Linee Guida sull'interoperabilità tecnica delle Pubbliche Amministrazioni – Pattern di interazione

Versione 1.1 del 29/11/2023

AGID | via Liszt, 21 – 00144 Roma | protocollo@pec.agid.gov.it | www.agid.gov.it | tel. 06 85264.1

Versione	Data	Tipologia modifica
1	27/04/2021	Prima emissione
1.1	29/11/2023	Modifica Titolo Aggiunta Capitolo 7 - Richiesta Risorse Massive Fix Format Sommario

Sommario

Introduzione

Il pattern di interazione definisce le modalità secondo le quali un fruitore ed un erogatore possono interagire. L'interazione avviene definendo le API.

I pattern di interazione quali riferimenti concettuali coniugano, fatto salvo per l'accesso CRUD, i 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 Operativo - Pattern di sicurezza.

Il Documento operativo descrive i pattern di interazione individuati da AgID che gli erogatori DEVONO utilizzare per soddisfare le necessità individuate dai requisiti funzionali e non funzionali delle specifiche interazioni con i propri fruitori.

Data la variabilità nel tempo delle esigenze delle amministrazioni e delle tecnologie abilitanti, nonché considerata la natura incrementale del ModI, l'elenco dei pattern di interazione non è da intendersi esaustivo. Nel caso in cui un'amministrazione abbia esigenze non ricoperte nei seguenti pattern di interazione DEVE informare l'Agenzia per l'Italia Digitale, nei modi indicati nel capitolo 7 «Pattern e profili di interoperabilità» delle Linee Guida sull'interoperabilità tecnica delle Pubbliche Amministrazioni.

Capitolo 1

Ambito di applicazione

Il presente Documento operativo, allegato delle Linee Guida sull'interoperabilità tecnica delle Pubbliche Amministrazioni, riporta i pattern di interazione definiti nel ModI.

1.1 Soggetti destinatari

Il Documento Operativo è destinato ai soggetti di cui all'articolo 2, comma 2 del CAD, così come indicato dall'articolo 75 dello stesso. I destinatari la attuano nella realizzazione dei propri sistemi informatici che fruiscono o erogano dati e/o servizi digitali di/ad altri soggetti.

Il Documento Operativo è rivolto ai soggetti privati che devono interoperare con la Pubblica Amministrazione per erogare o fruire di dati e servizi tramite sistemi informatici.

Capitolo 2

Riferimenti e sigle

2.1 Note di lettura del documento

Conformemente alle norme ISO/IEC Directives, Part 3 per la stesura dei documenti tecnici la presente linea guida utilizzerà le parole chiave «DEVE», «DEVONO», «NON DEVE», «NON DEVONO», «E" RICHIESTO», «DOVREBBE», «NON DOVREBBE», «RACCOMANDATO», «NON RACCOMANDATO» «PUO"» e «OPZIONALE», la cui interpretazione è descritta di seguito.

- **DEVE** o **DEVONO**, indicano un requisito obbligatorio per rispettare la linea guida;
- **NON DEVE** o **NON DEVONO**, indicano un assoluto divieto delle specifiche;
- **DOVREBBE** o **RACCOMANDATO** o **NON DOVREBBE** o **NON RACCOMANDATO**, indicano che le implicazioni devono essere comprese e attentamente pesate prima di scegliere approcci alternativi;
- **PUÓ** o **POSSONO** o l'aggettivo **OPZIONALE**, indica che il lettore può scegliere di applicare o meno senza alcun tipo di implicazione la specifica.

2.2 Riferimenti Normativi

Sono riportati di seguito gli atti normativi di riferimento del presente documento.

[CAD]	decreto legislativo 7 marzo 2005, n. 82 recante «Codice dell'Amministrazione Digitale»; NOTA – Il D. Lgs. 82/2005 è noto anche con l'abbreviazione «CAD»
[EIF]	European Interoperability Framework (EIF)
[CE 1205/2008]	Regolamento (CE) n. 1205/2008 della Commissione del 3 dicembre 2008 recante attuazione della direttiva 2007/2/CE del Parlamento europeo e del Consiglio per quanto riguarda i metadati
[D.lgs. 196/2003]	Codice in materia di protezione dei dati personali
[UE 2016/679]	Regolamento (UE) n. 2016/679 del 27 aprile 2016 relativo alla protezione delle persone fisiche con riguardo al trattamento dei dati personali (in breve GDPR)

[UE 910/2014] Regolamento (UE) n. 910/2014 del 23 luglio 2014 in materia di identificazione elettronica e servizi fiduciari per le transazioni elettroniche nel mercato interno (in breve eIDAS)

[DUE 2019/1024] Direttiva (UE) 2019/1024 del Parlamento Europeo e del Consiglio del 20 giugno 2019 relativa all'apertura dei dati e al riutilizzo dell'informazione del settore pubblico

2.3 Standard di riferimento

Sono riportati di seguito gli standard tecnici indispensabili per l'applicazione del presente documento.

[RFC7396] JSON Merge Patch

[RFC7389] Separazione di controllo e piano utente per Proxy Mobile IPv6

[RFC5789] Metodo PATCH per HTTP

2.4 Termini e definizioni

Di seguito si riportano gli ACRONIMI che verranno utilizzati nella presente Linee Guida:

[AgID] Agenzia per l'Italia Digitale

[CAD] Codice Amministrazione Digitale, D.lgs. 7 marzo 2005, n. 82

[PA] Pubblica Amministrazione

[UML] Linguaggio di modellazione unificato (Unified Modeling Language)

[RPC] Remote procedure call

[SOAP] Simple Object Access Protocol

[REST] Representational State Transfer

Capitolo 3

Principi generali

3.1 Interazione bloccante e 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. 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

3.2 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 potenzialmente 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 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, in breve uno sviluppatore deve poter concentrare la propria attività sullo sviluppo sulle attività di predisposizione dei contenuti delle request e il consumo dei contenuti delle response) all'utilizzatore del meccanismo. I problemi di comunicazione possono essere gestiti anch'essi in modo trasparente, oppure generare errori o eccezioni. 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.

