ISSN 2039-7941
anno 2022 numero 447

RAPPORTI TECNICI INGV

REfAsE - Rest Engine for Asynchronous job Execution: un motore software modulare RESTful per il dispatch e processing asincrono di job eterogenei

ISTITUTO NAZIONALE DI GEOFISICA E VULCANOLOGIA

447

REGISTRAZIONE AL TRIBUNALE DI ROMA N.174 | 2014, 23 LUGLIO

ISTITUTO NAZIONALE DI GEOFISICA E VULCANOLOGIA

ISSN 2039-7941
anno **2022** numero **447**

RAPPORTI TECNICI INGV

REfAsE - Rest Engine for Asynchronous job Execution: un motore software modulare RESTful per il dispatch e processing asincrono di job eterogenei

A modular RESTful software engine for asynchronous dispatching and processing of heterogeneous jobs

Alessandro Di Filippo, Rosario Peluso

INGV | Istituto Nazionale di Geofisica e Vulcanologia, Sezione di Napoli - Osservatorio Vesuviano

Accettato 10 novembre 2021 | *Accepted 10 November 2021*
Come citare | *How to cite* Di Filippo A., Peluso R., (2022). REfAsE - Rest Engine for Asynchronous job Execution: un motore software modulare RESTful per il dispatch e processing asincrono di job eterogenei. Rapp. Tec. INGV, 447: 1-32, https://doi.org/10.13127/rpt/447

In copertina Astrazione di un motore software: ogni componente singolo contribuisce al corretto funzionamento di tutto il sistema, come in una catena di ingranaggi | *Cover Software engine abstraction: each single module concur to the smooth functioning of the global system, as in a gears chain*

447

INDICE

Riassunto 7

Abstract 7

Introduzione 8

1. Specifiche tecniche 9
1.1 Contesto operativo 9
1.2 Requisiti 9

2. Implementazione 10
2.1 Approccio API-driven 10
2.2 Modello RESTful 11
2.3 RESTalk *Domain Specific modeling Language* 12
2.4 *Long Running Operation with Polling* (LROP) 14
2.5 Scelte tecnologiche 16
2.5.1 Web Server Gateway Interface 16
2.5.2 FLASK framework 17
2.5.3 REDIS 17
2.5.4 REDIS queue 18

3. Protocollo e punti di accesso 19
3.1 Struttura del protocollo 20
3.2 Risorse 20
3.2.1 Risorsa principale 20
3.2.2 Risorse *list* 21
3.2.3 Risorse *worker* 21
3.2.4 Risorse *status* 22
3.2.5 Risorse *result* 22
3.3 *Worker* correntemente implementati 23
3.3.1 Generatore dei bollettini sismici periodici 23
3.3.2 Comunicati di evento sismico 25
3.3.3 Mappe e grafici 26

4. Integrazione e messa in servizio 27

5. Conclusioni e sviluppi futuri 28

Bibliografia 28

Riassunto

Nell'ambito delle operazioni e funzionalità relative alla gestione di un evento sismico e delle attività collegate, è stato sviluppato il motore "REfAsE" - *Rest Engine for Asynchronous job Execution*. Il *software*, progettato ed implementato integralmente all'interno dell'Unità Funzionale "Sala di Monitoraggio ed IT" dell'Osservatorio Vesuviano di Napoli, assume il ruolo di *"middleware"* e *"broker"* per quella parte di carico elaborativo necessario alla produzione dei comunicati degli eventi sismicisopra soglia, dei grafici e delle mappe utilizzate per la creazione dei bollettini sismici periodici e delle relazioni automatiche di evento. L'intento primario è stato quello di "isolare" WESSEL (*WEb Service for Seismic Event Location*, il sistema che permette la gestione e l'analisi di tutte le fasi della localizzazione degli eventi sismici dei vulcani campani) sganciando tali funzionalità dal flusso di elaborazione principale ed implementando quelle aggiuntive con approccio granulare ed atomico.
Grazie ad un'interfaccia ed un protocollo RESTful che permettono di gestire completamente le risorse mediante comandi HTTP (*HyperText Transfer Protocol*), è così possibile sottomettere dei *job* ed ottenerne i relativi risultati in maniera asincrona, senza che il costo computazionale gravi sul sistema principale. L'idea che ha guidato la progettazione di REfAsE è stata quindi quella di realizzare un motore *software* capace di prendere in carico tutte quelle richieste che è possibile soddisfare anche in tempo non reale, ovvero accettando dei *job* (quindi dei compiti da portare a termine) che richiedano risorse e tempo di calcolo da non "sottrarre" a WESSEL, inteso sia come sistema *software* sia come infrastruttura *hardware* di calcolo che lo mantiene in produzione.
REfAsE, progettato per essere modulare, scalabile ed integrabile, attualmente implementa vari scenari operativi:

- la produzione di comunicati per il Dipartimento di Protezione Civile (DPC);
- la creazione dei bollettini sismici periodici;
- la creazione di immagini e grafici per le relazioni automatiche.

