedente)

```
          type: string
        cognome:
          type: string
        codice_fiscale:
          $ref: '#/components/schemas/TaxCode'
        dettagli:
          $ref: '#/components/schemas/PatchPrenotazione'
    PatchPrenotazione:
      type: object
      properties:
        data:
          type: string
          format: date-time
        motivazione:
          type: string
    ErrorMessage:
      type: object
      properties:
        detail:
          description: |
            A human readable explanation specific to this occurrence of␣
↪the
            problem.
          type: string
        instance:
          description: |
            An absolute URI that identifies the specific occurrence of␣
↪the problem.
            It may or may not yield further information if dereferenced.
          format: uri
          type: string
        status:
          description: |
            The HTTP status code generated by the origin server for this␣
↪occurrence
            of the problem.
          exclusiveMaximum: true
          format: int32
          maximum: 600
          minimum: 100
          type: integer
        title:
          description: |
            A short, summary of the problem type. Written in english and␣
↪readable
            for engineers (usually not suited for non technical␣
↪stakeholders and
            not localized); example: Service Unavailable
          type: string
        type:
          default: about:blank
          description: |
            An absolute URI that identifies the problem type.  When␣
↪dereferenced,
            it SHOULD provide human-readable documentation for the␣
↪problem type
            (e.g., using HTML).
          format: uri
          type: string
```

Di seguito un esempio di chiamata per creare una prenotazione.

Listato 4.9: Request

```
POST /rest/appuntamenti/v1/municipio/{id_municipio}/ufficio/{id_ufficio}/
↪prenotazioni HTTP/1.1

{
  "nome_proprio": "Mario",
  "cognome": "Rossi",
  "codice_fiscale": "MRORSS77T05E472I",
  "dettagli": {
    "data": "2018-12-03T14:29:12.137Z",
    "motivazione": "string"
  }
}
```

Listato 4.10: Response

```
HTTP/1.1 201 Created
Location: https://api.amministrazioneesempio.it/rest/appuntamenti/v1/municipio/{id_
↪municipio}/ufficio/{id_ufficio}/prenotazioni/12323254

{
  "id": 12323254,
  "nome_proprio": "Mario",
  "cognome": "Rossi",
  "codice_fiscale": "MRORSS77T05E472I",
  "dettagli": {
    "data": "2018-12-03T14:29:12.137Z",
    "motivazione": "string"
  }
}
```

Di seguito un esempio in cui il fruitore richiede l'estrazione di una specifica prenotazione. Si noti l'utilizzo dell'URL restituito nell'" **HTTP header Location** [248] al passo precedente.

Listato 4.11: Request

```
GET /rest/appuntamenti/v1/municipio/{id_municipio}/ufficio/{id_ufficio}/
↪prenotazioni/12323254  HTTP/1.1
```

Listato 4.12: Response

```
HTTP/1.1 200 OK

{
  "id": 12323254,
  "nome_proprio": "Mario",
  "cognome": "Rossi",
  "codice_fiscale": "MRORSS77T05E472I",
  "dettagli": {
    "data": "2018-12-03T14:29:12.137Z",
    "motivazione": "string"
  }
}
```

Di seguito una richiesta di modifica dei dettagli di una prenotazione.

[248] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Location

Listato 4.13: Request

```
PATCH /rest/appuntamenti/v1/municipio/{id_municipio}/ufficio/{id_ufficio}/
↪prenotazioni/12323254  HTTP/1.1
Content-Type: application/merge-patch+json

{
   "dettagli": {
     "data": "2018-12-03T14:29:12.137Z",
     "motivazione": "nuova motivazione"
   }
}
```

Listato 4.14: Response

```
HTTP/1.1 200 OK

{
  "nome_proprio": "Mario",
  "cognome": "Rossi",
  "codice_fiscale": "MRORSS77T05E472I",
  "dettagli": {
      "data": "2018-12-03T14:29:12.137Z",
      "motivazione": "nuova motivazione"
   }
}
```

Di seguito una richiesta di modifica dei dettagli di una prenotazione con media-type `application/json`, che non avendo una semantica di patching definita, dev'essere rifiutato seguendo le indicazioni presenti in **RFC 5789#section-2.2**[249]. La response ritorna il media-type suggerito dalla specifica tramite **HTTP header Accept-Patch** [250]

Listato 4.15: Request

```
PATCH /rest/appuntamenti/v1/municipio/{id_municipio}/ufficio/{id_ufficio}/
↪prenotazioni/12323254  HTTP/1.1
Content-Type: application/json

{
   "dettagli": {
     "data": "2018-12-03T14:29:12.137Z",
     "motivazione": "nuova motivazione"
   }
}
```

Listato 4.16: Response

```
HTTP/1.1 415 Unsupported Media Type
Accept-Patch: application/merge-patch+json
```

Di seguito un esempio di cancellazione di una specifica prenotazione.

Listato 4.17: Request

```
DELETE /rest/appuntamenti/v1/municipio/{id_municipio}/ufficio/{id_ufficio}/
↪prenotazioni/12323254  HTTP/1.1
```

[249] https://tools.ietf.org/html/rfc5789.html#section-2.2
[250] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept-Patch

Listato 4.18: Response

```
HTTP/1.1 200 OK
```

4.2.5 Raccomandazioni tecniche per REST

In questa sezione si raccolgono per la tecnologia REST delle indicazioni al fine di favorire l'interoperabilità.

Formato dei dati

Nella tecnologia REST, la comunicazione DOVREBBE avvenire tramite oggetti JSON **RFC 7159**[251] con il relativo media-type[252] `application/json`.

E" possibile eccepire in presenza di specifiche in cui gli oggetti di comunicazione sono formalizzati in forma diversa da JSON (es. INSPIRE, HL7).

Codificare dati strutturati con oggetti JSON

I dati in formato JSON **RFC 7159**[253] strutturati DEVONO essere trasferiti tramite oggetti , in modo da permettere l'estensione retrocompatibile della response con ulteriori attributi (Eg. paginazione).

Cioè:

- il payload di una response contenente una entry ritorna un oggetto

```
{ "given_name": "Paolo", "last_name": "Rossi", "id": 313 }
```

- il payload di una response contenente più entry ritorna un oggetto contenente una lista e non direttamente una lista.

```
{
   "items":[
      {
         "given_name":"Carlo",
         "family_name":"Bianchi",
         "id":314
      },
      {
         "given_name":"Giuseppe",
         "family_name":"Verdi",
         "id":315
      }
   ]
}
```

Evitare Content-Type personalizzati

Si raccomanda di evitare l'uso di media-type personalizzati come da **RFC 6838**[254] section-3.4 (eg. `application/x.custom.name+json`) ed utilizzare nomi standard come:

- application/json[255],

[251] https://tools.ietf.org/html/rfc7159.html
[252] https://www.iana.org/assignments/media-types/media-types.xhtml
[253] https://tools.ietf.org/html/rfc7159.html
[254] https://tools.ietf.org/html/rfc6838.html
[255] https://www.iana.org/assignments/media-types/application/json

- application/problem+json[256],
- application/jose+json[257],

Si raccomanda di utilizzare Content-Type semanticamente coerenti

Quando si ritornano dati binari, immagini o documenti (eg. pdf, png, ...) utilizzare

Utilizzare le properties secondo nomenclature standard

Le properties DEVONO utilizzare ove possibile la nomenclatura indicata nelle Linee Guida per la valorizzazione del Patrimonio informativo pubblico[258] e le relative ontologie[259].

Si raccomanda di utilizzare formati standard per Data ed Ora

Le date DEVONO essere conformi ad **RFC 3339**[260], ad esempio *2015-05-28* per la data e *2015-05-28T14:07:17Z* per l'ora.

Le date negli header HTTP DEVONO essere conformi allo standard HTTP date format definito in **RFC 7231**[261].

RFC 3339[262] permette di indicare una timezone prefissando la data con la distanza da UTC:

- *2015-05-28T14:07:17+01:00*
- *2015-05-28T14:07:17-05:00*

Quando la data è specificata in UTC occorre utilizzare sempre il suffisso Z (Zulu time zone)

- *2015-05-28T14:07:17Z*

Tempi di durata e intervalli devono utilizzare ISO 8601.

Di seguito alcuni esempi di durata in formato ISO 8601 per i tempi di durata[263].

Le durate sono prefissate da «P», giorni e Ore sono separati da «T».

Esempi:

- `P1Y2M3D` - 1 anno, 2 mesi e 3 giorni
- `PT1H4M5S` - 1 ora, 4 minuti e 5 secondi
- `P1M` - 1 mese
- `PT1M` - 1 minuto
- `P1Y2M10DT2H30M` - 1 anno, 2 mesi, 10 giorni 2 ore e 30 minuti

Un'analoga sintassi ISO8601 per lo stesso intervallo è la seguente:

```
P0001-02-10T2:30:00
```

[256] https://www.iana.org/assignments/media-types/application/problem+json
[257] https://www.iana.org/assignments/media-types/application/jose+json
[258] https://docs.italia.it/italia/daf/lg-patrimonio-pubblico/it/bozza/
[259] https://github.com/italia/daf-ontologie-vocabolari-controllati
[260] https://tools.ietf.org/html/rfc3339.html
[261] https://tools.ietf.org/html/rfc7231.html
[262] https://tools.ietf.org/html/rfc3339.html
[263] https://www.iso.org/iso-8601-date-and-time-format.html

Utilizzare le seguenti convenzioni di rappresentazione

- I booleani non DEVONO essere `null`.
- Gli array vuoti non DEVONO essere `null`, ma liste vuote, ad es. `[]`.
- Le enumeration DEVONO essere rappresentate da stringhe non nulle.

Usare standard per Lingue, Nazioni e Monete

Utilizzare per le codifiche web gli standard indicati in Linee Guida per la Valorizzazione del Patrimonio Informativo Nazionale, inclusi:

- ISO 3166-1-alpha2 country (due lettere)[264]
- ISO 639-1 language code[265]
- **:BCP:'47'** (basato su ISO 639-1) per le varianti dei linguaggi.
- ISO 4217 alpha-3 currency codes[266]

Per le valute, è possibile basarsi sullo schema Money - ripreso dal lavoro di standardizzazione del Berlin Group sotto l'egida dell'European Standards[267] e contenente i campi:

- amount (string)
- currency (iso-4217)

Esempio 1:

```
{
   "tax_id": "imu-e472",
   "value": {
      "amount": "100.23",
      "currency": "EUR"
   }
}
```

Definire `format` quando si usano i tipi Number ed Integer

I numeri e gli interi DEVONO indicare la dimensione utilizzando il parametro `format`. La seguente tabella - non esaustiva - elenca un set minimo di formati. Le implementazioni DEVONO utilizzare il tipo più adatto.

Le parti possono concordare la definizione di nuovi tipi, che dev'essere documentata nell'interfaccia.

type	format	valori ammessi
integer	int32	interi tra -2^31 e 2^31-1
integer	int64	interi tra -2^63 e 2^63-1
number	decimal32 / float	IEEE 754-2008/ISO 60559:2011 decimale a 32 bit
number	decimal64 / double	IEEE 754-2008/ISO 60559:2011 decimale a 64 bit
number	decimal128	IEEE 754-2008/ISO 60559:2011 decimale a 128 bit

Le proprietà degli oggetti JSON devono avere un naming consistente

Segliere uno dei due stili di seguito e codificarlo in ASCII:

[264] https://www.iso.org/iso-3166-country-codes.html
[265] https://www.iso.org/standard/22109.html
[266] https://www.iso.org/iso-4217-currency-codes.html
[267] https://www.berlin-group.org/

- snake_case
- camelCase

Non usare contemporaneamente snake_case e camelCase nella stessa API.

Analogamente non usare contemporaneamente i due stili nella naming convention, ad esempio

- sì: `{ "givenName": "Mario", "familyName": "Rossi"}`
- sì: `{ "given_name": "Mario", "family_name": "Rossi"}`
- no: `{ "givenName": "Mario", "family_name": "Rossi"}`

Progettazione e Naming delle Interfacce di Servizio

In assenza di specifiche regole (es. HL7, INSPIRE, ..) per l'API Naming, valgono le seguenti.

Uso corretto dei metodi HTTP

I metodi HTTP devono essere utilizzati rispettando la semantica indicata in **RFC 7231**[268] section-4.3.

Uso corretto degli header HTTP

In generale gli header:

- DEVONO essere utilizzati solo per passare informazioni di contesto
- la semantica e gli intenti delle operazioni deve essere definita tramite URI, Status e Method e non dagli Header, che dovrebbero supportare funzionalità di protocollo come indicato in **RFC 7231**[269].

Prima di usare un header:

- si deve verificare se è già adottato da IANA _IANA_message_headers

Usare parole separate da trattino «-» per i Path (kebab-case)

Questo si applica solo al Path.

Esempio:

```
/tax-code/{tax_code_id}
```

Inoltre, il Path dovrebbe essere semplice, intuitivo e coerente.

Preferire Hyphenated-Pascal-Case per gli header HTTP

Esempi:

```
Accept-Encoding

Apply-To-Redirect-Ref

Disposition-Notification-Options

Original-Message-ID
```

[268] https://tools.ietf.org/html/rfc7231.html
[269] https://tools.ietf.org/html/rfc7231.html

Le collezioni di risorse possono usare nomi al plurale

Si consiglia di differenziare il nome delle collezioni e delle risorse. Questo permette di separare a livello di URI, endpoint che sono in larga parte funzionalmente differenti.

Esempio 1: ricerca documenti per data in una collezione

```
GET /documenti?data=2018-05-01

{
  "items": [ .. ]
  "limit": 10
  "next_cursor": 21314123
}
```

Esempio 2: recupera un singolo documento

```
GET /documento/21314123

{

  "id": 21314123
  "title: "Atto di nascita ...",
  ..
}
```

Utilizzare Query Strings standardizzate

Esempio 1: La paginazione dev'essere implementata tramite i parametri:

```
cursor, limit, offset, sort
```

Esempio 2: La ricerca, il filtering e l'embedding dei parametri dev'essere implementata tramite i parametri:

```
q, fields, embed
```

Non passare credenziali o dati riservati nell'URL

Eventuali dati riservati o credenziali e token di autenticazione NON DEVONO essere passati nei query parameters o comunque nell'URL.

Non usare l'header `Link` RFC 8288[270] se la response è in JSON

Eventuali link a risorse vanno restituiti nel payload. Va" invece evitato di ritornare l'header `Link` definito in **RFC 8288**[271] (già **RFC 5988**[272]).

Usare URI assoluti nei risultati

Restituendo URI assoluti si indica chiaramente al client l'indirizzo delle risorse di destinazione e non si obbligano i client a fare «inferenza» dal contesto.