3.3 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 CorrelationID oppure tramite il confronto dati basato su dati che fungono da chiave.

Capitolo 4

Pattern bloccanti

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

- L'esecuzione del metodo 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.

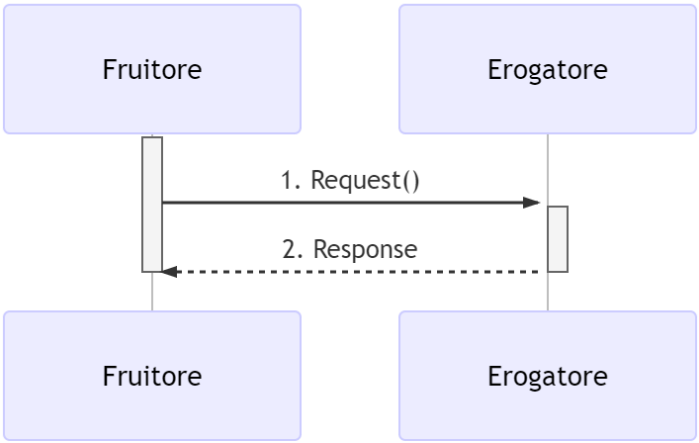

Figura 1- Interazione bloccante RPC

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

Nel seguito, per gli esempi proposti si fa riferimento ad un'amministrazione denominata **ente.example** che offre un'interfaccia di servizio secondo le due diverse tecnologie REST o SOAP. Inoltre, per quanto riguarda i pattern 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.

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

Di seguito le indicazioni per le tecnologie accolte dal ModI.

L'AgID assicura l'aggiornamento degli stessi per soddisfare le esigenze espresse dalle PA.

4.1 [BLOCK_REST] Blocking 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 previsti dall'interfaccia, con il 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 su un URL contenente gli ID interessati ed il nome del metodo;
- Al passo (2) l'erogatore DEVE utilizzare HTTP status 2xx a meno che non si verifichino errori.

4.1.1 Regole di processamento

Al termine del processamento della richiesta, l'erogatore DEVE fare uso dei codici di stato HTTP rispettando la semantica. 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 fornendo nel body di risposta dettagli circa l'errore;
- DOVREBBE, in caso di rappresentazione semanticamente non corretta, ritornare HTTP status 422 Unprocessable Entity;

- DOVREBBE, se qualcuno degli ID nel path o nel body non esiste, restituire HTTP status 404 Not Found, indicando nel body di risposta quale degli ID è mancante;
- PUÒ, se ipotizza che la richiesta sia malevola, ritornare HTTP status 400 Bad Request o HTTP status 404 Not Found
- 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 2xx 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.

4.1.2 Esempio

Specifica Servizio

https://api.ente.example/rest/nome-api/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:
         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
        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.

Endpoint

https://api.ente.example/rest/nome-api/v1/resources/1234/M

1. Request

```
POST /rest/nome-api/v1/resources/1234/M HTTP/1.1
Host: api.ente.example
Content-Type: application/json

{
  "a": {
    "a1s": [ 1, 2 ],
    "a2": "RGFuJ3MgVG9vbHMgYXJlIGNvb2wh"
```

```
  },
  "b": "Stringa di esempio"
}
```

2. Response HTTP status 200 OK

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

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

2. Response HTTP status 500 Internal Server Error

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/problem+json

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

2. Response HTTP status 400 Bad Request

```
HTTP/1.1 400 Bad Request
Content-Type: application/problem+json

{
    "type": "https://apidoc.ente.example/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 HTTP status 404 Not Found

```
HTTP/1.1 404 Not Found
Content-Type: application/problem+json

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

4.2 [BLOCK_SOAP] Blocking SOAP

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

4.2.1 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 fornendo dettagli circa l'errore utilizzando il meccanismo della SOAP fault;
- Se l'erogatore ipotizza che la richiesta sia malevola PUÒ ritornare HTTP status 400 Bad Request o HTTP status 404 Not Found
- 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 indicando il motivo dell'errore nella SOAP fault;
- In caso di successo restituire HTTP status 200 OK, riempiendo il body di risposta con il risultato dell'operazione.

4.2.2 Esempio

Specifica Servizio

https://api.ente.example/soap/nome-api/v1?wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://ente.example/nome-api"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap12/" name="SOAPBlockingImplService"
targetNamespace="http://ente.example/nome-api">
  <wsdl:types>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:tns="http://ente.example/nome-api" attributeFormDefault="unqualified"
elementFormDefault="unqualified" targetNamespace="http://ente.example/nome-api">
      <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>
      </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.ente.example/soap/nome-api/v1"/>
            </wsdl:port>
          </wsdl:service>
        </wsdl:definitions>
```

A seguire un esempio di chiamata al metodo M.

Endpoint

https://api.ente.example/soap/nome-api/v1

1. Request Body

```
        <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
```

```
        xmlns:m="http://ente.example/nome-api">
        <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://ente.example/nome-api">