Sono già in corso di sviluppo i protocolli e le implementazioni che permetteranno a REfAsE di fungere da provider di diversi software per la localizzazione e analisi di eventi sismici come Hypo71, Localmag, NonLinLoc, etc.

Abstract

The REfAsE - Rest Engine for Asynchronous job Execution - system is positioned within the operations and functionalities related to the management of a seismic event and related activities. The software, designed and implemented entirely within the Functional Unit "Monitoring Room and IT", has taken on the role of "middleware" and "broker" for that part of the processing load necessary to the production of reports of above threshold events, graphs and maps used for the creation of periodic seismic bulletins and automatic reports. The primary intent was to "isolate" WESSEL (WEb Service for Seismic Event Location, the system that allows the management and analysis of all phases of localization of seismic events of neapolitan volcanoes) detaching these features from the main processing flow and implementing the additional ones with granular and atomic approach.
Thanks to its RESTful interface and a simple protocol allowing to completely manage the resources through HTTP (HyperText Transfer Protocol) commands, it is possible to submit jobs and obtain the related results in an asynchronous way, without the computational cost burdening the main system. The idea that guided the design of REfAsE was therefore to create a software engine capable of taking charge of all those requests that it is possible to satisfy even in non-real time, i.e. accepting jobs (tasks to be completed) that require resources and computation time that should not be

"subtracted" from WESSEL, intended both as a software system and as a computing hardware infrastructure that keeps it in production.
REfAsE, designed to be modular, scalable and integrable, currently implements several operational scenarios:

- *production of releases for the Civil Protection Department (DPC);*
- *creation of periodic seismic bulletins;*
- *creation of maps and graphs for the automatic reports.*

At the moment there are other protocols and implementations under development that will allow REfAsE to act as a provider for different seismic location softwares as Hypo71, Localmag, NonLinLoc, etc.

Keywords Restful; Asincrono; Motore software | Asynchronous; Software Engine

Introduzione

L'Osservatorio Vesuviano, sezione di Napoli dell'INGV, si occupa del monitoraggio dei tre vulcani Campani (Vesuvio, Campi Flegrei ed Ischia): in questo ambito si inserisce la sorveglianza sismica svolta in Sala Operativa da due unità di personale per ognuno dei tre turni giornalieri.
A partire dal 2017 si sta provvedendo ad un aggiornamento generale dei sistemi a supporto del personale turnista. In quest'ambito è iniziato lo sviluppo dei sistemi SERENADE (*SEismic Restful ENAbled DatabasE)* [Peluso et. Al, 2020b] e WESSEL (*WEb Service for Seismic Event Location*) [Peluso et. Al, 2020a], sviluppo che è poi confluito nelle attività del progetto FISR S.O.I.R. (Sale Operative Integrate e Reti di monitoraggio del futuro [AA. VV., 2020]). WESSEL è stato progettato ed implementato con lo scopo di essere un unico portale web in grado di permettere l'analisi e la gestione degli eventi sismici da parte del personale preposto, esponendo una interfaccia web unificata. Esso permetterà la sostituzione dei sistemi storicamente utilizzati nelle attività di sorveglianza sismica ed ammodernare l'architettura *software* della Sala Operativa.
Nell'ambito delle attività di riorganizzazione dei sistemi a supporto del personale turnista e del personale monitorante, si posiziona il motore REfAsE - *Rest Engine for Asynchronous job Execution*, oggetto della trattazione nel presente lavoro. Questo *software*, progettato ed implementato integralmente all'interno dell'Unità Funzionale "Sala di Monitoraggio ed IT", ha assunto il ruolo di *"middleware"* e *"broker"* per quella parte di carico elaborativo necessario alla produzione dei comunicati degli eventi sopra soglia[1], dei grafici e delle mappe utilizzate per la creazione dei bollettini sismici periodici e delle relazioni automatiche di evento per i reperibili.
L'intento primario è stato quello di "isolare" WESSEL sganciando tali funzionalità dal flusso di elaborazione principale (la localizzazione e l'inserimento degli eventi nel database) implementando quelle aggiuntive con approccio granulare ed atomico. Grazie ad un'interfaccia ed un protocollo RESTful che permettono di gestire le risorse completamente mediante comandi HTTP (*HyperText Transfer Protocol*), è così possibile sottomettere dei *job* ed ottenerne i relativi risultati in maniera asincrona, senza che il costo computazionale gravi su WESSEL.