[270] https://tools.ietf.org/html/rfc8288.html
[271] https://tools.ietf.org/html/rfc8288.html
[272] https://tools.ietf.org/html/rfc5988.html

Usare lo schema Problem JSON per le risposte di errore

In caso di errori si deve ritornare:

- un payload di tipo Problem definito in **RFC 7807**[273]
- il media type dev'essere `application/problem+json`
- lo status code dev'essere esplicativo
- l'oggetto può essere esteso

Quando si restituisce un errore è importante *non esporre dati interni* delle applicazioni.

Per prevenire il rischio di user-enumeration, i messaggi di errore di autenticazione non devono fornire informazioni sull'esistenza o meno dell'utenza.

Dopo aver validato il contenuto delle richieste si DEVE ritornare:

- **HTTP status 415 Unsupported Media Type**[274] se il Content-Type non è supportato;
- **HTTP status 400 Bad Request**[275] o **HTTP status 404 Not Found**[276] se si ipotizza che la richiesta sia malevola;
- **HTTP status 422** [277] se la representation della richiesta è sintatticamente corretta ma semanticamente non processabile.

Ottimizzare l'uso della banda e migliorare la responsività

Utilizzare quando possibile:

- tecniche di compressione;
- paginazione;
- un filtro sugli attributi necessari;
- le specifiche di optimistic locking (**HTTP header ETag** [278], if-(none-)match) **RFC 7232**[279]

E" possibile ridurre l'uso della banda e velocizzare le richieste filtrando i campi delle risorse restituite.

Esempio 1: Non filtrato

Listato 4.19: Request

```
GET https://api.example.org/resources/123 HTTP/1.1
```

Listato 4.20: Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id":"cddd5e44-dae0-11e5-8c01-63ed66ab2da5",
  "name":"Mario Rossi",
  "address":"via del Corso, Roma, Lazio, Italia",
  "birthday":"1984-09-13",
  "partner":{
    "id":"1fb43648-dae1-11e5-aa01-1fbc3abb1cd0",
```

(continues on next page)

[273] https://tools.ietf.org/html/rfc7807.html
[274] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/415
[275] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/400
[276] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/404
[277] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/422
[278] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/ETag
[279] https://tools.ietf.org/html/rfc7232.html

(continua dalla pagina precedente)

```
    "name":"Maria Rossi",
    "address":"via del Corso, Roma, Lazio, Italia",
    "birthday":"1988-04-07"
  }
}
```

Esempio 2: Filtrato

Listato 4.21: Request

```
GET /resources/123?fields=(name,partner(name)) HTTP/1.1
```

Listato 4.22: Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "name": "Mario Rossi",
    "partner": {
        "name": "Maria Rossi"
    }
}
```

Effettuare la Resource Expansion permette di ridurre il numero di richieste, quando bisogna ritornare risorse correlate tra loro.

In tal caso va usato:

- il parametro `embed` utilizzando lo stesso formato dei campi per il filtering
- l'attributo `_embedded` contenente le entry espanse.

Listato 4.23: Request

```
GET /tax_code/MRORSS12T05E472W?embed=(person) HTTP/1.1
Accept: application/json
```

Listato 4.24: Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "tax_code":"MRORSS12T05E472W",
  "_embedded":{
    "person":{
      "given_name":"Mario",
      "family_name":"Rossi",
      "id":"1234-ABCD-7890"
    }
  }
}
```

Di default il caching http deve essere disabilitato

Il caching va" disabilitato tramite **HTTP header Cache-Control** [280] per evitare che delle richieste vengano inopportunamente messe in cache. Esempio:

[280] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control

Le API che supportano il caching devono documentare le varie limitazioni e modalità di utilizzo tramite gli header definiti in **RFC 7234**[281]

- **HTTP header Cache-Control** [282]
- **HTTP header Vary** [283]

Eventuali conflitti nella creazione di risorse vanno gestiti tramite gli header:

- ETag[284]
- If-Match[285]
- If-None-Match[286].

contenenti un hash del response body, un hash dell'ultimo campo modificato della entry o un numero di versione.

Riferimenti

Specifiche

- OpenAPI Specification[287]
- **:BCP:'bcp47'**

Articoli

- Roy Thomas Fielding - Architectural Styles and the Design of Network-Based[288]
- Software Architectures[289] Definizione teorica dell'approccio REST.

Libri

- PIs: From Start to Finish[290]
- Blogs[291]
- Service Design Patterns[292]
- REST in Practice: Hypermedia and Systems Architecture[293]
- Build APIs You Won't Hate[294]
- InfoQ eBook - Web APIs: From Start to Finish[295]¶[296]

Blogs

- Lessons-learned blog: Thoughts on RESTful API Design[297]

[281] https://tools.ietf.org/html/rfc7234.html
[282] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control
[283] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Vary
[284] https://tools.ietf.org/html/rfc7232#section-2.3
[285] https://tools.ietf.org/html/rfc7232#section-3.1
[286] https://tools.ietf.org/html/rfc7232#section-3.2
[287] https://github.com/OAI/OpenAPI-Specification/
[288] http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
[289] http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
[290] http://www.infoq.com/minibooks/emag-web-%20api
[291] http://www.amazon.de/REST-Practice-Hypermedia-Systems-%20Architecture/dp/0596805829
[292] http://www.servicedesignpatterns.com/
[293] http://www.amazon.de/REST-Practice-Hypermedia-Systems-%20Architecture/dp/0596805829
[294] https://leanpub.com/build-apis-you-wont-hate
[295] http://www.infoq.com/minibooks/emag-web-%20api
[296] http://www.infoq.com/minibooks/emag-web-api
[297] http://restful-api-%20design.readthedocs.org/en/latest/

4.2.6 Raccomandazioni tecniche per SOAP

Nell'ambito del protocollo SOAP ai fini dell'interoperabilità è definito in WS-I Basic Profile.

ModI assume l'adozione della specifica WS-I Basic Profile versione 2.0[298] , in quanto i framework implementativi più diffusi sono conformi a questa specifica.

Di seguito sono riportate suggerimenti al fine di favorire l'interoperabilità.

Evitare l'uso di media-type personalizzati

Si raccomanda di evitare l'uso di media-type personalizzati come da **RFC 6838**[299] section-3.4 (eg. `application/x.custom.name+xml`) ed utilizzare nomi standard come application/xml[300]

Utilizzare le properties secondo nomenclature standard

Le properties DEVONO utilizzare ove possibile la nomenclatura indicata nelle Linee Guida per la valorizzazione del Patrimonio informativo pubblico[301] e le relative ontologie[302].

Utilizzare formati standard per Data ed Ora

Le date DEVONO essere conformi ad **RFC 3339**[303], ad esempio *2015-05-28* per la data e *2015-05-28T14:07:17Z* per l'ora.

RFC 3339[304] permette di indicare una timezone prefissando la data con la distanza da UTC:

- *2015-05-28T14:07:17+01:00*
- *2015-05-28T14:07:17-05:00*

Quando la data è specificata in UTC occorre utilizzare sempre il suffisso Z (Zulu time zone)

- *2015-05-28T14:07:17Z*

Tempi di durata e intervalli devono utilizzare ISO 8601.

Di seguito alcuni esempi di durata in formato ISO 8601 per i tempi di durata[305].

Le durate sono prefissate da «P», giorni e Ore sono separati da «T».

Esempi:

- `P1Y2M3D` - 1 anno, 2 mesi e 3 giorni
- `PT1H4M5S` - 1 ora, 4 minuti e 5 secondi
- `P1M` - 1 mese
- `PT1M` - 1 minuto
- `P1Y2M10DT2H30M` - 1 anno, 2 mesi, 10 giorni 2 ore e 30 minuti

Un'analoga sintassi ISO8601 per lo stesso intervallo è la seguente:

```
P0001-02-10T2:30:00
```

[298] http://www.ws-i.org/Profiles/BasicProfile-2.0-2010-11-09.html
[299] https://tools.ietf.org/html/rfc6838.html
[300] https://www.iana.org/assignments/media-types/application/xml
[301] https://docs.italia.it/italia/daf/lg-patrimonio-pubblico/it/bozza/
[302] https://github.com/italia/daf-ontologie-vocabolari-controllati
[303] https://tools.ietf.org/html/rfc3339.html
[304] https://tools.ietf.org/html/rfc3339.html
[305] https://www.iso.org/iso-8601-date-and-time-format.html

Usare standard per Lingue, Nazioni e Monete

Utilizzare per le codifiche web gli standard indicati in Linee Guida per la Valorizzazione del Patrimonio Informativo Nazionale, inclusi:

- ISO 3166-1-alpha2 country (due lettere)[306]
- ISO 639-1 language code[307]
- **:BCP:'47'** (basato su ISO 639-1) per le varianti dei linguaggi.
- ISO 4217 alpha-3 currency codes[308]

Descrittività dei nomi utilizzati

I nomi utilizzati per servizi ed operazioni nelle interfacce di servizio devono essere auto-descrittivi e fornire quanta più informazione possibile riguardo al comportamento implementato.

Si deve inoltre evitare l'utilizzo di acronimi quando questi non siano universalmente riconosciuti anche al di fuori del dominio applicativo.

Utilizzo di camelCase e PascalCase

Per i nomi dei servizi si suggerisce di utilizzare PascalCase mentre per le operazioni implementate e gli argomenti si suggerisce l'utilizzo del camelCase.

Non includere il numero di versione all'interno del nome del servizio

Unicità dei namespace e utilizzo di pattern fissi

Ove possibile all'interno del WSDL deve essere presente un namespace unico.

I namespace utilizzati per i servizi devono seguire un pattern specifico. In particolare, per i servizi:

```
http://<dominioOrganizzativo>/ws/<DominioApplicativo>/<NomeServizio>/V<major>
```

dove:

- `<dominioOrganizzativo>` indica l'organizzazione che espone il servizio,
- `<DominioApplicativo>` indica il settore all'interno dell'organizzazione,
- `<NomeServizio>` segue le specifiche di cui ai punti precedenti, e <major> indica il numero di versione (difatti non inserito nel nome del servizio).

Per quanto riguarda gli XSD all'interno del WSDL si segue il pattern seguente:

```
http://<dominioOrganizzativo>/xmlns/<DominioApplicativo>
```

Riferimenti

SOAP 1.2 Parte 1[309] e Parte 2[310]

WS-I Basic Profile 2.0[311]

[306] https://www.iso.org/iso-3166-country-codes.html
[307] https://www.iso.org/standard/22109.html
[308] https://www.iso.org/iso-4217-currency-codes.html
[309] https://www.w3.org/TR/soap12/
[310] https://www.w3.org/TR/soap12-part2/
[311] http://ws-i.org/profiles/BasicProfile-2.0-2010-11-09.html

WS-Addressing[312]

Standard eHealth Ontario[313]

4.2.7 Robustezza

Ai fini di garantire la responsività di una interfaccia di servizio, è necessario impedire a singoli fruitori di esaurire la capacità di calcolo dell'erogatore. La tecnica comunemente utilizzata in questi casi è il rate limiting (anche noto come throttling). Il rate limit fornisce ad uno specifico fruitore un numero massimo di richieste soddisfacibili all'interno di uno specifico arco temporale (es. 1000 richieste al minuto). Un numero di richieste che superi il limite imposto provoca il rifiuto di ulteriori richieste da parte di uno specifico fruitore per un intervallo di tempo. Sulle politiche riguardanti il numero massimo di richieste e la relativa finestra temporale, e quelle riguardanti il tempo di attesa per nuove richieste (che può essere incrementato in caso di richieste reiterate, es. con una politica di aumento esponenziale) si lascia libertà agli implementatori previa un'analisi di carico massimo sopportabile dall'erogatore.

Segnalare raggiunti limiti di utilizzo

Gli erogatori di interfacce di servizio REST DEVONO segnalare eventuali limiti raggiunti con **HTTP status 429** [314].

Le API restituiscono in ogni response i valori globali di throttling tramite i seguenti header:

- `X-RateLimit-Limit`: limite massimo di richieste per un endpoint
- `X-RateLimit-Remaining`: numero di richieste rimanenti fino al prossimo reset
- `X-RateLimit-Reset`: il numero di secondi che mancano al prossimo reset

In caso di superamento delle quote, le API restituiscono anche l'header:

- **HTTP header Retry-After** [315]: il numero minimo di secondi dopo cui il client è invitato a riprovare[1]

Nel caso di interfacce di servizio SOAP non esistono regole guida standard per la gestione del rate limit e del throttling. Si suggerisce l'utilizzo degli stessi header e status code HTTP visti nel caso REST.

I fruitori devono:

- rispettare gli header di throttling
- rispettare l'header `X-RateLimit-Reset` sia quando restituisce il numero di secondi che mancano al prossimo reset, sia quando ritorna il timestamp unix
- rispettare l'header **HTTP header Retry-After** [316] sia nella variante che espone il numero di secondi dopo cui riprovare, sia nella variante che espone la data in cui riprovare

Segnalare il sovraccarico del sistema o l'indisponibilità del servizio

Gli erogatori devono definire ed esporre un piano di continuità operativa segnalando il sovraccarico del sistema o l'indisponibilità del servizio con **HTTP status 503 Service Unavailable**[317].

In caso di sovraccarico o indisponibilità, l'erogatore deve ritornare anche:

- **HTTP header Retry-After** [318] con il numero minimo di secondi dopo cui il client è invitato a riprovare

[312] https://www.w3.org/Submission/ws-addressing/
[313] https://www.ehealthontario.on.ca/architecture/education/courses/service-%20oriented-architecture/downloads/SOA-ServiceNamingConventions.pdf
[314] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/429
[315] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Retry-After
[1] **RFC 7231** prevede che l'header **HTTP header Retry-After** possa essere utilizzato sia in forma di data che di secondi
[316] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Retry-After
[317] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/503
[318] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Retry-After

I fruitori devono:

- rispettare **HTTP header Retry-After** [319] sia nella variante che espone il numero di secondi dopo cui riprovare, sia nella variante che espone la data in cui riprovare.

Si propone di seguito una specifica di servizio REST relativa ad un profilo RPC bloccante arricchito con gli header relativi al rate limiting. L'utilizzo degli header HTTP in SOAP è fuori dagli obiettivi di WSDL come Interface Definition Language.

Specifica Servizio https://api.amministrazioneesempio.it/rest/nomeinterfacciaservizio/v1/openapi.yaml