        <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://ente.example/nome-api">
        <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>
```

Capitolo 5

Pattern non bloccanti

I pattern di interazione non bloccanti di tipo RPC-like sono quelli in cui un fruitore invia 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 dell'erogatore sono lunghi perché:

- la richiesta è onerosa in termini temporali;
- l'erogatore 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 CorrelationID e l'endpoint per le callback). Queste sono estranee solitamente alla business logic del servizio, ma è necessario definirle a livello di API 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 API 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 CorrelationID può essere sostituito da questi identificativi già previsti dal servizio.

Nel seguito, per gli esempi proposti si fa riferimento ad un'amministrazione denominata **ente.example** che offre un'interfaccia di servizio secondo le due diverse tecnologie REST o SOAP. Inoltre, per quanto riguarda i pattern 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.

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

Di seguito le indicazioni per le tecnologie accolte dal ModI.

L'AgID assicura l'aggiornamento degli stessi per soddisfare le esigenze espresse dalle PA.

5.1 Pattern non bloccante RPC PUSH (basato su callback)

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.

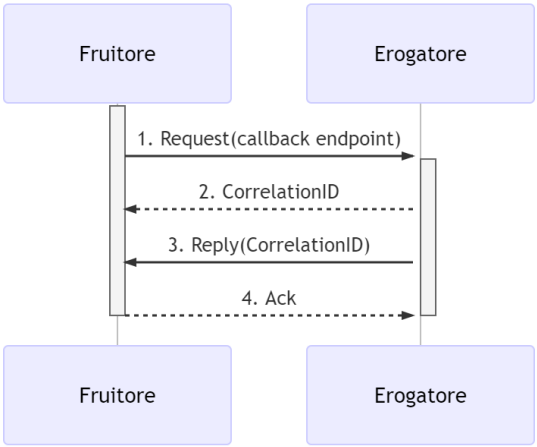

Figura 2 - Interazione non bloccante tramite callback

In questo pattern (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 CorrelationID (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 CorrelationID. Il fruitore riconosce (sempre mediante un messaggio di ack) la ricezione della risposta (passo 4).

5.1.1 [NONBLOCK_PUSH_REST] Not Blocking Push REST

Nel caso in cui il pattern 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 previsti dall'interfaccia, con il 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 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 ;
- Al passo (2), l'erogatore DEVE fornire insieme all'acknowledgement della richiesta nel body, il CorrelationID utilizzando l'header HTTP custom X-Correlation-ID; Il codice HTTP di stato DEVE essere HTTP status 202 Accepted a meno che non si verifichino errori;
- Al passo (3), l'erogatore DEVE utilizzare lo stesso CorrelationID 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 a meno che non si verifichino errori.

5.1.1.1 Regole di processamento

Al termine del processamento delle richieste, l'erogatore ed il fruitore DEVONO fare uso dei codici di stato HTTP rispettando la semantica. 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 fornendo dettagli circa l'errore;
- In caso di dati errati DEVONO restituire HTTP status 400 Bad Request fornendo nel body di risposta dettagli circa l'errore;
- In caso di representation semanticamente non corretta DEVONO ritornare HTTP status 422 Unprocessable Entity;

- Se qualcuno degli ID nel path o nel body non esiste, DEVONO restituire HTTP status 404 Not Found, indicando nel body di risposta quale degli ID è mancante;
- Se si ipotizza che la richiesta sia malevola, PUÒ ritornare HTTP status 400 Bad Request o HTTP status 404 Not Found
- 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. In caso di ricezione corretta della risposta, il fruitore DEVE restituire HTTP status 200 OK, 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.

5.1.1.2 Esempio

Specifica Servizio Server

https://api.ente.example/rest/nome-api/v1/RESTCallbackServer.yaml

```
openapi: 3.0.1
info:
 title: RESTCallbackServer
 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: 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:
                          $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: |
                 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.client.example/rest/nome-api/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:
         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: |
        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 il fruitore effettua una richiesta al metodo M; l'erogatore conferma la presa in carico della richiesta ritornando HTTP status 202 Accepted.

Endpoint:

https://api.ente.example/rest/nome-api/v1/resources/1234/M

1. Request

```
POST /rest/nome-api/v1/resources/1234/M HTTP/1.1
Host: api.ente.example
Content-Type: application/json
X-ReplyTo: https://api.client.example/rest/v1/nomeinterfacciaclient/Mresponse

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

2. Response

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

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

Quindi l'erogatore notifica al fruitore l'avvenuto processamento delle informazioni tramite una HTTP method POST all'URL concordato. Il fruitore conferma con HTTP status 200 OK l'avvenuta ricezione della notifica

Endpoint

https://api.client.example/rest/v1/nomeinterfacciaclient/Mresponse

3. Request

```
POST /rest/v1/nomeinterfacciaclient/Mresponse HTTP/1.1
Host: api.client.example
Content-Type: application/json
X-Correlation-ID: 69a445fb-6a9f-44fe-b1c3-59c0f7fb568d

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

4. Response

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

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

5.1.2 [NONBLOCK_PUSH_SOAP] Not Blocking Push SOAP

Nel caso di implementazione mediante tecnologia SOAP, l'endpoint di callback ed il CorrelationID, 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 CorrelationID utilizzando l'header SOAP custom X-Correlation-ID;
- Al passo (3), l'erogatore DEVE utilizzare lo stesso CorrelationID 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.

5.1.2.1 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 fornendo dettagli circa l'errore, utilizzando il meccanismo della SOAP fault;
- Nel caso in cui qualcuno degli ID nel path o nel body non esista, DEVONO restituire HTTP status 500 Internal Server Error, indicando nel body di risposta quale degli ID è mancante;
- Se ipotizzano che la richiesta sia malevola POSSONO ritornare HTTP status 400 Bad Request o HTTP status 404 Not Found
- 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 in caso di presenza della payload SOAP, riempiendo il body di risposta con il risultato relativo alla richiesta.

- HTTP status 200 OK o HTTP status 202 Accepted 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 ri-trasmettere le richieste.