[1] In futuro permetterà di gestire anche l'occorrenza di sciami di eventi, così come definiti nelle "Procedure Operative per i Comunicati" redatte secondo l'accordo-quadro INGV-DPC.

1. Specifiche tecniche

1.1 Contesto operativo

L'idea che ha guidato la progettazione di REfAsE è stata quella di affiancare a WESSEL un motore *software* capace di prendere in carico tutte quelle richieste che è possibile soddisfare anche in tempo non reale, ovvero accettando dei *job* (quindi dei "compiti" da portare a termine) che richiedono risorse e tempo di calcolo da non "sottrarre" a WESSEL, inteso sia come sistema *software* sia come infrastruttura *hardware* di calcolo che lo mantiene in produzione.

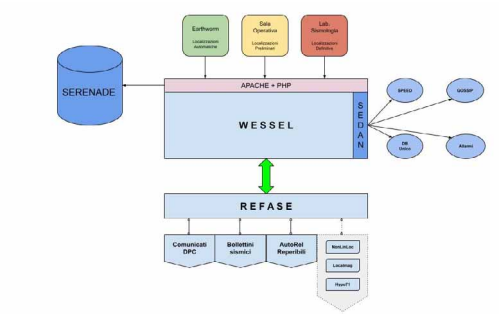


Figura 1 WESSEL e REfAsE nell'architettura globale del software di Sala Operativa.
Figure 1 *WESSEL and REfAsE in the global architecture of Monitoring Room software.*

Gli obiettivi principali che hanno guidato la sua realizzazione si possono riassumere, quindi, nei seguenti punti:

- isolare i sistemi principali di Sala Operativa sganciando le funzionalità secondarie dal flusso di elaborazione principale;
- mantenere coerenza di presentazione utilizzando il *client* predefinito con interfaccia ben nota al personale monitorante;
- implementare le funzionalità aggiuntive con approccio granulare ed atomico, realizzando risorse di calcolo autoconsistenti.

Mentre il contesto operativo, ovvero l'ambito di business in cui il sistema è chiamato ad operare, può essere delineato dalle seguenti attività:

- attività ordinarie di sorveglianza sismica e vulcanica h24;
- attività pianificate di produzione di documentazione di approfondimento delle attività di monitoraggio;
- attività event-driven di comunicazione istituzionale (verso il Dipartimento della Protezione civile, altri enti, etc.).

REfAsE, quindi, si comporta a tutti gli effetti come un espositore di microservizi in grado di prendere in carico una serie di operazioni computazionalmente pesanti che possono essere facilmente incapsulate e poste in flussi di elaborazione separati, in modo da mantenere inalterato il livello di disponibilità[2] dei sistemi principali.

1.2 Requisiti

Tra le richieste iniziali che hanno guidato lo sviluppo del sistema, sono maturati due requisiti in particolare che più di altri hanno contribuito alla strutturazione di REfAsE:

[2] La disponibilità misura l'attitudine di un'entità o sistema ad essere in grado di svolgere una funzione richiesta in determinate condizioni ad un dato istante (norma UNI 10147).

1. totale indipendenza dal *client* che utilizza le risorse di calcolo esposte;
2. protocollo di comunicazione sufficientemente generico, tale da permettere una semplice estensione in caso di aggiunta di nuove funzionalità.