```
openapi: 3.0.1
info:
  title: RESTrobustezza
  license:
    name: Apache 2.0 License
    url: http://www.apache.org/licenses/LICENSE-2.0.html
  version: "1.0"
paths:
  /resources/{id_resource}/M:
    post:
      description: M
      operationId: PushMessage_1
      parameters:
      - name: id_resource
        in: path
        required: true
        schema:
          type: integer
          format: int32
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/MType'
      responses:
        200:
          description: Esecuzione di M avvenuta con successo
          headers: &ratelimit_headers
            X-RateLimit-Limit:
              $ref: '#/components/headers/X-RateLimit-Limit'
            X-RateLimit-Remaining:
              $ref: '#/components/headers/X-RateLimit-Remaining'
            X-RateLimit-Reset:
              $ref: '#/components/headers/X-RateLimit-Reset'
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/MResponseType'
        404:
          description: Identificativo non trovato
          headers:
            <<: *ratelimit_headers
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorMessage'
        400:
          description: Richiesta malformata
          headers:
            <<: *ratelimit_headers
          content:
```

(continues on next page)

[319] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Retry-After

(continua dalla pagina precedente)

```
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorMessage'
        429:
          description: Limite di richieste raggiunto
          headers:
            Retry-After:
              description: Limite massimo richieste
              schema:
                type: string
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorMessage'
        500:
          description: Errore interno avvenuto
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorMessage'
components:
  headers:
    Retry-After:
      description: |-
          Retry contacting the endpoint *at least* after seconds.
          See https://tools.ietf.org/html/rfc7231#section-7.1.3
      schema:
        format: int32
        type: integer
    X-RateLimit-Limit:
      description: The number of allowed requests in the current period
      schema:
        format: int32
        type: integer
    X-RateLimit-Remaining:
      description: The number of remaining requests in the current period
      schema:
        format: int32
        type: integer
    X-RateLimit-Reset:
      description: The number of seconds left in the current period
      schema:
        format: int32
        type: integer
  schemas:
    MType:
      type: object
      properties:
        a:
          $ref: '#/components/schemas/AComplexType'
        b:
          type: string
    MResponseType:
      type: object
      properties:
        c:
          type: string
    AComplexType:
      type: object
      properties:
        a1s:
```

(continues on next page)

(continua dalla pagina precedente)

```
          type: array
          items:
            type: integer
            format: int32
        a2:
          type: string
    ErrorMessage:
      type: object
      properties:
        detail:
          description: |
            A human readable explanation specific to this occurrence of the
            problem.
          type: string
        instance:
          description: |
            An absolute URI that identifies the specific occurrence of the problem.
            It may or may not yield further information if dereferenced.
          format: uri
          type: string
        status:
          description: |
            The HTTP status code generated by the origin server for this occurrence
            of the problem.
          exclusiveMaximum: true
          format: int32
          maximum: 600
          minimum: 100
          type: integer
        title:
          description: |
            A short, summary of the problem type. Written in english and readable
            for engineers (usually not suited for non technical stakeholders and
            not localized); example: Service Unavailable
          type: string
        type:
          default: about:blank
          description: |
            An absolute URI that identifies the problem type.  When dereferenced,
            it SHOULD provide human-readable documentation for the problem type
            (e.g., using HTML).
          format: uri
          type: string
```

Di seguito un esempio di chiamata al servizio bloccante con risposta nel caso in cui i limiti non siano ancora stati raggiunti e nel caso in cui invece il fruitore debba attendere per presentare nuove richieste.

Endpoint https://api.amministrazioneesempio.it/rest/nomeinterfacciaservizio/v1/resources/1234/M

Listato 4.25: 1- Request

```
POST /rest/nomeinterfacciaservizio/v1/resources/1234/M   HTTP/1.1
Host: api.amministrazioneesempio.it
Content-Type: application/json

{
  "a": {
    "a1s": [1,2],
```

(continues on next page)

(continua dalla pagina precedente)

```
    "a2": "RGFuJ3MgVG9vbHMgYXJlIGNvb2wh"
  },
  "b": "Stringa di esempio"
}
```

Listato 4.26: 2- Response

```
HTTP/1.1 200 Success
X-Rate-Limit-Limit: 30
X-Rate-Limit-Remaining: 11
X-Rate-Limit-Reset: 44

{
  "c" : "risultato"
}
```

```
HTTP/1.1 429 Too Many Requests
Content-Type: application/problem+json
Retry-After: 60