5.1.2.2 Esempio

Specifica Servizio Erogatore

https://api.ente.example/soap/nome-api/v1?wsdl

```
<?xml version="1.0"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://ente.example/nome-api"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap12/"
    name="SOAPCallbackServerService"
    targetNamespace="http://ente.example/nome-api">
    <wsdl:types>
        <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns:tns="http://ente.example/nome-api" attributeFormDefault="unqualified"
elementFormDefault="unqualified" targetNamespace="http://ente.example/nome-api">

            <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>

            <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>

            <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.ente.example/soap/nome-api/v1"/>
                </wsdl:port>
            </wsdl:service>

        </wsdl:definitions>
```

Specifica Servizio Fruitore (callback)

https://api.client.example/soap/nome-api/v1?wsdl

```
        <?xml version='1.0' encoding='UTF-8'?>
        <wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
            xmlns:tns="http://ente.example/nome-api"
            xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap12/"
            name="SOAPCallbackClientInterfaceService"
            targetNamespace="http://ente.example/nome-api">
            <wsdl:types>
                <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                    xmlns:tns="http://ente.example/nome-api" attributeFormDefault="unqualified"
elementFormDefault="unqualified" targetNamespace="http://ente.example/nome-api">

                    <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"/>
                    <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.client.example/soap/nome-api/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.ente.example/soap/nome-api/v1

1. Request Body

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:m="http://ente.example/nome-api">
    <soap:Header>
        <m:X-ReplyTo>https://api.client.example/soap/nome-api/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://ente.example/nome-api">>
    <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>
```

Di seguito un esempio di richiesta da parte dell'erogatore verso la callback del fruitore.

Endpoint

https://api.client.example/soap/nomeinterfacciaclient/v1Method

3. Request Body

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:m="http://ente.example/nome-api">
    <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://ente.example/nome-api">
    <soap:Body>
        <m:MRequestResponseResponse>
            <return>
                <outcome>OK</outcome>
            </return>
        </m:MRequestResponseResponse>
    </soap:Body>
</soap:Envelope>
```

5.2 Pattern non bloccanti RPC PULL (busy waiting)

Questo caso particolare, denominato RPC PULL, è utilizzabile nel caso in cui il fruitore non abbia a sua volta la possibilità di esporre una interfaccia di servizio per la ricezione delle risposte. L'erogatore fornisce un URL per verificare lo stato di processamento di una richiesta e, alla fine dell'elaborazione della stessa, il risultato.

Questo scenario prevede due possibili workflow, uno per REST ed uno per SOAP riportati nelle seguenti figure.

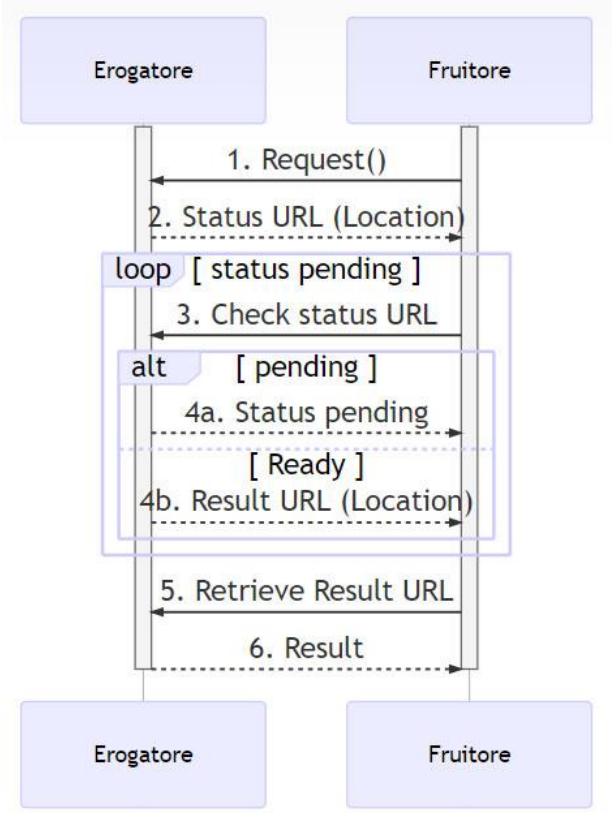

Figura 3 - Interazione non bloccante tramite busy waiting REST

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

- un URL dove verificare lo stato del processamento (REST);
- oppure un CorrelationID (SOAP).

D'ora in poi il fruitore, periodicamente, verifica (passo 3.) lo stato dell'operazione utilizzando:

- l'URL indicato (REST)
- oppure il CorrelationID (SOAP)

fin quando il risultato dell'operazione sarà pronto (passo 4b.).

Gli intervalli di polling possono essere definiti tra le parti.

Quando la risposta è pronta il fruitore può accedere (passi 5. e 6.) al risultato del processamento.

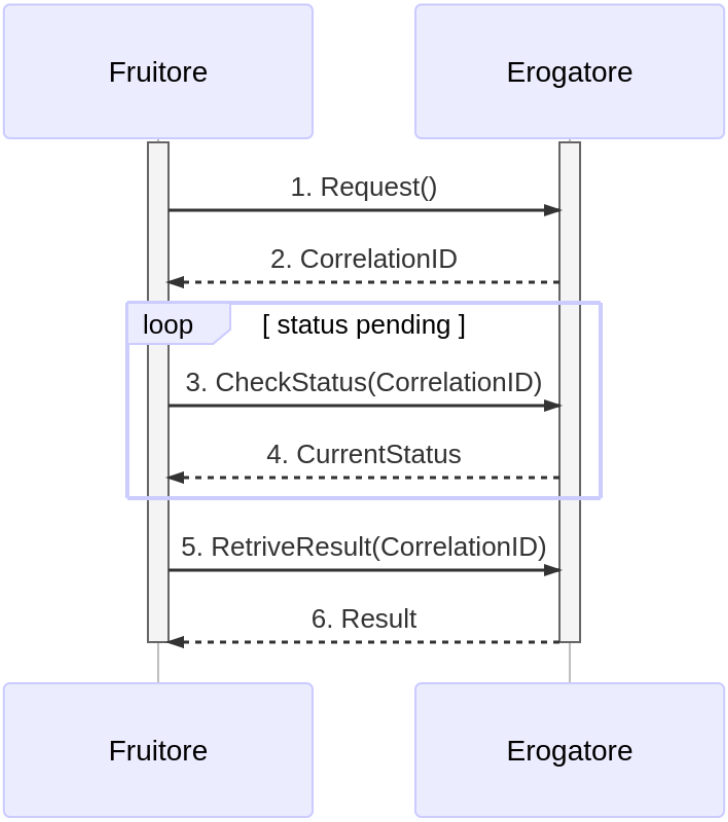

Figura 4 - Interazione non bloccante tramite busy waiting SOAP

5.2.1 [NONBLOCK_PULL_REST] Not Blocking Pull REST

Quando il profilo viene 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 URL per interrogare lo stato di processamento utilizzando HTTP header Location , il codice HTTP di stato DEVE essere HTTP status 202 Accepted 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, sulla base dello stato del processamento, che l'operazione: * 4a. non è completata (HTTP status 200 OK) * 4b. che la risorsa finale è pronta (HTTP status 303 See Other) all'URL indicato nell' HTTP header Location;
- Al passo (5), il fruitore DEVE recuperare la risorsa con una richiesta HTTP method GET all'URL indicato in (4b.);
- Al passo (6) l'erogatore risponde con la rappresentazione della risorsa, il codice HTTP restituito è HTTP status 200 OK.

5.2.1.1 Regole di processamento

Al termine del processamento delle richieste, l'erogatore deve fare uso dei codici di stato HTTP rispettando la semantica. 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 fornendo nel body di risposta dettagli circa l'errore;
- DOVREBBE, in caso di representation semanticamente non corretta, ritornare HTTP status 422 Unprocessable Entity;
- DEVE, se qualcuno degli ID nel path o nel body non esiste, restituire HTTP status 404 Not Found, indicando nel body di risposta quale degli ID è mancante;
- PUÒ, se ipotizza che la richiesta sia malevola, ritornare HTTP status 400 Bad Request o HTTP status 404 Not Found;
- 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.
- In caso di ricezione corretta della risposta, il fruitore DEVE restituire HTTP status 200 OK, 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.

5.2.1.2 Esempio

Specifica Servizio Server

https://api.ente.example/rest/nome-api/v1/openapi.yaml