La prima risorsa computazionale che ci si è trovati ad implementare è stata quella necessaria alla generazione di tutte le informazioni (immagini, tabelle ed altro) per la produzione dei bollettini sismici periodici inviati dall'Osservatorio Vesuviano al Dipartimento di Protezione Civile (DPC). In una fase successiva sono state poi implementate le risorse necessarie alla generazione dei comunicati da inviare in caso di accadimenti di singoli eventi sismici. Poiché, al momento della stesura di questo rapporto, né Wessel né Serenade supportano la gestione di sciami sismici, questa risorsa non è ancora stata implementata.
Parallelamente allo sviluppo di quanto appena descritto, è stato messo anche a punto un altro servizio per la generazione delle relazioni automatiche a servizio dei Reperibili Sismologi da comunicare in caso di eventi di magnitudo superiore a 4.0. Al momento questo servizio non è fornito integralmente da REfAsE ma, grazie alla struttura estremamente agile ed al protocollo facilmente estensibile, è stato possibile riutilizzare il codice scritto per la creazione dei prodotti che compongono i bollettini per la generazione delle mappe necessarie alla composizione di tale relazione. Un obiettivo futuro è quello di inglobare anche la composizione delle relazioni direttamente in REfAsE.

2. Implementazione

L'architettura progettata, ispirata dai requisiti illustrati in precedenza, è stata basata su quattro punti chiave:

1. approccio API-driven;
2. modello RESTful;
3. RESTalk *Domain Specific modeling Language* (DSL);
4. *Long Running Operation with Polling* (LROP).

L'ultimo punto, in particolare, è stato necessario per permettere lo sviluppo di un'interfaccia reattiva nel *client*, in modo da evitare che lunghe pause di elaborazione potessero essere confuse con malfunzionamenti del sistema. Di seguito si presenta lo stato dell'arte su questi temi e come tali punti sono stati utilizzati per creare l'architettura con cui REfAsE è stato implementato.

2.1 Approccio API-driven

Come è noto, le Application Programming Interfaces (API) permettono a diversi sistemi, anche basati su tecnologie eterogenee, di interagire tra loro garantendo l'accesso remoto ai servizi che questi espongono e rendono disponibili all'esterno. Il vantaggio principale è che questo tipo di approccio non richiede una profonda conoscenza del sistema che realizza tali servizi e che essi stessi possono essere consumati effettivamente soltanto quando sono necessari, ovvero quando sono richiesti dal sistema "consumatore".
Le tecnologie emergenti, i servizi *cloud* i service *mashup* e le architetture a microservizi, fanno tutte intensivamente uso di API. Quest'approccio ha sensibilmente migliorato il *deployment* dei sistemi non solo in termini di *design* e prestazioni, ma soprattutto in termini di usabilità, tant'è che l'ingegneria del *software* ha aumentato sempre di più l'interesse verso lo sviluppo *API-driven*, dove il ruolo chiave dello sviluppo è rappresentato dall'API stessa.

Le API possono essere progettate utilizzando diversi stili e protocolli, come Remote Procedure Calls (RPC), Simple Object Access Protocol (SOAP) e REpresentational State Transfer (REST), quest'ultimo utilizzato proprio dal protocollo di REfAsE.

2.2 Modello RESTful

REST è uno stile architetturale presentato per la prima volta nella tesi di dottorato di Roy Fielding nel corso dell'anno 2000 [Fielding, 2000]. Le API REST sono strutturate intorno ad entità chiamate *risorse* che implementano funzionalità ben definite (ed atomiche) ed isolano informazioni di potenziale interesse per gli utenti dell'API esponendole come servizi (spesso chiamati *endpoint*). A tali risorse si accede tramite Uniform Resource Identifier (URI), utilizzati proprio per esporre l'informazione che viene fornita come rappresentazione di dati, metadati e collegamenti ad altre risorse.
Affinché un'API sia considerata RESTful, deve essere conforme a questi criteri di base:

- architettura *client-server* composta da *client*, *server* e risorse, con richieste gestite integralmente tramite HTTP;
- comunicazione *client-server stateless*, ovvero nessuna informazione del cliente viene memorizzata tra le richieste ed ogni richiesta è atomica;
- dati memorizzabili in *cache* che semplificano le interazioni *client-server*;
- un sistema stratificato capace di organizzare gerarchicamente ogni tipo di *server* coinvolto nell'architettura informativa in modo del tutto trasparente per il *client*, così che questi non abbia la necessità di conoscere da quale *server* applicativo deve provenire la risposta o se il soddisfacimento di una particolare richiesta coinvolge più servizi/*server*;
- un'interfaccia uniforme, trasversale ai diversi componenti, in modo tale che la comunicazione tra i peer coinvolti sia quanto più possibile standardizzata.