{
    "status": 429,
    "title": "Hai superato la quota di richieste."
}
```

4.2.8 Passaggio di Allegati

Tra i parametri o i valori di ritorno di una interfaccia di servizio può esserci la presenza di allegati, dove per allegato si intende un contenuto binario o la cui struttura comunque non è definita direttamente dall'interfaccia di servizio (e.g., un file XML all'interno di un messaggio di SOAP in cui lo schema che definisce il messaggio SOAP non specifica anche la struttura dell'XML allegato). In generale, un allegato può essere passato o ricevuto nelle seguenti forme:

- Codificata in modo da essere rappresentabile con un set predefinito di caratteri. Il caso più comune è quello della codifica Base64.
- Come URL ad una risorsa esterna o in ogni caso come endpoint di accesso ad una risorsa.
- Nel suo formato binario originale.

Nei primi due casi, l'allegato fa parte del contenuto XML o JSON del messaggio, mentre nel secondo caso si ricorre a risposte di tipo multipart.

Ognuna di queste modalità presenta dei pro e dei contro. L'utilizzo di codifiche come Base64 oppure di URL ha il vantaggio di potere inserire un allegato all'interno del contenuto principale del messaggio. Si noti come anche nel caso XML l'utilizzo di formati binari all'interno dei campi CDATA sia sconsigliato poiché esiste il rischio che il contenuto binario possa in alcuni casi chiudere il campo CDATA stesso. D'altro canto l'utilizzo di codifiche comporta un incremento nella banda richiesta poiché l'occupazione dei dati non è ottimale. L'utilizzo di URL comporta invece un rischio potenziale nel fatto che la risorsa collegata possa essere modificata o rimossa nel tempo.

In ambito SOAP la prima proposta di standard per l'invio di allegati binari è rappresentato da SWA - SOAP with Attachments, a cui W3C ha però preferito come standard MTOM - Message Transmission Optimization Mechanism che ha il vantaggio di estendere agli allegati i meccanismi di sicurezza quali WS-Encryption e WS-Signature. Sebbene la cosa non sia immediata, MTOM è solitamente utilizzato insieme a XOP - XML-binary Optimized Packaging quale meccanismo per fare riferimento agli allegati all'interno del messaggio Multipart/Related. L'utilizzo di MTOM con XOP è supportato da tutti i maggiori framework per lo sviluppo di interfacce di servizio SOAP,

ma meccanismi similari, sempre basati su XOP, sono supportati anche dai maggiori framework per lo sviluppo di interfacce di servizio REST.

L'utilizzo di un approccio o di un altro è fortemente dipendente dallo scenario applicativo. Le seguenti regole possono essere introdotte:

- L'invio di allegati binari corrispondenti a file fa preferire solitamente l'invio di dati in formato binario, quindi mediante MTOM/XOP.
- L'utilizzo di Base64 è consigliato per l'invio di allegati di dimensioni ridotte quali ad esempio firme digitali o codici di controllo.
- L'utilizzo di URL può essere considerato nel caso in cui gli allegati siano di dimensioni tali da rendere il trasferimento via rete oneroso, a patto che si possa assicurare la persistenza della risorsa (in termini temporali) e che questa non venga modificata (a tal fine è possibile utilizzare tecniche ad esempio di hashing). Quest'ultimo caso richiede quindi solitamente trust tra fruitore ed erogatore.

Di seguito una tabella riepilogativa. In particolare si userà la scala decrescente ++, +, -, – per indicare quanto una specifica tecnologia è adatta in determinate condizioni:

Tecnologia	Invio File	File di grande dimensione	Mancanza di trust rispetto alla capacità del fruitore di garantire persistenza	Allegati di dimensioni ridotte
MTOM/XOP	+	-	++	+
Base64	-	–	++	++
URL	+	++	–	–

4.3 Concetti di base

Interazione bloccante vs non bloccante

Nell'interazione bloccante un fruitore effettua una chiamata al servente ed attende una risposta prima di continuare l'esecuzione. La chiamata codifica in modo opportuno la richiesta di servizio, anche attraverso il passaggio di dati (sia in input alla chiamata che in output nella risposta).

Nell'interazione non bloccante, invece, il fruitore invia un messaggio ma non si blocca in attesa di alcuna risposta (se non una notifica di presa in carico). Il messaggio contiene in modo opportuno la richiesta ed eventuali dati di input. Talvolta il messaggio, proprio ad indicare il fatto che codifica la richiesta e le informazioni necessarie a soddisfarla, viene indicato come documento. La risposta da parte del servente, nei casi in cui ci sia, può apparire significativamente più tardi, ove significativamente va interpretato rispetto al tempo di computazione proprio dell'interazione[2]. Anche la risposta del servente viene inviata tramite un messaggio.

Con abuso di nomenclatura, la comunicazione bloccante talvolta viene detta *sincrona*, ad indicare che client e servente si sono sincronizzati (attesa di uno da parte dell'altro); quella non bloccante viene detta *asincrona*, proprio a significare l'asincronicità che vi è tra l'invio di un messaggio e la risposta al messaggio stesso.

Alonso, G., Casati, F., Kuno, H., Machiraju, V. (2004). Web Services. Concepts, Architectures and Applications. Springer

---

Remote Procedure Call

Una Remote Procedure Call (chiamata a procedura remota, RPC) consiste nell'attivazione, da parte di un programma, di una procedura o subroutine attivata su un elaboratore diverso da quello sul quale il programma viene eseguito. Quindi l'RPC consente a un programma di eseguire subroutine «a distanza» su elaboratori remoti, accessibili attraverso una rete. Essenziale al concetto di RPC è l'idea di trasparenza: la chiamata di procedura remota

[2] Ad es., se fruitore ed erogatore computano nell'ordine dei secondi, la risposta potrebbe arrivare dopo minuti od ore, quindi significativamente più tardi.

deve essere infatti eseguita in modo il più possibile analogo a quello della chiamata di procedura locale; i dettagli della comunicazione su rete devono essere «nascosti» (resi trasparenti) all'utilizzatore del meccanismo. Se il linguaggio è orientato agli oggetti, l'invocazione della procedura remote è in realtà l'invocazione di un metodo su un oggetto remoto, e si parla di Remote Method Invocation - RMI.

RPC/RMI è il meccanismo base con cui realizzare una interazione bloccante.

Façade

È uno schema di organizzazione dei moduli in cui uno, detto appunto façade, maschera l'accesso ad un insieme di moduli sottostanti, ad esempio limitando l'accesso a determinate funzionalità tramite un meccanismo di gestione degli accessi, oppure nascondendo le complessità nell'organizzazione e gestione dei moduli sottostanti.

Idempotenza

Il concetto di idempotenza in matematica è una proprietà delle funzioni per la quale applicando molteplici volte una funzione data, il risultato ottenuto è uguale a quello derivante dall'applicazione della funzione un'unica volta (es. gli operatori di intersezione o unione). Applicato alle interfacce di servizio, questo concetto indica il fatto che una operazione, se eseguita più volte non comporta un risultato diverso sul sistema erogatore. Il caso classico è quello in cui si ha una operazione di creazione. Nel caso di errore di rete, l'operazione potrebbe essere eseguita senza che il fruitore riceva un messaggio di risposta. In questo caso il fruitore può ritentare la stessa operazione, ma il risultato in questo caso non deve essere la creazione di una seconda risorsa. L'erogatore dell'interfaccia di servizio deve invece riconoscere la duplicazione della richiesta ed evitare comportamenti indesiderati. Questo comportamento è solitamente ottenuto tramite l'utilizzo di correlation ID oppure tramite il confronto dati basato su dati che fungono da chiave.

Orchestrazione e coreografia

Per orchestrazione si intende un flusso di esecuzione che coinvolge diverse chiamate a servizi secondo regole prestabilite (ad es., un workflow) al fine di ottenere un servizio composto.

La coreografia dei servizi è una forma di specifica della composizione dei servizi in cui il protocollo di interazione tra i diversi servizi componenti è definito da una prospettiva globale. Cioè, in fase di esecuzione della coreografia, ogni partecipante esegue la sua parte (cioè il suo *ruolo*) in base al comportamento degli altri partecipanti. Il ruolo specifica il comportamento, in termini di scambi di messaggi attesi dai partecipanti, che riproducono il ruolo appunto in termini di sequenziamento e tempistica dei messaggi che possono consumare e produrre. La specifica dei messaggi implica anche la descrizione dei dati scambiati tra due o più partecipanti.

La differenza tra i due approcci è meglio compresa attraverso il loro confronto. Da una parte, nelle coreografie, la logica delle interazioni basate sui messaggi tra i partecipanti è specificata da una prospettiva globale. Nell'orchestrazione dei servizi, d'altra parte, la logica viene specificata dal punto di vista locale di un singolo partecipante, chiamato l'orchestratore. Nel linguaggio di orchestrazione BPEL, ad esempio, la specifica dell'orchestrazione del servizio (ad esempio il file del processo BPEL) può essere distribuita sull'infrastruttura (ad esempio un motore di esecuzione BPEL come Apache ODE), e questo costituisce l'implementazione del servizio composto.

In un certo senso, le coreografie e le orchestrazioni sono due facce della stessa medaglia. Da un lato, i ruoli di una coreografia possono essere estratti come orchestrazioni attraverso un processo chiamato *proiezione*; attraverso la proiezione, è possibile realizzare scheletri, ovvero orchestrazioni di servizi incomplete che possono essere utilizzate come linee di base per realizzare i servizi web che partecipano alla coreografia di servizio. D'altra parte, le orchestrazioni già esistenti possono essere composte in coreografie.

Chris Peltz (2003): Web Services Orchestration and Choreography. IEEE Computer 36(10):46-52

Classificazione delle interazioni in A2A, A2C e A2B

A2A : Administration-to-Administration

A2C : Administration-to-Citizen

A2B : Administration-to-Business

In contesti di digital government, è possibile classificare le interazioni tra componenti applicativi in base al soggetto organizzativo sotto la cui responsabilità e nel cui dominio viene eseguito il componente. Si parla di Administration-to-Administration quando sia il componente servente (ad esempio l'interfaccia di servizio) che quello cliente (ad esempio, applicazione Web, applicazione mobile, un altro servizio composto che utilizza il primo come componente all'interno della propria orchestrazione, ecc.) sono nel dominio delle amministrazioni (che probabilmente saranno differenti, ma non necessariamente). Si parla di Administration-to-Citizen quando servente e cliente sono uno nel dominio dell'amministrazione e l'altro su dispositivi del privato cittadino, mentre Administration-to-Business quando servente e cliente sono uno nel dominio dell'amministrazione e l'altro di un'organizzazione privata (azienda, concessionario privato di servizi pubblici, ecc.). La distinzione è utile non tanto dal punto di vista funzionale, ma degli aspetti non funzionali, ad esempio legati al trust, alla reciprocità ed ai livelli di sicurezza che devono essere instaurati nei vari casi.

Impedance mismatch

Derivato dall"*impedance mismatch* dell'elettrotecnica, si riferisce alle difficoltà concettuali e tecniche che si incontrano spesso quando due paradigmi differenti, spesso implicati da altrettante tecnologie, devono coesistere e mapparsi uno sull'altro durante la progettazione e realizzazione di un sistema.

Il più famoso caso di impedance mismatch è quello dell'object-to-relational, noto metaforicamente anche come il Vietnam dell'informatica[4], che si verifica quando un sistema di gestione di database relazionali (RDBMS) è servito da un programma applicativo (o da più programmi applicativi) scritto in un linguaggio di programmazione orientato agli oggetti, in particolare perché gli oggetti o le definizioni di classe devono essere associati a tabelle di database definite da uno schema relazionale. Nel ModI ci sono casi di impedance mismatch quando un'interfaccia di servizio progettata secondo lo stile RPC-like deve essere realizzata in REST.

4.4 Riferimenti Bibliografici

1. Mike P. Papazoglou (2013): Web Services and SOA: Principles and Technology (2nd edition). ISBN: 9780273732167
2. Thomas Erl, Benjamin Carlyle, Cesare Pautasso, Raj Balasubramanian (2013): SOA with REST - Principles, Patterns and Constraints for Building Enterprise Solutions with REST. ISBN 9780137012510
3. Leonard Richardson, Sam Ruby, Mike Amundsen (2013): RESTful Web APIs. ISBN: 9781449359713.

[4] Cf. http://blogs.tedneward.com/post/the-vietnam-of-computer-science/

CAPITOLO 5

Sicurezza

Il presente documento descrive i *profili* di sicurezza individuati da AgID che i soggetti erogatori DEVONO utilizzare per soddisfare le necessità espresse attraverso requisiti funzionali e non funzionali.

Data la variabilità nel tempo delle esigenze delle amministrazioni e delle tecnologie abilitanti, nonché considerata la natura incremenmtale del ModI, l'elenco dei *profili* di sicurezza non è da intendersi esaustivo.

Nel caso in cui un'amministrazione abbia esigenze non ricoperte nei seguenti profili DEVE informare AGID e PUO" utilizzare soluzioni differenti da quelle prospettate, ma queste DEVONO garantire il medesimo o superiore livello di sicurezza.

Nota: I *profili* individuati, coprono gli aspetti di comunicazione "sicura" tra i domini delle singole parti. Le parti mantengono la loro autonomia negli aspetti organizzativi e di sicurezza interni al proprio dominio.

I profili di sicurezza

- definiscono a livello di specifica tecnologica uno «strumento condiviso» utile a favorire l'interoperabilità tra le pubbliche amministrazioni, cittadini ed imprese.
- forniscono un comune linguaggio per fruitori ed erogatori utile a trattare le necessità e le caratteristiche delle *interfacce di servizio*.
- offrono agli sviluppatori le modalità tecniche supportate da standard tecnologici documentati, revisionati e testati per esporre i *servizi digitali*.

I profili affrontano il tema della sicurezza su due livelli differenti:

- **Canale**: definisce le modalità di trasporto dei messaggi tra i confini dei *domini* delle entità coinvolte.
- **Messaggio**: definisce le modalità di comunicazione dei messaggi tra componenti interne dei *domini* delle entità coinvolte.

Ogni profilo è strutturato come segue:

- **Scenario:** definizione dei requisiti funzionali e non funzionali soddisfatti dall'implementazione del profilo.
- **Descrizione:** rappresentazione in linguaggio naturale del profilo con relativi precondizioni e obiettivi.
- **Dettaglio:**
  - **Regole di processamento:** elenco dei passi da eseguire per implementare il profilo.
  - **Tracciato:** ove presente, fornisce un esempio dei messaggi prodotti nell'interazione.

I soggetti interessati, a seguito dell'**analisi dei requisiti** realizzata internamente, per individuare le proprie esigenze funzionali e non funzionali, DOVREBBERO:

1. individuare tra i *profili di interazione* quelli che soddisfano le proprie esigenze;
2. individuare tra i *profili di sicurezza* quelli che soddisfano le proprie esigenze;
3. implementare le *interfacce di servizio* attraverso la combinazione dei *profili di interazione* e di *profili di sicurezza*.

L'individuazione dei *profili* DEVE ricoprire **solamente** i requisiti necessari, inoltre la scelta dei profili da implementare risulta necessaria ove l'ente erogatore del servizio non disponga già di tecnologie che garantiscano i requisiti richiesti.

Trust

Il Trust è uno dei mezzi più importanti per gestire le problematiche di sicurezza nello scambio di informazione in rete per consentire l'interoperabilità tra i sistemi. Esso si basa sul reciproco riconoscimento delle entità interagenti e sulla fiducia nei rispettivi comportamenti.

Nel presente documento, per *direct trust,* si intende la relazione di fiducia tra fruitore ed erogatore, stabilita in modalità diretta, attraverso accordi che si basano sulla condivisione del reciproco *modus operandi*.

Tipologia di profili di sicurezza

I profili di sicurezza del presente documento sono suddivisi in due tipologie:

- **Autenticazione Intradominio [IDA]**

Il processo di identificazione avviene all'interno del dominio di una delle due parti (fruitore ed erogatore) del *direct trust*.

- **Autenticazione Extradominio [EDA]**

Il processo di identificazione avviene mediante una terza parte esterna alle due parti (fruitore ed erogatore) interessate.

Modalità di combinazione dei profili

I profili sono rappresentati con una sequenza di 6 caratteri:

- i primi 3 caratteri indicano la tipologia dell'entità che comunicano:
  - IDA -> Autenticazione Intradominio
  - EDA -> Autenticazione Extradominio
- il quarto carattere definisce l'ambito in cui viene gestita la problematica di sicurezza:
  - C: Ambito di canale ovvero livello trasporto
  - S: Ambito applicativo basato su tecnologia SOAP
  - R: Ambito applicativo basato su tecnologia REST
- Gli ultimi 2 caratteri identificano univocamente il profilo.

Partendo dai requisiti di sicurezza emersi durante la fase di analisi è possibile individuare il **profilo** di sicurezza o una **combinazione di più profili** di sicurezza che ricoprano le esigenze.

Di seguito viene mostrata una rappresentazione dei profili attualmente presenti nel documento in cui vengono evidenziate le relazioni.

Fig. 5.1: Rappresentazione grafica delle dipendenze tra i profili

NOTA BENE

Le parole chiave «DEVE», «DEVONO», «NON DEVE», «NON DEVONO», «E' RICHIESTO», «DOVREBBE», «NON DOVREBBE», «RACCOMANDATO», «NON RACCOMANDATO» «PUO'» e «OPZIONALE» nel testo

del documento debbono essere interpretate come descritto nel seguito, in conformità alle corrispondenti traduzioni contenute nel documento IETF **RFC 2119**[322].

5.1 Sicurezza di canale e/o identificazione delle organizzazioni

5.1.1 [IDAC01] Direct Trust Transport-Level Security

Scenario

Comunicazione tra fruitore ed erogatore che assicuri, a livello di canale:

- confidenzialità
- integrità
- identificazione dell'erogatore, quale organizzazione
- difesa dalle minacce derivanti dagli attacchi: Replay Attack e Spoofing

Descrizione

Il presente profilo assume l'esistenza di un trust tra fruitore (client) ed erogatore (server), che permette il riconoscimento del certificato X.509, o la CA emittente dell'erogatore, così come previsto dal protocollo Transport Layer Security [5] [6].

La sequenza dei messaggi di richiesta/risposta avviene dopo aver instaurato il canale di trasmissione sicuro.

Dettaglio

Fig. 5.2: Sicurezza di canale e/o Autenticazione dell'Erogatore

Regole di processamento

Il canale sicuro tra erogatore e fruitore viene instaurato utilizzando il protocollo TLS, secondo le modalità specificate alla sezione Elenco degli algoritmi.

A: Richiesta

1. Il fruitore costruisce un messaggio di richiesta.
2. Il fruitore spedisce sul canale sicuro stabilito il messaggio di richiesta all'interfaccia di servizio dell'erogatore.

B: Risposta

3. L'erogatore elabora il messaggio e restituisce il risultato.

Come indicato in **RFC 5246**[323] l'impiego del protocollo TLS garantisce a **livello di canale**:

- l'autenticazione dell'erogatore identificato mediante il certificato X.509
- la confidenzialità dei dati scambiati
- l'integrità dei dati scambiati

L'impiego del protocollo TLS 1.2 o maggiore, mitiga il rischio di:

- Replay Attack

[322] https://tools.ietf.org/html/rfc2119.html
[323] https://tools.ietf.org/html/rfc5246.html

- Spoofing

5.1.2 [IDAC02] Direct Trust mutual Transport-Level Security

Scenario

Comunicazione tra fruitore ed erogatore che assicuri a livello di canale:

- confidenzialità
- integrità
- identificazione dell'erogatore e del fruitore, quale organizzazioni
- difesa dalle minacce derivanti dagli attacchi: Replay Attack e Spoofing

Descrizione

Il presente profilo assume l'esistenza di un trust tra fruitore (client) ed erogatore (server), che permette il riconoscimento da entrambe le parti dei certificati X.509, o le CA emittenti, così come previsto dal protocollo Transport Layer Security [5] [6].

La sequenza dei messaggi di richiesta/risposta avviene dopo aver instaurato il canale di trasmissione sicuro.

Dettaglio

Fig. 5.3: Sicurezza di canale e/o Autenticazione delle organizzazioni

Regole di processamento

Il canale sicuro tra erogatore e fruitore viene instaurato in mutua autenticazione utilizzando il protocollo TLS, secondo le modalità specificate alla sezione Elenco degli algoritmi.

A: Richiesta

1. Il fruitore costruisce un messaggio di richiesta.
2. Il fruitore spedisce utilizzando canale sicuro stabilito con il il messaggio di richiesta all'interfaccia di servizio dell'erogatore.

B: Risposta

3. L'erogatore elabora il messaggio e restituisce un risultato.

Come indicato in **RFC 5246**[324] l'impiego del protocollo TLS garantisce a **livello di canale**:

- l'autenticazione di erogatore e fruitore identificati mediante certificati X.509
- la confidenzialità dei dati scambiati
- l'integrità dei dati scambiati

L'impiego del protocollo TLS 1.2 o maggiore, mitiga il rischio di:

- Replay Attack
- Spoofing

[324] https://tools.ietf.org/html/rfc5246.html

5.2 Accesso del soggetto fruitore

5.2.1 [IDAS01] Direct Trust con certificato X.509 su SOAP

Scenario

Comunicazione tra fruitore ed erogatore che assicuri a livello di messaggio:

- accesso del soggetto fruitore, quale organizzazione o unità organizzativa fruitrice, o entrambe le parti;

Descrizione

Il presente profilo specializza lo standard OASIS Web Services Security X.509 Certificate Token Profile Versione 1.1.1 [4].

Si assume l'esistenza di un trust tra fruitore ed erogatore, che permette il riconoscimento da parte dell'erogatore del certificato X.509, o la CA emittente.

Il meccanismo con cui è stabilito il trust, inclusa la modalità di scambio dei certificati X.509) non condiziona il presente profilo.

Il fruitore inoltra un messaggio all'interfaccia di servizio dell'erogatore includendo o referenziando il certificato X.509 e una porzione significativa del messaggio firmata.

L'erogatore, ricevuto il messaggio, verifica il certificato X.509 e valida la porzione firmata del messaggio. Se la verifica e la validazione sono superate, l'erogatore consuma la richiesta e produce la relativa risposta.

Dettaglio

Fig. 5.4: Accesso del Fruitore

Regole di processamento

A: Richiesta

1. Il fruitore costruisce un messaggio SOAP per il servizio.
2. Il fruitore aggiunge al messaggio l'header `WS-Addressing` e l'elemento `<wsu:Timestamp>` composto dagli elementi `<wsu:Created>` e `<wsu:Expires>`
3. Il fruitore calcola la firma per gli elementi significativi del messaggio, in particolare `<wsu:Timestamp>` e `<wsa:To>` del blocco `WS-Addressing`. Il digest è firmato usando la chiave privata associata al certificato X.509 del fruitore. L'elemento `<Signature>` è posizionato nell'header `<Security>` del messaggio.
4. Il fruitore referenzia il certificato X.509 usando in maniera alternativa, nell'header `<Security>`, i seguenti elementi previsti nella specifica ws-security:
   (a) `<wsse:BinarySecurityToken>`
   (b) `<wsse:KeyIdentifier>`
   (c) `<wsse:SecurityTokenReference>`
5. Il fruitore spedisce il messaggio all'interfaccia di servizio dell'erogatore.

B: Risposta

6. L'erogatore verifica il contenuto dell'elemento `<wsu:Timestamp>` nell'header del messaggio al fine di verificare la validità temporale del messaggio.

7. L'erogatore verifica la corrispondenza tra se stesso e quanto definito nell'elemento `<wsa:To>` del blocco WS-Addressing.
8. L'erogatore recupera il certificato X.509 referenziato nell'header `<Security>`.
9. L'erogatore verifica il certificato secondo i criteri del trust.
10. L'erogatore valida l'elemento <Signature> nell'header `<Security>`.
11. L'erogatore garantisce l'accesso al fruitore.
12. Se le azioni da 6 a 11 hanno avuto esito positivo, il messaggio viene elaborato e viene restituito il risultato del servizio richiamato.

Note:

- Per quanto riguarda gli algoritmi da utilizzare nell'elemento `<Signature>` rispettivamente `<DigestMethod>`, `<SignatureMethod>` e `<CanonicalizationMethod>` si fa riferimento agli algoritmi indicati alla sezione Elenco degli algoritmi,
- Un meccanismo simile può essere utilizzato specularmente per l'erogatore.

Tracciato

Di seguito è riportato un tracciato del messaggio inoltrato dal fruitore all'interfaccia di servizio dell'erogatoe relativo ad un servizio di `echo`.

I namespace utilizzati nel tracciato sono riportati di seguito:

```
soap="http://www.w3.org/2003/05/soap-envelope"
wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.
↪0.xsd"
wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.
↪0.xsd"
ds="http://www.w3.org/2000/09/xmldsig#"
ec="http://www.w3.org/2001/10/xml-exc-c14n#"
"http://www.w3.org/2005/08/addressing"
```

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
   <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
↪wss-wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
↪oasis-200401-wss-wssecurity-utility-1.0.xsd" soap:mustUnderstand="1">
     <wsse:BinarySecurityToken EncodingType="http://docs.oasis-open.org/wss/2004/
↪01/oasis-200401-wss-soap-message-security-1.0#Base64Binary" ValueType="http://
↪docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"␣
↪wsu:Id="X509-39011475-65d5-446e-ba38-be84220fd720">
↪MIICqDCCAZCgAwIBAgIEXLSSUTANBgkqhkiG9w0BAQsFADAW...</wsse:BinarySecurityToken>
     <wsu:Timestamp wsu:Id="TS-819df7b7-379d-48f7-8d9c-28c5b5d252f0">
       <wsu:Created>2019-04-15T14:53:34.649Z</wsu:Created>
       <wsu:Expires>2019-04-15T14:58:34.649Z</wsu:Expires>
     </wsu:Timestamp>
     <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="SIG-6e09e972-
↪cbe6-43fc-a10c-38e6dce56dbe">
       <ds:SignedInfo>
         <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-
↪c14n#">
           <ec:InclusiveNamespaces xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n
↪#" PrefixList="soap"/>
         </ds:CanonicalizationMethod>
         <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-
↪sha256"/>
         <ds:Reference URI="#TS-819df7b7-379d-48f7-8d9c-28c5b5d252f0">
```

(continues on next page)

(continua dalla pagina precedente)

```
          <ds:Transforms>
            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
              <ec:InclusiveNamespaces xmlns:ec="http://www.w3.org/2001/10/xml-exc-
↪c14n#" PrefixList="soap wsse"/>
            </ds:Transform>
          </ds:Transforms>
          <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
          <ds:DigestValue>K/3Fq1fYjG5PXv8UlKBuT4XBCWudGR5w2M10wPcZ/Yo=</
↪ds:DigestValue>
        </ds:Reference>
        <ds:Reference URI="#id-96f9b013-17e5-489d-8068-52c3f1345c75">
          <ds:Transforms>
            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
              <ec:InclusiveNamespaces xmlns:ec="http://www.w3.org/2001/10/xml-exc-
↪c14n#" PrefixList="soap"/>
            </ds:Transform>
          </ds:Transforms>
          <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
          <ds:DigestValue>eH3Vlc3l19NbBawDOuFDN11BfmbgGAnl6Z4LpJVM3UM=</
↪ds:DigestValue>
        </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue>jAtZqkfRcFJW+jx9YDv+r2Q8V4IWEWLAZckZlWsmo...</
↪ds:SignatureValue>
      <ds:KeyInfo Id="KI-32484d1e-867e-4465-a96f-52a8668d5a0c">
        <wsse:SecurityTokenReference xmlns:wsse="http://docs.oasis-open.org/wss/
↪2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-
↪open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" wsu:Id="STR-
↪3cf69cce-c56f-461a-905d-dfc20ab0742c">
          <wsse:Reference URI="#X509-39011475-65d5-446e-ba38-be84220fd720"␣
↪ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-
↪profile-1.0#X509v3"/>
        </wsse:SecurityTokenReference>
      </ds:KeyInfo>
    </ds:Signature>
  </wsse:Security>
  <Action xmlns="http://www.w3.org/2005/08/addressing">http://profile.security.
↪modi.agid.org/HelloWorld/sayHi</Action>
  <MessageID xmlns="http://www.w3.org/2005/08/addressing">urn:uuid:55e6bc57-2286-
↪4b7d-82a9-fdbcf57721b1</MessageID>
  <To xmlns="http://www.w3.org/2005/08/addressing" xmlns:wsu="http://docs.oasis-
↪open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" wsu:Id="id-
↪96f9b013-17e5-489d-8068-52c3f1345c75">https://api.amministrazioneesempio.it/soap/
↪echo/v1</To>
  <ReplyTo xmlns="http://www.w3.org/2005/08/addressing">
    <Address>http://www.w3.org/2005/08/addressing/anonymous</Address>
  </ReplyTo>
 </soap:Header>
 <soap:Body>
  <ns2:sayHi xmlns:ns2="http://profile.security.modi.agid.org/">
    <arg0>OK</arg0>
  </ns2:sayHi>
 </soap:Body>
</soap:Envelope>
```

Il tracciato rispecchia le seguenti scelte implementative esemplificative:

- riferimento al security token (`BinarySecurityToken`)
- algoritmi di canonizzazione (`CanonicalizationMethod`)
- algoritmi di firma (`SignatureMethod`).

- algoritmo per il digest (`DigestMethod`)

Le parti, in base alle proprie esigenze, usano gli algoritmi indicati in Elenco degli algoritmi , nonché la modalità di inclusione o referenziazione del certificato X.509.

5.2.2 [IDAS02] Direct Trust con certificato X.509 su SOAP con con unicità del token/messaggio

Scenario

Il seguente profilo estende il profilo IDAS01.

Comunicazione tra fruitore ed erogatore che assicuri a livello di messaggio:

- accesso del soggetto fruitore, quale organizzazione o unità organizzativa fruitore, o entrambe le parti;
- difesa dalle minacce derivanti dagli attacchi: Replay Attack

Descrizione

Il presente profilo specializza lo standard OASIS Web Services Security X.509 Certificate Token Profile Versione 1.1.1 [4].

Si assume l'esistenza di un trust tra fruitore ed erogatore, che permette il riconoscimento da parte dell'erogatore del certificato X.509, o la CA emittente.

Il meccanismo con cui è stabilito il trust, inclusa la modalità di scambio dei certificati X.509, non condiziona il presente profilo.

Il fruitore inoltra un messaggio all'interfaccia di servizio dell'erogatore includendo o referenziando il certificato X.509 e assicurando la firma dei claim del messaggio.

L'erogatore, ricevuto il messaggio, verifica il certificato X.509, valida la firma dei claim ed garantisce l'accesso al fruitore. Se la verifica e la validazione sono superate, l'erogatore consuma la richiesta e produce la relativa risposta.

Dettaglio

Fig. 5.5: Accesso del Fruitore

Regole di processamento

A: Richiesta

1. Il fruitore costruisce un messaggio SOAP per il servizio.
2. Il fruitore aggiunge al messaggio l'header `WS-Addressing` e l'elemento `<wsu:Timestamp>` composto dagli elementi `<wsu:Created>` e `<wsu:Expires>`
3. Il fruitore calcola la firma per gli elementi significativi del messaggio, in particolare `<wsa:To>` e `<wsa:MessageID>` del blocco `WS-Addressing` e `<wsu:Timestamp>`. Il digest è firmato usando la chiave privata associata al certificato X.509 del fruitore. L'elemento `<Signature>` è posizionato nell'header `<Security>` del messaggio.
4. Il fruitore referenzia il certificato X.509 usando in maniera alternativa, nell'header `<Security>`, i seguenti elementi previsti nella specifica ws-security:
   (a) `<wsse:BinarySecurityToken>`
   (b) `<wsse:KeyIdentifier>`

(c) <wsse:SecurityTokenReference>

5. Il fruitore spedisce il messaggio all'interfaccia di servizio dell'erogatore.

B: Risposta

6. L'erogatore verifica il contenuto dell'elemento <wsu:Timestamp> nell'header del messaggio al fine di verificare la validità temporale del messaggio.
7. L'erogatore verifica la corrispondenza tra se stesso e quanto definito nell'elemento <wsa:To> del blocco WS-Addressing.
8. L'erogatore verifica l'univocità del <wsa:MessageID> del blocco WS-Addressing
9. L'erogatore recupera il certificato X.509 referenziato nell'header <Security>.
10. L'erogatore verifica il certificato secondo i criteri del trust.
11. L'erogatore valida l'elemento <Signature> nell'header <Security>.
12. L'erogatore garantisce l'accesso al fruitore.
13. Se le azioni da 6 a 12 hanno avuto esito positivo, il messaggio viene elaborato e viene restituito il risultato del servizio richiamato.

Note:

- Per quanto riguarda gli algoritmi da utilizzare nell'elemento <Signature> rispettivamente <DigestMethod>, <SignatureMethod> e <CanonicalizationMethod> si fa riferimento agli algoritmi indicati alla sezione Elenco degli algoritmi,
- Un meccanismo simile può essere utilizzato specularmente per l'erogatore.

Tracciato

Di seguito è riportato un tracciato del messaggio inoltrato dal fruitore all'interfaccia di servizio dell'erogatore relativo ad un servizio di echo.

I namespace utilizzati nel tracciato sono riportati di seguito:

```
soap="http://schemas.xmlsoap.org/soap/envelope/"
wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.
↪0.xsd"
wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.
↪0.xsd"
ds="http://www.w3.org/2000/09/xmldsig#"
ec="http://www.w3.org/2001/10/xml-exc-c14n#"
"http://www.w3.org/2005/08/addressing"
```

```
 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Header>
  <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
↪wss-wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
↪oasis-200401-wss-wssecurity-utility-1.0.xsd" soap:mustUnderstand="1">
    <wsse:BinarySecurityToken EncodingType="http://docs.oasis-open.org/wss/2004/01/
↪oasis-200401-wss-soap-message-security-1.0#Base64Binary" ValueType="http://docs.
↪oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"␣
↪wsu:Id="X509-bf881daf-371a-4d18-9502-d9f92af9a949">
↪MIICqDCCAZCgAwIBAgIEXLSSUTANBgkqhkiG9w0BAQsFADAWMRQwEgYDVQQDDAttb2RpU2VjUHJvZjAeFw0xOTA0MTUxNDE
↪LxaIZAxSdxJpVNWg/
↪profO+xKz0B6QHK+I0yecHg7TtI4es9AuDyR4pKslpcXyMEqJQ7m5N8v2e4WldeHF2SRN/
↪ereEOuewEi15c7akh4TdkGdiwOSif2AXIugHRgdpHjH86iJxFu24IJmBA7C5tytz7mfKollGhI9+2d0902ayVshCV4/
↪pmnX0pDiGayV1C6SDPTbapXKXJrp1+fBHaUkDY+W/
↪2Q9sC4o8pttmcpHeMRxFDkwIDAQABMA0GCSqGSIb3DQEBCwUAA4IBAQALwKbIm8S2BpYpHaqMwJLeWBPCaDeT7J+KDj39Ac
↪hGM+Hn1mq2ssYqu5JTvuAQ9o8v3UpcIct15RPgOKYfBzxnH1h2vCavpiFCFTc6UoQgPBZGyyNOOKNOxEnXtW7ff1gl2GRLI
↪RyRLf0o03lwCRhAyrU7ya9IMYgrxgjEos2fHB2IGJJ1Wh+gTQWMP+wJymlC0qyjTHx5pyZOzJGtH5HnaVU7EgtxdBRC9dTl
↪am5cZJZrkIHRyfxqkA2W</wsse:BinarySecurityToken>
```

(continues on next page)

(continua dalla pagina precedente)

```
    <wsu:Timestamp wsu:Id="TS-09f1357c-beb4-4804-9410-76c5a06e2e48">
      <wsu:Created>2019-04-15T15:02:15.515Z</wsu:Created>
      <wsu:Expires>2019-04-15T15:07:15.515Z</wsu:Expires>
    </wsu:Timestamp>
    <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="SIG-4d949c5b-
↪968b-4fd5-be67-4cd1d1a41ce3">
      <ds:SignedInfo>
        <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-
↪c14n#">
          <ec:InclusiveNamespaces xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#
↪" PrefixList="soap"/>
        </ds:CanonicalizationMethod>
        <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-
↪sha256"/>
        <ds:Reference URI="#TS-09f1357c-beb4-4804-9410-76c5a06e2e48">
          <ds:Transforms>
            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
              <ec:InclusiveNamespaces xmlns:ec="http://www.w3.org/2001/10/xml-exc-
↪c14n#" PrefixList="soap wsse"/>
            </ds:Transform>
          </ds:Transforms>
          <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
          <ds:DigestValue>HPYjNXdxIuJIWk1EArE+8PIgyWt5nAD+upwcjOSDB20=</
↪ds:DigestValue>
        </ds:Reference>
        <ds:Reference URI="#id-27c23bc8-0c4f-4d98-b046-6e590ea9661b">
          <ds:Transforms>
            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
              <ec:InclusiveNamespaces xmlns:ec="http://www.w3.org/2001/10/xml-exc-
↪c14n#" PrefixList="soap"/>
            </ds:Transform>
          </ds:Transforms>
          <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
          <ds:DigestValue>MJzRD4ZRMsFOxskbnfNV9BnDTCLxuLSnmZ8I4IjaxHw=</
↪ds:DigestValue>
        </ds:Reference>
        <ds:Reference URI="#id-fb4c1fa0-e804-4169-b70e-5b55c5f9d912">
          <ds:Transforms>
            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
              <ec:InclusiveNamespaces xmlns:ec="http://www.w3.org/2001/10/xml-exc-
↪c14n#" PrefixList="soap"/>
            </ds:Transform>
          </ds:Transforms>
          <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
          <ds:DigestValue>MIi+ovLTqYu1HqxUtmUnuhVdMmNKOpOX8vn/fKjvQFU=</
↪ds:DigestValue>
        </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue>SBYs6aikHbfsHHV04ifV/
↪ljVTysxNLRTPU6gsOGJamWGYLMPqOETjBf+NFJhPDVdolQSSHw0SD7uA/RlYkE9amRH1K+hoaUIa/
↪PEhPgC1io/LqZdi3rt+b8uRlk+CXcUKOObgf/N960F/sM6s0ArKQxg/Yx6pqWamXBXo0PH/
↪1FvHSgwdA62s0+Sli96qY0EnJPoyKIrqzskiscLXI1jCe8sesyA+xtJ0qBdFKAn2af48sVStPFv4gizC8+bsCRpQ36ihUIl
↪rC4PheExO7HvSNTpBFdQt+Wr9wAb3oHq4urRBdugA6mX2xaJ8/XyZVajivvuVTw==</
↪ds:SignatureValue>
      <ds:KeyInfo Id="KI-dab2ce54-b000-439a-bcc2-9b8249626a1c">
        <wsse:SecurityTokenReference xmlns:wsse="http://docs.oasis-open.org/wss/
↪2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-
↪open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" wsu:Id="STR-
↪068909fe-1a64-4cf1-bd5a-355a20b0495f">
          <wsse:Reference URI="#X509-bf881daf-371a-4d18-9502-d9f92af9a949"␣
↪ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-
↪profile-1.0#X509v3"/>
```

(continues on next page)

(continua dalla pagina precedente)

```
        </wsse:SecurityTokenReference>
      </ds:KeyInfo>
    </ds:Signature>
  </wsse:Security>
  <Action xmlns="http://www.w3.org/2005/08/addressing">http://profile.security.
↪modi.agid.org/HelloWorld/sayHi</Action>
  <MessageID xmlns="http://www.w3.org/2005/08/addressing" xmlns:wsu="http://docs.
↪oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" wsu:Id=
↪"id-fb4c1fa0-e804-4169-b70e-5b55c5f9d912">urn:uuid:46da4ec1-f962-4f24-8524-
↪48bb74b505d7</MessageID>
  <To xmlns="http://www.w3.org/2005/08/addressing" xmlns:wsu="http://docs.oasis-
↪open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" wsu:Id="id-
↪27c23bc8-0c4f-4d98-b046-6e590ea9661b">http://localhost:8080/security-profile/echo
↪</To>
  <ReplyTo xmlns="http://www.w3.org/2005/08/addressing">
    <Address>http://www.w3.org/2005/08/addressing/anonymous</Address>
  </ReplyTo>
</soap:Header>
<soap:Body>
  <ns2:sayHi xmlns:ns2="http://profile.security.modi.agid.org/">
    <arg0>OK</arg0>
  </ns2:sayHi>
</soap:Body>
</soap:Envelope>
```

Il tracciato rispecchia le seguenti scelte implementative esemplificative:

- riferimento al security token (`BinarySecurityToken`)
- algoritmi di canonizzazione (`CanonicalizationMethod`)
- algoritmi di firma (`SignatureMethod`).
- algoritmo per il digest (`DigestMethod`)

Le parti, in base alle proprie esigenze, usano gli algoritmi indicati in Elenco degli algoritmi , nonché la modalità di inclusione o referenziazione del certificato X.509.