```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:
              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:
              $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
          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
```

Il fruitore richiede la risorsa M, e l'erogatore risponde dichiarando di aver preso in carico la richiesta.

Endpoint

https://api.ente.example/rest/nome-api/v1/resources/1234/M

1. Request

```
POST /rest/nome-api/v1/resources/1234/M HTTP/1.1
Host: api.ente.example
Content-Type: application/json

{
  "a": {
     "a1": [1, "…", 2],
```

```
      "a2": "Stringa di esempio"
    },
    "b": "Stringa di esempio"
}
```

2. Response

```
HTTP/1.1 202 Accepted
Content-Type: application/json
Location: /rest/nome-api/v1/resources/1234/M/8131edc0

{
  "status": "accepted",
  "message": "Preso carico della richiesta",
  "id": "8131edc0"
}
```

Quindi il fruitore verifica lo stato dell'esecuzione di M (3). L'erogatore ritorna un payload diverso a seconda dei casi: - processamento ancora in atto (4a) - e di processamento avvenuto (4b).

Endpoint

https://api.ente.example/rest/nome-api/v1/resources/1234/M/8131edc0

3. Request

```
GET /rest/nome-api/v1/resources/1234/M/8131edc0 HTTP/1.1
Host: api.ente.example
```

4a. Response (HTTP status 200 OK)

```
HTTP/1.1 200 OK
Host: api.ente.example
Content-Type: application/json

{
  "status": "processing",
```

```
  "message": "Richiesta in fase di processamento"
}
```

4b. Response (HTTP status 303 See Other)

Poiché al termine del processamento il client viene rediretto verso la risorsa finale, il payload deve contenere solamente le informazioni utili al redirect. Questo anche perché alcuni client potrebbero seguire direttamente l'"HTTP header Location ignorando il payload HTTP status 303 See Other e ritornando invece quello indicato nel punto 5.

```
HTTP/1.1 303 See Other
Content-Type: application/json
Content-Location: /rest/nome-api/v1/resources/1234/M/8131edc0
Location: /rest/nome-api/v1/resources/1234/M/8131edc0/result

{
  "status": "done",
  "message": "Processamento completo",
  "href": "https://api.ente.example/rest/nome-api/v1/resources/1234/M/8131edc0/result"
}
```

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

Endpoint

https://api.ente.example/rest/nome-api/v1/resources/1234/M/8131edc0/result

5. Request

```
GET /rest/nome-api/v1/resources/1234/M/8131edc0/result HTTP/1.1
Host: api.ente.example
```

6. Response