L'ultimo aspetto a sua volta richiede che:

- le risorse richieste siano identificabili e separate dalle rappresentazioni inviate al *client*;
- le risorse possano essere manipolate dal *client* attraverso la rappresentazione che ricevono poiché questa contiene le informazioni necessarie e sufficienti;
- i messaggi autodescrittivi restituiti al *client* abbiano abbastanza informazioni per descrivere come il *client* debba elaborarli;
- le informazioni siano ipermediali, ovvero accedendo alla risorsa/*endpoint* il *client* deve poter individuare, seguendo uno o più *hyperlink*, tutte le altre opzioni disponibili al momento.

In merito all'approccio *stateless*, è possibile dire che le richieste del *client* debbano essere autocontenute, in modo tale che il *server* non abbia bisogno di ricordare le interazioni precedenti. Questo implica che ogni interazione è sempre iniziata dal *client*, che invia una nuova richiesta ogni volta che è pronto ad avanzare nello stato della conversazione. Il *client* inizia la conversazione mirando a un certo obiettivo (la risorsa da consumare) e può terminarla in qualsiasi momento semplicemente cessando di inviare ulteriori richieste.
Tuttavia, la responsabilità di guidare il flusso informativo della conversazione non ricade soltanto sul *client*: anche il *server* determina quali collegamenti a risorse correlate inviare, se ce ne sono, a seconda dello stato della risorsa richiesta e nulla viene inviato se la richiesta non è autorizzata o se non ci sono collegamenti da scoprire (ad esempio, dopo una richiesta DELETE).
D'altro canto, è comunque il *client* a decidere se e quali collegamenti seguire: essi si riferiscono agli URI che sono utilizzati per identificare in modo univoco le risorse. Nel modello RESTful il

vincolo conosciuto come HATEOAS *(Hypermedia As The Engine of Application State)* richiede che il *client* non sia a conoscenza della loro struttura. Tutto ciò che il *client* deve conoscere è quindi l'URI principale per l'accesso ai servizi esposti, mentre tutti gli altri URI sono scoperti dinamicamente durante la conversazione. In questo modo il *client* è effettivamente disaccoppiato dal *server*, permettendo di evolvere la logica applicativa senza bisogno di coinvolgerne i fruitori (ovvero i *client* stessi) al netto degli *endpoint* che espongono le risorse. Data la natura sincrona delle interazioni RESTful, le richieste sono sempre seguite da una risposta. Solo in caso di fallimento, ovvero quando il *server* non è disponibile o per qualche motivo il messaggio di richiesta non viene processato, il *client* potrebbe reinviare la richiesta dopo un determinato *timeout*. In quest'ultimo caso, l'eventuale idempotenza[3] del metodo HTTP utilizzato è importante: se la richiesta rinviata non è idempotente, ad esempio nel caso di una richiesta POST, l'API deve essere progettata specificamente per affrontare e gestire una tale azione. Ovviamente le risorse non subiscono alcuna conseguenza dal reinvio di richieste idempotenti come GET, PUT e DELETE. Quando il *client* effettua una richiesta tramite una API RESTful, viene trasferita una rappresentazione dello stato della risorsa dall'*endpoint* che la espone al *client* stesso. Questa informazione, ovvero questa rappresentazione dello stato, viene consegnata in uno dei diversi formati utilizzabili tramite HTTP: JSON, HTML, XLT, Python, PHP ed anche testo semplice. Il JSON (Javascript Object Notation) è il formato più comunemente usato perché, nonostante il suo nome, è indifferente al linguaggio e comprensibile sia dagli uomini che dalle macchine.
Ciò che rende preferibile l'adozione del modello RESTful, nonostante la serie di requisiti cui esso debba essere conforme, è che il suo impiego è comunque considerato più semplice rispetto a quello di un protocollo prescrittivo come SOAP (che presenta requisiti specifici come la messaggistica XML e la conformità integrata di sicurezza e transazioni) considerando anche il supporto a diversi formati di dati, un accoppiamento ai "motori" software degli *endpoint* più libero e più conciso nella forma e nell'utilizzo.