5.2.3 [IDAR01] Direct Trust con certificato X.509 su REST

Scenario

Comunicazione tra fruitore ed erogatore che assicuri a livello di messaggio:

- accesso del soggetto fruitore, quale organizzazione o unità organizzativa fruitore, o entrambe le parti.

Descrizione

Il presente profilo declina l'utilizzo di:

- JSON Web Token (JWT) definita dall' **RFC 7519**[325]
- JSON Web Signature (JWS) definita dall' **RFC 7515**[326]

Si assume l'esistenza di un trust tra fruitore ed erogatore, che permette il riconoscimento da parte dell'erogatore del certificato X.509, o la CA emittente.

Il meccanismo con cui è stabilito il trust, inclusa la modalità di scambio dei certificati X.509, non condiziona il presente profilo.

[325] https://tools.ietf.org/html/rfc7519.html
[326] https://tools.ietf.org/html/rfc7515.html

Il fruitore inoltra un messaggio all'erogatore includendo o referenziando il certificato X.509 e una porzione significativa del messaggio firmata.

L'erogatore, ricevuto il messaggio, verifica il certificato X.509 e valida la porzione firmata del messaggio, inclusa la corrispondenza del destinatario e l'intervallo di validità della firma.

Se la verifica e la validazione sono superate, l'erogatore consuma la richiesta e produce la relativa risposta.

Dettaglio

Fig. 5.6: Accesso del Fruitore

Regole di processamento

A: Richiesta

1. Il fruitore predispone la payload del messaggio (ad esempio un oggetto JSON)
2. Il fruitore costruisce il JWT popolando:
   (a) il Jose Header[327] con almeno i `parameter`:
      - alg[328] con l'algoritmo di firma
      - typ[329] uguale a `JWT`
      - una o più delle seguenti opzioni per referenziare il certificato X.509:
        – x5u[330] (X.509 URL)
        – x5c[331] (X.509 Certificate Chain)
        – x5t#256[332] (X.509 Certificate SHA-256 Thumbprint)

   (b) la payload del JWT coi claim rappresentativi degli elementi chiave del messaggio, contenente almeno:
      - i riferimenti temporali di emissione e scadenza: iat[333] , exp[334]. Se il flusso richiede di verificare l'istante di prima validità del token, si può usare il claim nbf[335].
      - il riferimento dell'erogatore in aud[336]
3. il fruitore firma il token adottando la JWS Compact Serialization[337]
4. il fruitore posiziona il `JWT` nell' **HTTP header Authorization** [338]
5. Il fruitore spedisce il messaggio all'erogatore

B: Risposta

6. L'erogatore decodifica il `JWT` presente in **HTTP header Authorization** [339] e valida i claim contenuti nel Jose Header[340], in particolare verifica:

[327] https://tools.ietf.org/html/rfc7515#section-4
[328] https://tools.ietf.org/html/rfc7515#section-4.1.1
[329] https://tools.ietf.org/html/rfc7519#section-5.1
[330] https://tools.ietf.org/html/rfc7515#section-4.1.5
[331] https://tools.ietf.org/html/rfc7515#section-4.1.6
[332] https://tools.ietf.org/html/rfc7515#section-4.1.8
[333] https://tools.ietf.org/html/rfc7519#section-4.1.6
[334] https://tools.ietf.org/html/rfc7519#section-4.1.4
[335] https://tools.ietf.org/html/rfc7519#section-4.1.5
[336] https://tools.ietf.org/html/rfc7519#section-4.1.3
[337] https://tools.ietf.org/html/rfc7515#section-7.1
[338] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Authorization
[339] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Authorization
[340] https://tools.ietf.org/html/rfc7515#section-4

- il contenuto dei claim iat[341] ed exp[342];
- la corrispondenza tra se stesso e il claim aud[343];

7. L'erogatore recupera il certificato X.509 referenziato nel Jose Header[344]
8. L'erogatore verifica il certificato secondo i criteri del trust
9. L'erogatore valida la firma verificando l'elemento Signature del `JWT`
10. L'erogatore garantisce l'accesso al fruitore
11. Se le azioni da 6 a 10 hanno avuto esito positivo, il messaggio viene elaborato e viene restituito il risultato del servizio richiamato

Note:

- Gli algoritmi da utilizzare in alg[345] sono indicati in Elenco degli algoritmi
- Un meccanismo simile può essere utilizzato specularmente per l'erogatore.
- Per prevenire il rischio di user-enumeration, i messaggi di errore di autenticazione non devono fornire informazioni sull'esistenza o meno dell'utenza.

Tracciato

Di seguito è riportato un tracciato del messaggio inoltrato dal fruitore all'erogatore.

Esempio porzione messaggio HTTP

```
GET https://api.erogatore.org/rest/service/v1/hello/echo/Ciao  HTTP/1.1
Accept: application/json
Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5c.vz8...

.
.
.
```

Esempio porzione JWT

```
# header
{
  "alg": "ES256",
  "typ": "JWT",
  "x5c": [
    "MIICyzCCAbOgAwIBAgIEC..."
  ]
}
# payload
{
 "iat": 1554382877,
 "nbf": 1554382877,
 "exp": 1554382879,
 "aud": "https://api.erogatore.org/rest/service/v1/hello/echo"
}
```

Gli elementi presenti nel tracciato rispettano le seguenti scelte implementative e includono:

- l'intervallo temporale di validità, in modo che il JWT possa essere usato solo tra gli istanti nbf[346] ed exp[347];

[341] https://tools.ietf.org/html/rfc7519#section-4.1.6
[342] https://tools.ietf.org/html/rfc7519#section-4.1.4
[343] https://tools.ietf.org/html/rfc7519#section-4.1.3
[344] https://tools.ietf.org/html/rfc7515#section-4
[345] https://tools.ietf.org/html/rfc7515#section-4.1.1
[346] https://tools.ietf.org/html/rfc7519#section-4.1.5
[347] https://tools.ietf.org/html/rfc7519#section-4.1.4

- indica l'istante iat[348] di emissione del JWT. Se le parti possono accordarsi nel considerarlo come l'istante iniziale di validità del token, **RFC 7519**[349] non assegna a questo claim nessun ruolo specifico nella validazione, a differenza di nbf[350];
- il destinatario del JWT, che deve sempre essere validato;
- contenuto della certificate chain X.509 (x5c[351])
- algoritmi di firma e digest (alg[352]).

Le parti, in base alle proprie esigenze, individuano gli specifici algoritmi secondo quanto indicato alla sezione Elenco degli algoritmi nonché la modalità di inclusione o referenziazione del certificato X.509.

5.2.4 [IDAR02] Direct Trust con certificato X.509 su REST con unicità del token/messaggio

Scenario

Il seguente profilo estende il profilo IDAR01.

Comunicazione tra fruitore ed erogatore che assicuri a livello di messaggio:

- accesso del soggetto fruitore, quale organizzazione o unità organizzativa fruitore, o entrambe le parti
- la difesa dalle minacce derivanti dagli attacchi: Replay Attack quando il JWT o il messaggio non devono essere riprocessati

Descrizione

Il presente profilo declina l'utilizzo di:

- JSON Web Token (JWT) definita dall'**RFC 7519**[353]
- JSON Web Signature (JWS) definita dall'**RFC 7515**[354]

Si assume l'esistenza di un trust tra fruitore (client) ed erogatore (server), che permette il riconoscimento da parte dell'erogatore del certificato X.509, o la CA emittente.

Il meccanismo con cui è stabilito il trust, inclusa la modalità di scambio dei certificati X.509) non condiziona il presente profilo.

Il fruitore inoltra un messaggio all'interfaccia di servizio dell'erogatore includendo o referenziando il certificato X.509 e assicurando la firma dei claim del messaggio.

L'erogatore, ricevuto il messaggio, verifica il certificato X.509 e valida la porzione firmata del messaggio, inclusa la corrispondenza del destinatario e l'intervallo di validità della firma.

L'erogatore verifica inoltre l'univocità dell'identificativo ricevuto nel JWT.

Se la verifica e la validazione sono superate, l'erogatore consuma la richiesta e produce la relativa risposta.

Dettaglio

Fig. 5.7: Accesso del Fruitore

[348] https://tools.ietf.org/html/rfc7519#section-4.1.6
[349] https://tools.ietf.org/html/rfc7519.html
[350] https://tools.ietf.org/html/rfc7519#section-4.1.5
[351] https://tools.ietf.org/html/rfc7515#section-4.1.6
[352] https://tools.ietf.org/html/rfc7515#section-4.1.1
[353] https://tools.ietf.org/html/rfc7519.html
[354] https://tools.ietf.org/html/rfc7515.html

Regole di processamento

A: Richiesta

1. Il fruitore predispone la payload del messaggio (ad esempio un oggetto JSON)
2. Il fruitore costruisce il JWT popolando:
   (a) il Jose Header[355] con almeno i `parameter`:
      - alg[356] con l'algoritmo di firma
      - typ[357] uguale a `JWT`
      - una o più delle seguenti opzioni per referenziare il certificato X.509:
        – x5u[358] (X.509 URL)
        – x5c[359] (X.509 Certificate Chain)
        – x5t#256[360] (X.509 Certificate SHA-256 Thumbprint)

   (b) la payload del JWT coi claim rappresentativi degli elementi chiave del messaggio, contenente almeno:
      - i riferimenti temporali di emissione e scadenza: iat[361] , exp[362]. Se il flusso richiede di verificare l'istante di prima validità del token, si può usare il claim nbf[363].
      - il riferimento dell'erogatore in aud[364];
      - un identificativo univoco del token jti[365]. Se utile alla logica applicativa l'identificativo può essere anche collegato al messaggio.
3. il fruitore firma il token adottando la JWS Compact Serialization[366]
4. il fruitore posiziona il `JWT` nell' **HTTP header Authorization** [367]
5. Il fruitore spedisce il messaggio all'erogatore

B: Risposta

6. L'erogatore decodifica il `JWT` presente in **HTTP header Authorization** [368] e valida i claim contenuti nel Jose Header[369], in particolare verifica:
   - il contenuto dei claim iat[370] ed exp[371];
   - la corrispondenza tra se stesso e il claim aud[372];
   - l'univocità del claim jti[373]
7. L'erogatore recupera il certificato X.509 referenziato nel Jose Header[374]
8. L'erogatore verifica il certificato secondo i criteri del trust

[355] https://tools.ietf.org/html/rfc7515#section-4
[356] https://tools.ietf.org/html/rfc7515#section-4.1.1
[357] https://tools.ietf.org/html/rfc7519#section-5.1
[358] https://tools.ietf.org/html/rfc7515#section-4.1.5
[359] https://tools.ietf.org/html/rfc7515#section-4.1.6
[360] https://tools.ietf.org/html/rfc7515#section-4.1.8
[361] https://tools.ietf.org/html/rfc7519#section-4.1.6
[362] https://tools.ietf.org/html/rfc7519#section-4.1.4
[363] https://tools.ietf.org/html/rfc7519#section-4.1.5
[364] https://tools.ietf.org/html/rfc7519#section-4.1.3
[365] https://tools.ietf.org/html/rfc7519#section-4.1.7
[366] https://tools.ietf.org/html/rfc7515#section-7.1
[367] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Authorization
[368] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Authorization
[369] https://tools.ietf.org/html/rfc7515#section-4
[370] https://tools.ietf.org/html/rfc7519#section-4.1.6
[371] https://tools.ietf.org/html/rfc7519#section-4.1.4
[372] https://tools.ietf.org/html/rfc7519#section-4.1.3
[373] https://tools.ietf.org/html/rfc7519#section-4.1.7
[374] https://tools.ietf.org/html/rfc7515#section-4

9. L'erogatore valida la firma verificando l'elemento Signature del `JWT`
10. L'erogatore garantisce l'accesso al fruitore
11. Se le azioni da 6 a 10 hanno avuto esito positivo, il messaggio viene elaborato e viene restituito il risultato del servizio richiamato

Note:

- Per quanto riguarda gli algoritmi da utilizzare si fa riferimento ad Elenco degli algoritmi.
- Un meccanismo simile può essere utilizzato specularmente per l'erogatore.

Tracciato

Di seguito è riportato un tracciato del messaggio inoltrato dal fruitore all'interfaccia di servizio dell'erogatore.

Esempio porzione pacchetto HTTP

```
GET https://api.erogatore.org/rest/service/v1/hello/echo/Ciao  HTTP/1.1
Accept: application/json
Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5c.vz8...

.
.
```

Esempio porzione JWT

```
# header
{
  "alg": "ES256",
  "typ": "JWT",
  "x5c": [
    "MIICyzCCAbOgAwIBAgIEC..."
  ]
}
# payload
{
  "aud": "https://api.erogatore.org/rest/service/v1/hello/echo"
  "iat": 1516239022,
  "nbf": 1516239022,
  "exp": 1516239024,
  "jti": "065259e8-8696-44d1-84c5-d3ce04c2f40d"
}
```

Gli elementi presenti nel tracciato rispettano le seguenti scelte implementative e includono:

- l'intervallo temporale di validità, in modo che il JWT possa essere usato solo tra gli istanti nbf[375] ed exp[376];
- indica l'istante iat[377] di emissione del JWT. Se le parti possono accordarsi nel considerarlo come l'istante iniziale di validità del token, **RFC 7519**[378] non assegna a questo claim nessun ruolo specifico nella validazione, a differenza di nbf[379];
- il destinatario del JWT, che deve sempre essere validato;
- contenuto della certificate chain X.509 (x5c[380])
- algoritmi di firma e digest (alg[381]).