```
HTTP/1.1 200 OK
Host: api.ente.example
Content-Type: application/json
```

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

5.2.2 [NONBLOCK_PULL_SOAP] Not Blocking Pull SOAP

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, controllare 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 CorrelationID riportato nel header custom SOAP X-Correlation-ID;
- Al passo (3), il fruitore DEVE utilizzare il CorrelationID 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 CorrelationID di cui al passo (2) al fine di richiedere il risultato della richiesta;
- Al passo (6), l'erogatore fornisce il risultato del processamento.

5.2.2.1 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 fornendo dettagli circa l'errore utilizzando il meccanismo della SOAP fault;
- Se l'erogatore ipotizza che la richiesta sia malevola PUÒ ritornare HTTP status 400 Bad Request o HTTP status 404 Not Found;

- 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 indicando il motivo dell'errore nella SOAP fault;
- In caso di successo restituire HTTP status 200 OK, riempiendo il body di risposta con il risultato dell'operazione.

5.2.2.2 Esempio

Specifica Servizio Erogatore

https://api.ente.example/soap/nome-api/v1?wsdl

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

            <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" />

            <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" />
                        </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"/>
            </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>

                </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.ente.example/soap/nome-api/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.ente.example/soap/nome-api/v1

1. Request Body

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:m="http://ente.example/nome-api">
    <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 (HTTP status code 200 OK)

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:m="http://ente.example/nome-api">
    <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.

Endpoint

https://api.ente.example/soap/nome-api/v1

3. Request Body status

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:m="http://ente.example/nome-api">
    <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. Response Body (HTTP status code 200 OK) status in attesa

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

4. Response Body (HTTP status code 200 OK) status completata

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:m="http://ente.example/nome-api">
    <soap:Body>
        <m:MProcessingStatusResponse>
            <return>
                <status>done</status>
                <message>Richiesta completata</message>
            </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.ente.example/soap/nome-api/v1

5. Request Body result

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:m="http://ente.example/nome-api">
    <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. Response Body (HTTP status code 200 OK) result

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:m="http://ente.example/nome-api">
    <soap:Body>
        <m:MResponseResponse>
            <return>
                <c>OK</c>
            </return>
        </m:MResponseResponse>
    </soap:Body>
</soap:Envelope>
```

Capitolo 6

Accesso CRUD a risorse

In un'architettura orientata alle risorse le API vengono utilizzate, più che per eseguire compiti complessi, per 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.

In questo scenario si assume che le API siano utilizzate per la gestione da parte del fruitore delle risorse messe a disposizione dall'erogatore. L'insieme di operazioni CRUD offerte dall'erogatore dipende dalla natura della risorse e dalla relazione costruita con i fruitori: sono possibili relazioni in cui l'erogatore rende disponibile ai fruitori la sola operazione di lettura (Read).

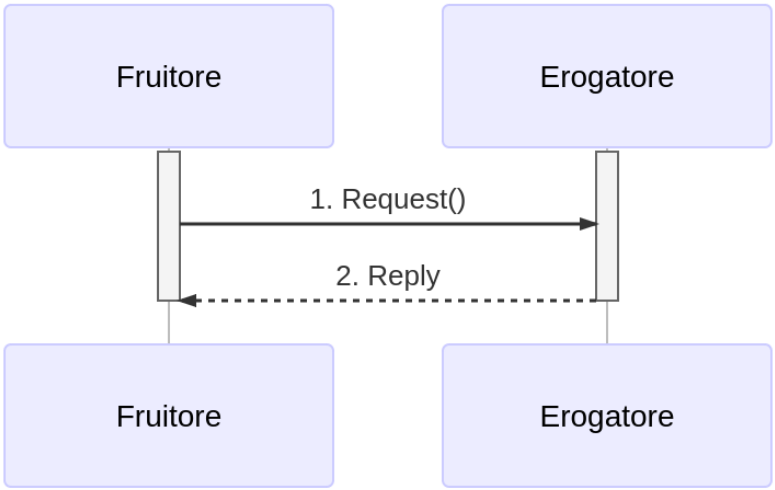

Figura 5 - Interazione CRUD

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 prevede lo sviluppo di una interfaccia REST.

6.1 [CRUD_REST] CRUD REST

6.1.1 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 metodi standard dell'HTTP per indicare tali operazioni. In particolare, la seguente mappatura viene utilizzata:

CRUD	METODO HTTP	ESITO (STATO HTTP) SE APPLICATO A INTERA COLLEZIONE (ES. /CLIENTI)	ESITO (STATO HTTP) SE APPLICATO A RISORSA SPECIFICA (ES. /CLIENTI/{ID})
Create	POST	HTTP status 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.

CRUD	METODO HTTP	ESITO (STATO HTTP) SE APPLICATO A INTERA COLLEZIONE (ES. /CLIENTI)	ESITO (STATO HTTP) SE APPLICATO A RISORSA SPECIFICA (ES. /CLIENTI/{ID})
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 e HTTP method PATCH per la sostituzione e l'applicazione di modifiche ad una risorsa rispettivamente. È possibile utilizzare anche altri HTTP method a patto che si rispettino i dettami dell'approccio RESTful.

In alcuni casi l'' HTTP method PUT 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 bisogna usare alcuni accorgimenti, perché questo metodo non è definito nelle nuove specifiche di HTTP/1.1 del 2014 ma nel precedente RFC 5789.

NON SI DOVREBBE associare un significato di patch a dei media-type che non lo prevedano (eg. application/json o application/xml) ma utilizzare dei media-type adeguati[1].

È possibile ad esempio usare application/merge-patch+json definito in RFC 7396 facendo attenzione:

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

6.1.2 Esempio

Per illustrare l'approccio RESTful al CRUD, faremo l'esempio di 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 (creazione), HTTP method DELETE (eliminazione), HTTP method PATCH (modifica) e HTTP method GET (lettura).

Specifica Servizio Server

https://api.ente.example/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:
```

[1] Cf. https://www.rfc-editor.org/errata/eid3169

```
      - $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
        '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':
            $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
              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:
              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.