2.3 RESTalk *Domain Specific modeling Language*

Un *Domain Specific modeling Language* (DSL) è un piccolo linguaggio verticale, altamente specializzato, usato per modellare domini di *business* chiaramente identificabili, a differenza di un linguaggio di uso generale (*General Purpose Language* - GPL) che è pensato per un utilizzo non specialistico di tipo *cross domain*.
RESTalk [Pautasso, 2016; Ivanchikj, 2021] è un DSL per la rappresentazione delle conversazioni RESTful, implementato come un'estensione della *Business Process Modeling Notation* (BPMN) *Choreography Diagrams*[4] con dettagli specifici REST[5]. In Figura 2 si riportano i formalismi grafici utilizzati per la rappresentazione di questo tipo di diagrammi.
Nell'ambito di una conversazione RESTful, in RESTalk il contenuto del messaggio da includere nell'interazione tra *client* e *server* contiene i seguenti dettagli: metodo HTTP, URI, codice di stato della risposta ed, eventualmente, il *link*.
Un evento di inizio e uno di fine sono generalmente usati per rappresentare l'inizio e la fine della conversazione, mentre un evento *timer* è utilizzato come riferimento temporale e *timeout* della risposta nel caso in cui l'elaborazione del *server* richieda troppo tempo (in tal caso il *client* provvede a reinviare la richiesta al *server*).

[3] L'effetto di più richieste identiche è lo stesso di quello di una sola richiesta.

[4] http://blog.maxconsilium.com/2013/09/bpmn-20-models-part-2.html

[5] Nella sua versione attuale RESTalk supporta solo la modellazione di conversazioni RESTful uno a uno e non le conversazioni multi entità che saranno supportate in futuro.

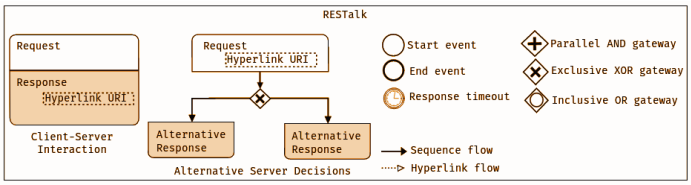


Figura 2 Formalismi grafici utilizzati per la rappresentazione grafica di un BPMN.
Figure 2 *Graphic formalisms for a generic BPMN representation.*

In Figura 3 è mostrato il *sequence diagram* che schematizza il flusso globale della conversazione, nel quale si inserisce anche il flusso dello *scouting* dei collegamenti ipertestuali da parte del *client* che utilizza così gli URI per accedere alle risorse.

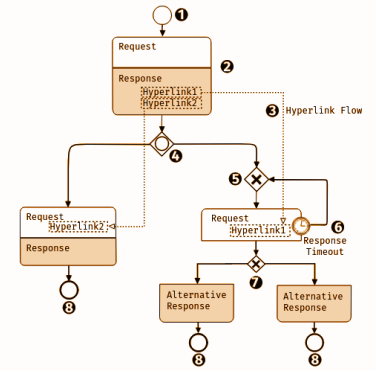


Figura 3 Esempio di conversazione REST.
Figure 3 *REST conversation example.*

Nell'esempio mostrato[6] in Figura 3, il *client* inizia la conversazione (1) inviando una richiesta alla quale il *server* risponde (2) con due URI che possono essere seguiti dal *client* (3) e che sono relativi alla risorsa richiesta. Il *client* può decidere di seguire solo il primo *hyperlink* o solo il secondo *hyperlink* o entrambi (4). Se il *server* non risponde in tempo (6) alla richiesta del *client* per il primo collegamento ipertestuale, il *client* può inviare nuovamente la richiesta (5). Il *server* può rispondere con una delle due alternative (7) a seconda dello stato della risorsa richiesta. Una volta ricevuta la risposta, la conversazione termina (8).
Per modellare le conversazioni, RESTalk utilizza alcuni principi di base:

- anche se il *client* può terminare la conversazione in qualsiasi momento, non inviando ulteriori richieste, si utilizzano gli eventi finali per modellare solo i percorsi che portano al successo o al fallimento dell'intento iniziale della conversazione;
- viene rappresentato solo il flusso del collegamento ipertestuale dell'ultima risposta ricevuta, mentre in realtà il collegamento potrebbe essere stato ottenuto anche prima nel corso della conversazione;
- a causa dell'assenza di conseguenze nel reinvio di richieste idempotenti, si modella esplicitamente soltanto il reinvio di richieste non idempotenti (POST, PATCH) senza enfatizzare il fatto che il *client* potrà eventualmente rinunciare e smettere di reinviare la richiesta;
- le risposte con codici di stato di classe 5xx (errore del *server*, *service unavailable*, etc.) poiché possono verificarsi dopo qualsiasi richiesta, non vengono modellate esplicitamente, anche se incluse nel flusso della conversazione.

[6] Come si può notare, i gateway logici (AND, OR e XOR) sono utilizzati per far convergere o divergere i diversi percorsi plausibili del flusso della conversazione REST. Possono essere di diverso tipo, sia paralleli quando sono necessari percorsi concomitanti, sia esclusivi quando solo uno di n percorsi è possibile. Possono anche essere inclusivi quando può verificarsi qualsiasi combinazione.