[375] https://tools.ietf.org/html/rfc7519#section-4.1.5
[376] https://tools.ietf.org/html/rfc7519#section-4.1.4
[377] https://tools.ietf.org/html/rfc7519#section-4.1.6
[378] https://tools.ietf.org/html/rfc7519.html
[379] https://tools.ietf.org/html/rfc7519#section-4.1.5
[380] https://tools.ietf.org/html/rfc7515#section-4.1.6
[381] https://tools.ietf.org/html/rfc7515#section-4.1.1

Le parti, in base alle proprie esigenze, individuano gli specifici algoritmi secondo quanto indicato alla sezione Elenco degli algoritmi nonché la modalità di inclusione o referenziazione del certificato X.509.

5.3 Integrità

5.3.1 [IDAS03] Integrità della payload del messaggio SOAP

Scenario

Il presente profilo estende **IDAS01** o **IDAS02**, aggiungendo alla comunicazione tra fruitore ed erogatore a livello di messaggio:

- Integrità della payload del messaggio.

Descrizione

Il presente profilo specializza lo standard OASIS Web Services Security X.509 Certificate Token Profile Versione 1.1.1 [4].

Si assume l'esistenza di un trust tra fruitore ed erogatore, che permette il riconoscimento da parte dell'erogatore del certificato X.509, o la CA emittente.

Il meccanismo con cui è stabilito il trust non condiziona il presente profilo.

Il fruitore inoltra un messaggio all'interfaccia di servizio dell'erogatore includendo o referenziando il certificato X.509 e la firma della payload del messaggio.

L'erogatore, ricevuto il messaggio, verifica il certificato X.509 e valida l'integrità della payload del messaggio firmato. Se la verifica e la validazione sono superate, l'erogatore consuma la richiesta e produce la relativa risposta.

Dettaglio

Fig. 5.8: Integrità della payload del messaggio

Regole di processamento

A: Richiesta

1. Il fruitore costruisce un messaggio SOAP per il servizio.
2. Il fruitore calcola la firma della payload del messaggio usando l'XML Signature. Il digest è firmato usando la chiave privata associata al certificato X.509 del fruitore. L'elemento `<Signature>` è posizionato nell'header `<Security>` del messaggio.
3. Il fruitore referenzia il certificato X.509 usando in maniera alternativa, nell'header `<Security>`, i seguenti elementi previsti nella specifica ws-security:
   (a) `<wsse:BinarySecurityToken>`
   (b) `<wsse:KeyIdentifier>`
   (c) `<wsse:SecurityTokenReference>`
4. Il fruitore spedisce il messaggio all'interfaccia di servizio dell'erogatore.

B: Risultato

5. L'erogatore recupera il certificato X.509 referenziato nell'header `<Security>`.
6. L'erogatore verifica il certificato secondo i criteri del trust.

7. L'erogatore valida la firma verificando l'elemento <Signature> nell'header `<Security>`.
8. Se il certificato è valido anche per identificare il soggetto fruitore, l'erogatore autentica lo stesso
9. Se le azioni da 5 a 8 hanno avuto esito positivo, il messaggio viene elaborato e viene restituito il risultato del servizio richiamato.

Note:

- Per quanto riguarda gli algoritmi da utilizzare nell'elemento <Signature> rispettivamente `<DigestMethod>` , `<SignatureMethod>` e `<CanonicalizationMethod>` si fa riferimento agli algoritmi indicati alla sezione Elenco degli algoritmi.
- Un meccanismo simile può essere utilizzato per garantire l'integrità della payload del del messaggio risposta dell'erogatore al fruitore.

.._integrita-tracciato-4:

Tracciato

Di seguito è riportato un tracciato del messaggio inoltrato dal fruitore all'interfaccia di servizio dell'erogatore.

I namespace utilizzati nel tracciato sono riportati di seguito:

```
soap="http://schemas.xmlsoap.org/soap/envelope/"
wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.
↪0.xsd"
ds="http://www.w3.org/2000/09/xmldsig#"
ec="http://www.w3.org/2001/10/xml-exc-c14n#"
"http://www.w3.org/2005/08/addressing"
```

```
<soap:Envelope>
  <soap:Header>
    <wsse:Security soap:mustUnderstand="1">
      <wsse:BinarySecurityToken EncodingType="http://docs.oasis-open.org/wss/2004/
↪01/oasis-200401-wss-soap-message-security-1.0#Base64Binary"    ValueType="http://
↪docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3" ␣
↪  wsu:Id="X509-44680ddc-e35a-4374-bcbf-2b6dcba722d7">
↪MIICyzCCAbOgAwIBAgIECxY+9TAhkiG9w...
      </wsse:BinarySecurityToken>
      <ds:Signature Id="SIG-f58c789e-e3d3-4ec3-9ca7-d1e9a4a90f90">
        <ds:SignedInfo>
          <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-
↪c14n#">
            <ec:InclusiveNamespaces PrefixList="soap" />
          </ds:CanonicalizationMethod>
          <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more
↪#rsa-sha256" />
            <ds:Reference URI="#bd-567d101-aed1-789e-81cb-5ae1c5dbef1a">
↪<ds:Transforms>
              <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
                <ec:InclusiveNamespaces PrefixList="soap" />
              </ds:Transform>
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256" />
            <ds:DigestValue>0cJNCJ1W8Agu66fGTXlPRyy0EUNUQ9OViFlm8qf8Ysw=</
↪ds:DigestValue>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>AIrDa7ukDfFJD867goC+c7K3UampxpX/Nj/...</
↪ds:SignatureValue>
        <ds:KeyInfo Id="KI-cad9ee47-dec8-4340-8fa1-74805f7e26f8">
          <wsse:SecurityTokenReference wsu:Id="STR-e193f25f-9727-4197-b7aa-
↪25b01c9f2ba3">
```

(continues on next page)

(continua dalla pagina precedente)

```
          <wsse:Reference URI="#X509-44680ddc-e35a-4374-bcbf-2b6dcba722d7"␣
↪ValueType="http://docs.oasis-open.org/   wss/2004/01/oasis-200401-wss-x509-token-
↪profile-1.0#X509v3"/>          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
    </wsse:Security>
     </soap:Header>
  <soap:Body xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
↪wssecurity-utility-1.0.xsd"    wsu:id="bd-567d101-aed1-789e-81cb-5ae1c5dbef1a">
    <ns2:sayHi xmlns:ns2="http://example.profile.security.modi.agid.gov.it/">
      <arg0>Hello World!</arg0>
    </ns2:sayHi>
  </soap:Body>
</soap:Envelope>
```

Il codice rispecchia alcune scelte implementative esemplificative in merito:

- riferimento al security token (`BinarySecurityToken`)
- algoritmi di canonizzazione (`CanonicalizationMethod`)
- algoritmi di firma (`SignatureMethod`)
- algoritmo per il digest (`DigestMethod`)

Le parti, in base alle proprie esigenze, individuano gli specifici algoritmi secondo quanto indicato alla sezione Elenco degli algoritmi nonché la modalità di inclusione o referenziazione del certificato X.509.

5.3.2 [IDAR03] Integrità della payload messaggio REST

Scenario

Il presente profilo estende **IDAR01** o **IDAR02**, aggiungendo alla comunicazione tra fruitore ed erogatore a livello di messaggio:

- Integrità della payload del messaggio.

Si adottano le indicazione riportate in **RFC 7231**[382].

Considereremo sempre richieste e risposte complete, con i metodi standard definiti in **RFC 7231#section-4**[383].

Questo scenario non copre quindi *Range Requests* **RFC 7233**[384] o **HTTP method PATCH** [385] che trasmette una rappresentazione parziale.

Descrizione

Il presente profilo propone l'utilizzo di:

- semantica HTTP **RFC 7231**[386];
- `Digest` HTTP header **RFC 3230**[387] per l'integrità della rappresentazione della risorsa;
- JSON Web Token (JWT) definita dall' **RFC 7519**[388];
- JSON Web Signature (JWS) definita dall' **RFC 7515**[389].

[382] https://tools.ietf.org/html/rfc7231.html
[383] https://tools.ietf.org/html/rfc7231.html#section-4
[384] https://tools.ietf.org/html/rfc7233.html
[385] https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PATCH
[386] https://tools.ietf.org/html/rfc7231.html
[387] https://tools.ietf.org/html/rfc3230.html
[388] https://tools.ietf.org/html/rfc7519.html
[389] https://tools.ietf.org/html/rfc7515.html

Si assume l'esistenza di un trust tra fruitore ed erogatore, che permette il riconoscimento da parte dell'erogatore del certificato X.509, o la CA emittente.

Il meccanismo con cui è stabilito il trust non condiziona il presente profilo.

Dettaglio

Fig. 5.9: Integrità del messaggio

Regole di processamento

A: Richiesta

1. Il fruitore predispone il body del messaggio (ad esempio un oggetto JSON)
2. Il fruitore calcola il valore del `Digest` header dei representation data[390] secondo le indicazioni in **RFC 3230**[391]
3. Il fruitore individua l'elenco degli HTTP Header da firmare, inclusi `Digest`, **HTTP header Content-Type** [392] e **HTTP header Content-Encoding** [393]
4. Il fruitore crea la struttura o la stringa da firmare in modo che includa gli http header da proteggere, i riferimenti temporali di validità della firma e degli estremi della comunicazione, ovvero:
   (a) il Jose Header[394] con almeno i `parameter`:
      - alg[395] con l'algoritmo di firma
      - typ[396] uguale a `JWT`
      - una o più delle seguenti opzioni per referenziare il certificato X.509:
        – x5u[397] (X.509 URL)
        – x5c[398] (X.509 Certificate Chain)
        – x5t#256[399] (X.509 Certificate SHA-256 Thumbprint)

   (b) i seguenti claim obbligatori:
      - i riferimenti temporali di emissione e scadenza: iat[400] , exp[401]. Se il flusso richiede di verificare l'istante di prima validità del token, si può usare il claim nbf[402].
      - il riferimento dell'erogatore in aud[403];

   (c) i seguenti claim, secondo la logica del servizio:
      - sub[404]: oggetto (*principal* see **RFC 3744#section-2**[405]) dei claim contenuti nel jwt
      - iss[406]: identificativo del mittente

[390] https://tools.ietf.org/html/rfc7231#section-3.2
[391] https://tools.ietf.org/html/rfc3230.html
[392] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type
[393] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Encoding
[394] https://tools.ietf.org/html/rfc7515#section-4
[395] https://tools.ietf.org/html/rfc7515#section-4.1.1
[396] https://tools.ietf.org/html/rfc7519#section-5.1
[397] https://tools.ietf.org/html/rfc7515#section-4.1.5
[398] https://tools.ietf.org/html/rfc7515#section-4.1.6
[399] https://tools.ietf.org/html/rfc7515#section-4.1.8
[400] https://tools.ietf.org/html/rfc7519#section-4.1.6
[401] https://tools.ietf.org/html/rfc7519#section-4.1.4
[402] https://tools.ietf.org/html/rfc7519#section-4.1.5
[403] https://tools.ietf.org/html/rfc7519#section-4.1.3
[404] https://tools.ietf.org/html/rfc7519#section-4.1.2
[405] https://tools.ietf.org/html/rfc3744.html#section-2
[406] https://tools.ietf.org/html/rfc7519#section-4.1.1

- jti[407]: identificativo del JWT, per evitare replay attack

(d) il claim `signed_headers`[419] con gli header http da proteggere ed i rispettivi valori, ovvero:

- `Digest`
- `Content-Type`
- `Content-Encoding`

3. il fruitore firma il token adottando la JWS Compact Serialization[408]
4. il fruitore posiziona il `JWT` nell' `Authorization` header
5. Il fruitore spedisce il messaggio all'erogatore.

B: Risultato

6. L'erogatore decodifica il `JWT` presente in `Authorization` header e valida i claim contenuti nel Jose Header[409], in particolare verifica:
   - il contenuto dei claim iat[410] ed exp[411];
   - la corrispondenza tra se stesso e il claim aud[412];
   - l'univocità del claim jti[413]
7. L'erogatore recupera il certificato X.509 referenziato nel Jose Header[414]
8. L'erogatore verifica il certificato secondo i criteri del trust
9. L'erogatore valida la firma verificando l'elemento Signature del `JWT`
10. L'erogatore verifica la corrispondenza tra i valori degli header passati nel messaggio e quelli presenti nel claim `signed-header`
11. L'erogatore quindi verifica la corrispondenza tra `Digest` ed il payload body ricevuto
12. Se le azioni da 6 a 11 hanno avuto esito positivo, il messaggio viene elaborato e viene restituito il risultato del servizio richiamato.

Note:

- Per gli algoritmi da utilizzare in alg[415] e `Digest` si veda Elenco degli algoritmi
- Un meccanismo simile può essere utilizzato per garantire l'integrità della risposta da parte dell'erogatore al fruitore. In questo caso si ricorda che `Digest` fa" riferimento al checksum del payload body della selected representation[416]. Per una richiesta con **HTTP method HEAD** [417] il server deve ritornare il checksum dell'ipotetico payload body ritornato dalla corrispondente richiesta con **HTTP method GET** [418][420].

Tracciato

Di seguito è riportato un tracciato del messaggio inoltrato dal fruitore all'interfaccia di servizio dell'erogatore.

[407] https://tools.ietf.org/html/rfc7519#section-4.1.7
[419] Il presente documento ha individuato il claim «signed_headers» per contenere l'elenco degli header firmati.
[408] https://tools.ietf.org/html/rfc7515#section-7.1
[409] https://tools.ietf.org/html/rfc7515#section-4
[410] https://tools.ietf.org/html/rfc7519#section-4.1.6
[411] https://tools.ietf.org/html/rfc7519#section-4.1.4
[412] https://tools.ietf.org/html/rfc7519#section-4.1.3
[413] https://tools.ietf.org/html/rfc7519#section-4.1.7
[414] https://tools.ietf.org/html/rfc7515#section-4
[415] https://tools.ietf.org/html/rfc7515#section-4.1.1
[416] https://tools.ietf.org/html/rfc7231#section-3
[417] https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/HEAD
[418] https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET
[420] Per coerenza con **RFC 7231#section-3.1** «In a response to a HEAD request, the representation header fields describe the representation data that would have been enclosed in the payload body if the same request had been a GET.»