1. Request

```
POST /rest/appuntamenti/v1/municipio/{id_municipio}/ufficio/{id_ufficio}/prenotazioni HTTP/1.1
Host: api.ente.example
Content-Type: application/json

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

2. Response

```
        HTTP/1.1 201 Created
        Content-Type: application/json
        Location:
https://api.ente.example/rest/appuntamenti/v1/municipio/{id_municipio}/ufficio/{id_ufficio}/prenotazioni/12
323254

        {
            "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 al passo precedente.

1. Request

```
        GET /rest/appuntamenti/v1/municipio/{id_municipio}/ufficio/{id_ufficio}/prenotazioni/12323254
HTTP/1.1
        Host: api.ente.example
```

2. Response

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

        {
            "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.

1. Request

```
PATCH /rest/appuntamenti/v1/municipio/{id_municipio}/ufficio/{id_ufficio}/prenotazioni/12323254
HTTP/1.1
Host: api.ente.example
Content-Type: application/merge-patch+json

{
    "dettagli": {
        "data": "2018-12-03T14:29:12.137Z",
        "motivazione": "nuova motivazione"
    }
}
```

2. Response

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

{
 "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. La response ritorna il media-type suggerito dalla specifica tramite HTTP header Accept-Patch.

1. Request

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

```
        Host: api.ente.example
        Content-Type: application/json

            {
              "dettagli": {
                    "data": "2018-12-03T14:29:12.137Z",
                    "motivazione": "nuova motivazione"
              }
            }
```

2. Response

```
        HTTP/1.1 415 Unsupported Media Type
        Accept-Patch: application/merge-patch+json
```

Di seguito un esempio di cancellazione di una specifica prenotazione.

1. Request

```
        DELETE /rest/appuntamenti/v1/municipio/{id_municipio}/ufficio/{id_ufficio}/prenotazioni/12323254
HTTP/1.1
        Host: api.ente.example
```

2. Response

```
        HTTP/1.1 200 OK
```

Capitolo 7

Richiesta Risorse Massive

Lo sviluppo di una interfaccia per la richiesta di risorse massive è auspicabile nei casi in cui:

- Il numero di risorse richieste è elevato e le stesse possono essere aggregate in un'unica risposta.
- Ad una richiesta corrisponde una risposta di dimensioni considerevoli.
- Il contesto rende poco stabile il canale di comunicazione e si vuole fornire flessibilità al fruitore nella gestione delle fluttuazioni dello stesso.

L'erogatore DOVREBBE adottare interfacce per la richiesta di risorse massive sulla base di proprie valutazioni dei criteri sopra indicati con l'obiettivo di rendere più efficace ed efficiente il recupero della stesse ai fruitori.

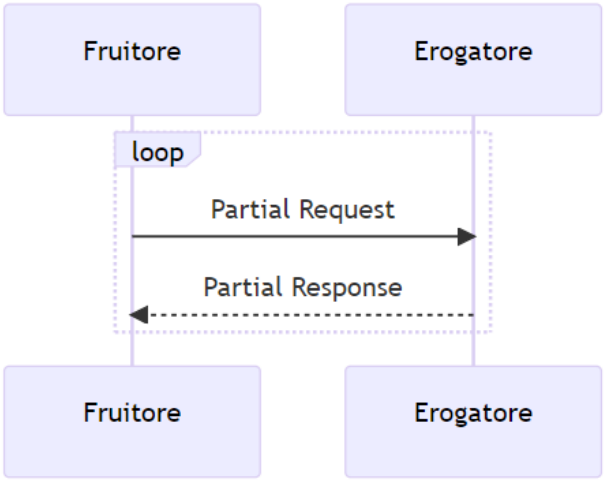

Figura 6 - Interazione richiesta/risposta parziale

In questo pattern l'erogatore rende disponibile ai fruitori la possibilità di realizzare richieste parziali di una risorsa per recuperare una porzione della stessa.

Il sequence diagram descrive un loop di request/response in cui il fruitore richiede porzioni di risorse e l'erogatore risponde con delle risposte parziali relative della risorsa richiesta.

La richiesta DEVE:

- Corrispondere ad una operazione di Read sulla risorsa richiesta.
- Contenere un percorso che indica un identificativo di risorsa oppure di una collezione di risorse.

7.1 [BULK_RESOURCE_REST] Richiesta Risorse Massive REST

L'implementazione in tecnologia REST utilizza la specifica Range Request definita nell'RFC 9110.

La specifica dell'interfaccia DEVE dichiarare tutti i codici di stato HTTP previsti dall'interfaccia, con il relativo schema della risposta, oltre che ad eventuali header HTTP restituiti.

Il pattern è indipendente dalla representation della risorsa restituita, in quanto le richieste sono di tipo byte-range, ovvero la Range Unit è pari a "bytes".

Il pattern ammette l'utilizzo dei seguenti Range Specifier:

- int-range
- suffix-range così come indicato in 14.1.2. Byte Ranges dell' RFC 9110.

L'erogatore DEVE restituire l'header Accept-Range pari a "bytes" ad un richiesta per la risorsa con metodo HEAD.

L'erogatore DEVE rispondere ad una richiesta parziale con lo status code 206 Partial Content assicurando il popolamento dell'header Content-Range pari all'intervallo parziale della risorsa restituita e dell'header Content-Length pari alla numero di octets della risposta.

L'erogatore DEVE rispondere ad una richiesta del fruitore di un range della risorsa errato lo status code 416 Range Not Satisfiable assicurando il popolamento dell'header Content-Range pari alla lunghezza corrente della risorsa richiesta.