Gli *endpoint* esposti come risorse della API RESTful vengono "scoperti" dai *client* attraverso conversazioni che si dettagliano a mano a mano che l'interazione si sviluppa partendo dal primo contatto fino a raggiungere diversi obiettivi, come ad esempio: la creazione di risorse aggiuntive, la scoperta di risorse correlate, l'enumerazione degli elementi trovati all'interno di una collezione di dati o di una lista di ulteriori risorse.
RESTalk, essendo un piccolo linguaggio specifico per la modellazione di conversazioni REST, è dotato di *pattern* che possono essere seguiti ed applicati per risolvere problemi chiave in domini specifici. In fase di progettazione dell'architettura *software* che si vuole modellare, è quindi possibile individuare e scegliere uno dei *pattern* che meglio si applica al dominio per cui si sta costruendo il set di API REStful.
L'utilizzo di un *pattern* disponibile, considerato una *best practice* dell'ingegneria del *software*, permette quindi di concentrarsi sugli aspetti specifici delle risorse da progettare, sui requisiti di *business*, sull'insieme di funzionalità che si vorranno implementare e sulle caratteristiche dei *client* che realizzeranno la conversazione REST, senza la necessità di rimodellare gli aspetti generali per il dominio di applicazione.
Per la progettazione del modello RESTful di REfAsE sono stati utilizzati diversi *pattern*, tra cui il *Long Running Operation with Polling*, che ricade nell'insieme di *pattern* "creazionali" di RESTalk[7], utilizzati per modellare la creazione di risorse rispettando vincoli e scenari di fallimento ben definiti.

2.4 *Long Running Operation with Polling* (LROP)

Possiamo definire un *pattern* come:

> *"un modello descrittivo della soluzione ad un problema ricorrente nell'ambito di un dominio applicativo o di business che può essere utilizzato n volte con successo senza necessità di riprogettare ogni volta la soluzione stessa"*[8]

e affermare, quindi, che LROP sia stato determinante per ideare, progettare ed implementare l'architettura generale ed il modello di funzionamento di REfAsE.
Nello specifico, LROP si pone l'intento di utilizzare il *polling* per evitare i *timeout* del *client* quando vengono consumate risorse che forniscono un risultato dopo un tempo di elaborazione non trascurabile, anche chiamati *long running task*, ovvero letteralmente operazioni di lunga durata. Infatti, l'elaborazione di operazioni complesse o ad alto carico computazionale potrebbe richiedere molto tempo e "bloccare" il *client* che ha effettuato la richiesta in una attesa anche molto lunga. In questo caso, il *client* risulterebbe indisponibile ad altre operazioni proprio perché in attesa della risposta da parte della risorsa.
Inoltre, poiché la rete per definizione non è affidabile[9], il *client* può perdere la connessione prima che il *server* abbia completato l'elaborazione del risultato. Più tempo impiega il *server* a rispondere al *client*, più alte sono le possibilità che il *client* non sia più disponibile a ricevere il risultato o interessato a recuperarlo. Per esempio, il *server* potrebbe aver bisogno di tempo per soddisfare le richieste del *client* a causa di una catena di calcolo eterogenea che aumenta la complessità computazionale. In questo caso, se la connessione relativa alla richiesta effettuata dal *client* va in *timeout*, bisognerebbe ripetere tutta l'elaborazione, con notevole dispendio di risorse.

[7] Qui per "*pattern* creazionali" non si intendono quelli comunemente trattati nell'ingegneria del *software*, dove vengono utilizzati per astrarre il processo di istanziazione, rendere il sistema indipendente da come gli oggetti sono creati e rappresentati e dalle relazioni di composizione tra essi e nascondere come le istanze delle classi sono realmente create, incapsulando la conoscenza relativa all'implementazione specifica utilizzata dal sistema.

[8] Christopher Alexander et al., (1977), in *A Pattern Language: Towns, Buildings, Construction*.

[9] Concetto di "best effort networking" nel dominio IP.