Listato 5.1: Richiesta HTTP con *Digest* e representation metadata

```
POST https://api.erogatore.org/rest/service/v1/hello/echo/ HTTP/1.1
Accept: application/json
Agid-JWT-Signature: eyJhbGciOiJSUzI1NiIsInR5c.vz8...
Digest: SHA-256=cFfTOCesrWTLVzxn8fmHl4AcrUs40Lv5D275FmAZ96E=
Content-Type: application/json
Content-Encoding: identity


{"testo": "Ciao mondo"}
```

Listato 5.2: Porzione JWT con campi protetti dalla firma

```
# header
{
  "alg": "ES256",
  "typ": "JWT",
  "x5c": [
    "MIICyzCCAbOgAwIBAgIEC..."
  ]
}
# payload
{
  "aud": "https://api.erogatore.org/rest/service/v1/hello/echo"
  "iat": 1516239022,
  "nbf": 1516239022,
  "exp": 1516239024,
  "signed_headers": [
      {"digest": "SHA-256=cFfTOCesrWTLVzxn8fmHl4AcrUs40Lv5D275FmAZ96E="},
      {"content-type": "application/json"},
      {"content-encoding": "identity"},
  ],
}
```

Il tracciato rispecchia alcune scelte implementative esemplificative in merito:

- include tutti gli elementi del `JWT` utilizzati in **IDAR02**
- mette in `minuscolo` i nomi degli header firmati
- utilizza il claim custom `signed_headers` contenente una lista di json objects per supportare la firma di più header ed eventualmente verificare il loro ordinamento

Le parti, in base alle proprie esigenze, individuano gli specifici algoritmi secondo quanto indicato alla sezione Elenco degli algoritmi nonché la modalità di inclusione o referenziazione del certificato X.509.

5.4 Soluzioni di sicurezza

5.4.1 Soluzione per confidenzialità ed autenticazione del soggetto fruitore

Scenario

Dare seguito ad uno scambio tra fruitore ed erogatore che garantisca:

- la confidenzialità a livello di canale
- l'autenticazione del soggetto Fruitore

Il soggetto fruitore potrebbe non coincidere con l'unità organizzativa fruitore, ma comunque appartenere alla stessa.

Questa soluzione utilizza i seguenti profili:

- **IDAC01**
- **IDAS01** o in alternativa **IDAR01**

Precondizioni

Si assume l'esistenza di un trust tra fruitore ed erogatore che stabilisce:

- riconoscimento da parte dell'erogatore dei certificati X.509, o la CA emittente, relative al soggetto fruitore
- riconoscimento da parte del fruitore del certificato X.509, o la CA emittente, relative al soggetto erogatore

Il meccanismo con cui è stabilito il trust non condiziona quanto descritto di seguito.

Flusso delle interazioni

A: Richiesta

Il messaggio di richiesta viene firmato utilizzando il profilo **IDAS01** nel caso di utilizzo di SOAP o **IDAR01** nel caso di utilizzo di REST, per garantire:

- l'identità del soggetto fruitore

Il fruitore invia il messaggio di richiesta all'interfaccia di servizio dell'erogatore.

Il messaggio viene trasmesso su un canale sicuro utilizzando il profilo **IDAC01** per garantire:

- la confidenzialità a livello di canale.

B: Risposta

L'erogatore da seguito a quanto previsto nel profilo **IDAS01** nel caso di utilizzo di SOAP o **IDAR01** nel caso di utilizzo di REST

5.4.2 Soluzioni per la non ripudiabilità della trasmissione

Scenario

Dare seguito ad uno scambio tra fruitore ed erogatore che garantisca la non ripudiabilità [8] assicurando a livello di messaggio:

- integrità del messaggio
- autenticazione del soggetto fruitore, quale organizzazione o unità organizzativa fruitore quale mittente del contenuto
- conferma da parte dell'erogatore della ricezione del contenuto
- opponibilità ai terzi
- robustezza della trasmissione

Il presente scenario utilizza come modello di comunicazione i Profili di Interazione Bloccante, ed è possibile implementarlo indipendentemente con SOAP e REST.

Questa soluzione utilizza i seguenti profili:

- **IDAC01** o in alternativa **IDAC02**
- per SOAP **IDAS03** ed **IDAS02**
- per REST **IDAR03** ed **IDAR02**

Precondizioni

Si assume l'esistenza di un trust tra fruitore ed erogatore che stabilisce:

- reciproco riconoscimento da parte dell'erogatore e del fruitore dei certificati X.509, o le CA emittenti.

Il meccanismo con cui è stabilito il trust non condiziona quanto descritto nella sezione.

Il fruitore ed erogatore devono concordare:

- un identificativo univoco del messaggio, necessario a garantire il riscontro di ritrasmissioni (vedi **IDAS02** ed **IDAR02**), e le relative modalità di scambio;
- l'arco temporale di persistenza dei messaggi che dipende dalle caratteristiche del contenuto dei dati scambiati e nel rispetto delle norme di legge.
- il tempo di validità della transazione che intercorre tra:
  - l'istante di inoltro del fruitore
  - l'istante di ricezione dell'erogatore;
- il tempo massimo di attesa del fruitore del messaggio di risposta per ritenere la comunicazione non avvenuta;
- il numero massimo di tentativi di rinvio da parte del fruitore accettati dall'erogatore;
- eventuale utilizzo di canali alternativi per superare o evidenziare problemi di comunicazione riscontrati;

Attraverso le tecnologie di criptazione sono garantite le seguenti proprietà:

- integrità e non ripudio del messaggio inviato dal fruitore
- integrità e non ripudio del messaggio di conferma da parte dell'erogatore
- autenticazione del soggetto fruitore
- autenticazione del soggetto erogatore
- validazione temporale che certifichi l'istante in cui il messaggio è stato trasmesso
- validazione temporale che certifichi l'istante in cui il messaggio è stato ricevuto.

Flusso delle interazioni

Fig. 5.10: Non ripudiabilità della trasmissione

A: Verifica numero tentativi di inoltro

Il fruitore realizza una delle seguenti azioni:

A.1 [Primo Invio]

Il fruitore inizializza il numero di tentativi di inoltro ad 1 e prosegue a quanto indicato al passo B.

A.2 [Invio Successivo con numero di tentativi inferiore al massimo pattuito]

Il fruitore incrementa il numero di tentativi di inoltro e da seguito a quanto indicato al passo B.

A.3 [Superamento numero di tentativi massimi pattuiti]

Il fruitore utilizza i canali alternativi per superare o evidenziare problemi di comunicazione riscontrati non proseguendo con i passi successivi.

B: Richiesta

Il messaggio di richiesta viene costruito aggiungendo un identificativo univoco del messaggio (vedi **IDAS02** o **IDAR02**), l'istante di trasmissione

- **SOAP**: <*wsu:Timestamp*> della ws-security
- **REST:** claim *iat* contenuta nella payload del token JWT

Tutti gli elementi utili al non ripudio, inclusi quelli descritti in **IDAS02** o **IDAR02**, vengono firmati utilizzando il profilo desiderato **IDAS03** o **IDAR03** per garantire:

- l'integrità del contenuto,
- l'identità del mittente
- il momento di invio.

Il fruitore invia il messaggio di richiesta all'interfaccia di servizio dell'erogatore.

Il messaggio viene trasmesso su un canale sicuro per garantire:

- la confidenzialità a livello di canale (almeno profilo **IDAC01**)

C: Persistenza Erogatore

Per garantire la non ripudiabilità del messaggio ricevuto dal fruitore, così come previsto dai profili utilizzati:

- L'erogatore provvede all'autenticazione del fruitore;
- L'erogatore verifica l'integrità del messaggio firmato, inoltre la presenza dell'istante di trasmissione nel messaggio, ne garantisce validità a lungo termine;

Per assicurare l'opponibilità a terzi:

- L'erogatore rende persistente il messaggio firmato tracciando l'istante di ricezione.

La persistenza del messaggio:

- DEVE garantire la capacità di ricercare ed esportare le informazioni memorizzate;
- DEVE essere garantita per un periodo di tempo che dipende dagli accordi tra le parti.

L'erogatore realizza una delle seguenti azioni:

C.1 [Prima Ricezione]

L'erogatore inizializza il numero di tentativi di richieste ricevute ad 1 e prosegue al passo D.

C.2 [Duplicato con numero di tentativi inferiore al massimo pattuito]

L'erogatore accerta la presenza di un identificativo univoco del messaggio già ricevuto, a causa di una mancata ricezione del messaggio di conferma da parte del fruitore.

Incrementa il numero di tentativi di richieste ricevute e prosegue al passo D.

C.3 [Superamento numero massimo di tentativi pattuiti]

L'erogatore accerta la presenza di un identificativo univoco del messaggio già ricevuto, a causa di una mancata ricezione del messaggio di conferma da parte del fruitore.

L'erogatore accerta di aver raggiunto il numero massimo di tentativi di richieste ricevute. L'erogatore utilizza i canali alternativi per superare o evidenziare problemi di comunicazione riscontrati non proseguendo con i passi successivi.

D: Risposta

L'erogatore costruisce un messaggio di conferma includendo un identificativo che permetta di associare univocamente al messaggio di richiesta (ad esempio il digest presente nel messaggio di richiesta) e l'istante di trasmissione.

Inoltre al messaggio di conferma viene aggiunto l'istante di trasmissione

- **SOAP**: <*wsu:Timestamp*> della ws-security
- **REST:** claim *iat* contenuta nella payload del token JWT

Tutti gli elementi utili al non ripudio, inclusi quelli descritti in **IDAS02** o **IDAR02**, vengono firmati utilizzando il profilo desiderato **IDAS03** o **IDAR03** per garantire:

- l'integrità del contenuto,
- l'identità del mittente

- il momento di invio.

E: Persistenza Richiedente

Per garantire la non ripudiabilità del messaggio inviato all'erogatore:

- Il fruitore provvede all'autenticazione dell'erogatore rispetto al messaggio di risposta.
- Il fruitore verifica l'integrità del messaggio di risposta firmato in cui, la presenza del timestamp sul protocollo di messaggio ne garantisce validazione a lungo termine e il tempo di ricezione.

ed inoltre per assicurare l'opponibilità a terzi:

- Il fruitore rende persistente il messaggio di risposta firmato.

La persistenza del messaggio deve:

- garantire la capacità di ricercare ed esportare le informazioni memorizzate;
- essere garantita per un periodo di tempo che dipende dagli accordi tra le parti.

Note

Nel caso in cui il fruitore non riceve il messaggio di risposta entro i termini concordati tra le parti, si ritiene la comunicazione non conclusa, in quanto può presentarsi uno dei seguenti casi:

- il messaggio di richiesta non ha raggiunto l'erogatore
- il messaggio di richiesta ha raggiunto l'erogatore ma non ha ricevuto il messaggio di risposta.

In queste situazioni il fruitore riesegue il passo A.

5.5 Elenco degli algoritmi

Di seguito sono elencati gli algoritmi di criptazione individuati per la corretta implementazione dei profili di sicurezza.

5.5.1 Digest SOAP

SIGLA	URI
SHA-224	http://www.w3.org/2001/04/xmldsig-more#sha224
SHA-256	http://www.w3.org/2001/04/xmlenc#sha256
SHA-384	http://www.w3.org/2001/04/xmldsig-more#sha384
SHA-512	http://www.w3.org/2001/04/xmlenc#sha512
HMAC-SHA-224	http://www.w3.org/2001/04/xmldsig-more#hmac-sha224
HMAC-SHA-256	http://www.w3.org/2001/04/xmldsig-more#hmac-sha256
HMAC-SHA-384	http://www.w3.org/2001/04/xmldsig-more#hmac-sha384
HMAC-SHA-512	http://www.w3.org/2001/04/xmldsig-more#hmac-sha512

5.5.2 Signature public key SOAP

SIGLA	URI
DSA-SHA-256	http://www.w3.org/2009/xmldsig11#dsa-sha256
RSA-SHA-224	http://www.w3.org/2001/04/xmldsig-more#rsa-sha224
RSA-SHA-256	http://www.w3.org/2001/04/xmldsig-more#rsa-sha256
RSA-SHA-384	http://www.w3.org/2001/04/xmldsig-more#rsa-sha384
RSA-SHA-512	http://www.w3.org/2001/04/xmldsig-more#rsa-sha512
ECDSA-SHA-224	http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha224
ECDSA-SHA-256	http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256
ECDSA-SHA-384	http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha384
ECDSA-SHA-512	http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha512

5.5.3 Canonicalization

SIGLA	URI
Canonical XML 1.0	http://www.w3.org/TR/2001/REC-xml-c14n-20010315
Canonical XML 1.1	http://www.w3.org/2006/12/xml-c14n11
Exclusive XML Canonicalization 1.0	Exclusive XML Canonicalization 1.0

5.5.4 Digest and signature public key REST

SIGLA	ALGORITMI
HS256	HMAC using SHA-256 hash algorithm
HS384	HMAC using SHA-384 hash algorithm
HS512	HMAC using SHA-512 hash algorithm
RS256	RSA using SHA-256 hash algorithm
RS384	RSA using SHA-384 hash algorithm
RS512	RSA using SHA-512 hash algorithm
ES256	ECDSA using P-256 curve and SHA-256 hash algorithm
ES384	ECDSA using P-384 curve and SHA-384 hash algorithm
ES512	ECDSA using P-521 curve and SHA-512 hash algorithm

5.5.5 Digest REST

SIGLA	ALGORITMI
S256	SHA-256 hash algorithm
S384	SHA-384 hash algorithm
S512	SHA-512 hash algorithm

5.6 Riferimenti Bibliografici

1. **RFC 7519**[422] - J. Bradley; N. Sakimura; M. Jones. «JSON Web Token (JWT)».
2. **RFC 7515**[423] - J. Bradley; N. Sakimura; M. Jones. «JSON Web Signature (JWS)».
3. ISO/IEC 13888-1:2009. «*Information technology - Security techniques - Non-repudiation - Part 1: General*». ISO/IEC, 2009

[422] https://tools.ietf.org/html/rfc7519.html
[423] https://tools.ietf.org/html/rfc7515.html

4. OASIS wss-x509TokenProfile-v1.1.1-csprd01 - A. Nadalin; C. Kaler; R. Monzillo; P. Hallam-Baker; C. Milono. «Web Services Security X.509 Certificate Token Profile Version 1.1.1[424]»
5. **RFC 5246**[425] - T. Dierks; E. Rescorla - «The Transport Layer Security (TLS) Protocol Version 1.2»
6. **RFC 8446**[426] - E. Rescorla - «The Transport Layer Security (TLS) Protocol Version 1.3»
7. **RFC 2119**[427] - S. Bradner - «Key words for use in RFCs to Indicate Requirement Levels»
8. **RFC 3553**[428] - E. Rescorla, et al. - «Guidelines for Writing RFC Text on Security Considerations»

[424] http://docs.oasis-open.org/wss-m/wss/v1.1.1/csprd01/wss-x509TokenProfile-v1.1.1-csprd01.html
[425] https://tools.ietf.org/html/rfc5246.html
[426] https://tools.ietf.org/html/rfc8446.html
[427] https://tools.ietf.org/html/rfc2119.html
[428] https://tools.ietf.org/html/rfc3553.html

Indice