L'erogatore DEVE rispondere ad una richiesta totale della risorsa con lo status code 200 OK.

Il fruitore per effettuare una richiesta parziale DEVE assicurare il popolamento dell'header Range pari alla porzione della risorsa richiesta.

7.1.1 Regole di processamento

Al termine del processamento della richiesta, l'erogatore DEVE fare uso dei codici di stato HTTP rispettando la semantica. 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 fornendo nel body di risposta dettagli circa l'errore;
- DOVREBBE, in caso di rappresentazione semanticamente non corretta, ritornare HTTP status 422 Unprocessable Entity;
- DOVREBBE, se qualcuno degli ID nel path non esiste, restituire HTTP status 404 Not Found, indicando nel body di risposta quale degli ID è mancante;
- PUÒ, se ipotizza che la richiesta sia malevola, ritornare HTTP status 400 Bad Request o HTTP status 404 Not Found;
- DEVE, in caso di errori non dipendenti dalla richiesta, restituire HTTP status 5xx rispettando la semantica degli stessi;
- DEVE, rifiutare richieste con Range Specifier non valido nell'header Range della richiesta, e ritornare HTTP status 416 Range Not Satisfiable;
- DEVE, restituire l'intera risorsa richiesta nel caso in cui il fruitore non popoli l'header Range, e ritornare HTTP status 200;
- DEVE, in caso di successo, restituire HTTP status 206 Partial Content inviando la porzione, o le porzioni in casi di richiesta di porzioni multiple, di representation della risorsa richiesta corrispondenti al/ai range indicati nella richiesta nel payload body;
- DOVREBBE, restituire l'header Accept-Range pari a "bytes" ad una richiesta per la risorsa con metodo HEAD.

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

7.1.2 Esempio

Specifica Servizio https://api.ente.example/rest/range-req/v1/RESTRangeRequest.yaml

```yaml
openapi: 3.0.1
info:
  title: Richieste Massive REST
  version: "1.0"
  description: |-
    Questo file descrive semplicemente un'API di esempio
    e non è da considerare esaustivo di tutte le informazioni
           di metadatazione eventualmente richieste
  license:
    name: Apache 2.0 License
    url: http://www.apache.org/licenses/LICENSE-2.0.html
paths:
  /resource/{id_res}:
    head:
      description: Check supporto Range Request
      responses:
        '200':
          description: Range Request supportate.
          headers:
            Accept-Ranges:
              description: Range unit supportata
              schema:
                type: string
                enum:
                  - bytes
                example: bytes

        '400':
           $ref: '#/components/responses/400BadRequest'
        '404':
           $ref: '#/components/responses/404NotFound'
        default:
           $ref: '#/components/responses/default'
    get:
      description: Recupero risorsa parziale
      responses:
        "206":
          $ref: '#/components/responses/206PartialContent'
        '400':
           $ref: '#/components/responses/400BadRequest'
        '404':
           $ref: '#/components/responses/404NotFound'
        "416":
           $ref: "#/components/responses/416RangeNotSatisfaible"
```

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


      parameters:
        - name: id_res
          in: path
          required: true
          schema:
            type: integer
            format: int32

  components:
    responses:
      206PartialContent:
        description: porzione di risorsa richiesta
        headers:
            Content-Range:
                description: espressione del range incluso nella risposta
                schema:
                    type: string
                    example: bytes 21010-47021/47022
      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'

      416RangeNotSatisfaible:
          description: richiesta di range non soddisfacibile
          headers:
              Content-Range:
                  description: lunghezza della representation richiesta
                  schema:
                      type: string
                      example: bytes */47022

      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:
```

```
    ErrorMessage:
      type: object
      properties:
        detail:
          description: |
            Una spiegazione human-readable specifica per questa occorrenza del problema.
          type: string
        instance:
          description: |
            Un URI assoluto identificativo della specifica occorrenza del problema.
            Può o non può fornire ulteriori informazioni se dereferenziato.
          format: uri
          type: string
        status:
          description: |
            Il codice di stato HTTP generato dal server di origine per questa occorrenza
            del problema
          format: int32
          maximum: 600
          minimum: 100
          type: integer
        title:
          description: |
            Un breve riepilogo del tipo di problema. Scritto in inglese e leggibile
            per ingegneri (di solito non adatto a parti interessate non tecniche e
            non localizzato); esempio: Servizio non disponibile
          type: string
        type:
          default: about:blank
          description: |
            Un URI assoluto che identifica il tipo di problema. Quando dereferenziato,
            DOVREBBE fornire una documentazione human-readable per il tipo di problema
            (ad esempio, utilizzando HTML).
          format: uri
          type: string
```

Di seguito un esempio di chiamata per la verifica del supporto alle range request.

1. Request

```
HEAD /rest/range-req/v1/municipio/{id_res} HTTP/1.1
Host: api.ente.example
```

2. Response

```
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 25000
Accept-Ranges: bytes
```

Di seguito un esempio di chiamata per il recupero parziale della risorsa.

Di seguito un esempio di chiamata per la verifica del supporto alle range request.

1. Request

```
GET /rest/range-req/v1/municipio/{id_res} HTTP/1.1
Host: api.ente.example
Range: bytes=0-999
```

2. Response HTTP status 206 Partial Content

```
HTTP/1.1 206 Partial Content
Content-Type: text/plain
Content-Length: 1000
Content-Range: bytes 0-999/25000

*1000 bytes of data are included in the message body*
```

2. Response HTTP status 400 Bad Request

```
HTTP/1.1 400 Bad Request
Content-Type: application/problem+json

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

2. Response HTTP status 404 Not Found

```
HTTP/1.1 404 Not Found
Content-Type: application/problem+json

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

```
        }
```