Ottenere il risultato di una tale operazione senza restare in attesa e tenere aperta la connessione HTTP per lungo tempo ed evitare di sprecare risorse per connessioni aperte e per elaborazioni complesse il cui risultato non sarà ricevuto dal *client* nel caso in cui la connessione vada in *timeout*, è possibile considerando che portare a termine una specifica richiesta computazionale e consegnarne i risultati sono due compiti ben definiti e distinti con un peso completamente diverso sull'infrastruttura del sistema, sia ai livelli logico-applicativi, sia al livello fisico.
L'operazione lunga e complessa viene trasformata in una risorsa, creata utilizzando la richiesta iniziale effettuata dal *client* all'*endpoint* che la espone come servizio, fornendo come risposta iniziale l'URI dove verranno esposti i risultati.
Il *client*, successivamente alla richiesta iniziale fatta all'*endpoint* (quella che ha fatto partire il calcolo "lungo e complesso") può interrogare la risorsa per conoscere lo stato corrente dell'elaborazione, ovvero il suo attuale progresso, ed una volta che l'operazione di lunga durata è stata completata, verrà reindirizzato all'*endpoint* di un'altra risorsa che espone il risultato.
Poiché il risultato finale dell'elaborazione avrà un proprio URI, è possibile recuperarlo più volte, a patto che non sia stato cancellato con una esplicita richiesta di DELETE o a seguito di precise politiche di storage implementate nell'architettura. Inoltre, l'operazione di lunga durata può essere annullata con una successiva richiesta di DELETE, interrompendo l'elaborazione sul *server*, o cancellando il suo *output* se nel frattempo è già stato completato.
Una schematizzazione della conversazione è rappresentata dal diagramma in Figura 4. Come si può notare, effettivamente il *client* non ha bisogno di tenere aperta la connessione con il *server* per tutta la durata della richiesta, permettendo così al *server* di gestire e soddisfare più richieste da parte di più *client*.

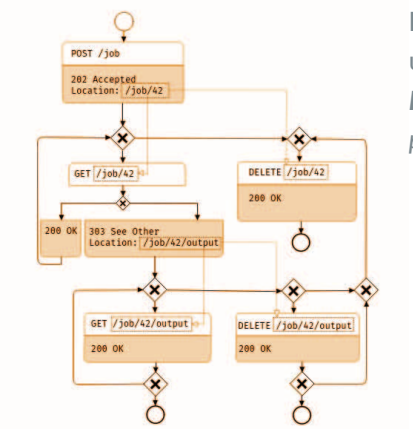


Figura 4 Schematizzazione di una conversazione che utilizza il pattern LROP.
Figure 4 *Schematic of a conversation using the LROP pattern.*

Un altro aspetto importante è che i risultati dell'elaborazione vengano resi disponibili in modo persistente: Il *link* alla risorsa risultato può essere condiviso con più *client* che possono recuperarlo senza che il *server* debba ricalcolarlo (ovviamente vale per il risultato di una specifica richiesta). Viene inoltre contemplato un meccanismo esplicito coerente con l'interfaccia REST per cancellare le richieste ed evitare così di sprecare risorse.
Per contro, i punti critici possono essere individuati come:

- ***Polling***: il *client* deve implementare il *polling* che, se eseguito con cadenza troppo ravvicinata, può gravare ulteriormente sul server e consumare banda. Per mitigare questo problema, potrebbe essere implementato un meccanismo che fornisca al *client* informazioni sull'avanzamento mentre viene effettuato il polling, in modo tale che il

numero di richieste GET sia adattivo rispetto allo stato dell'elaborazione[10];

- **Storage**: a seconda della dimensione del risultato e della numerosità di *client* serviti, lo spazio di archiviazione potrebbe rappresentare un collo di bottiglia per il corretto funzionamento di tutto il sistema. Per scongiurare l'esaurimento dello storage disponibile, dovranno essere implementate politiche per l'eliminazione dei risultati prodotti al superamento di un intervallo di tempo stabilito o di una soglia di spazio disco predefinita.

2.5 Scelte tecnologiche

Il linguaggio di programmazione scelto per l'implementazione di REfAsE è Python. È un linguaggio di alto livello che supporta diversi paradigmi di programmazione: object-oriented (con supporto all'ereditarietà multipla), imperativo e funzionale. Offre una tipizzazione dinamica forte ed è fornito di una libreria built-in estremamente ricca, che unitamente alla gestione automatica della memoria e a robusti costrutti per la gestione delle eccezioni fa di Python uno dei linguaggi più ricchi e flessibili. Per come il sistema nel suo complesso è stato modellato, si è avuta la necessità di scegliere le basi tecnologiche su cui sviluppare l'architettura, restando sempre nell'ambito dell'ecosistema Python. Sono state quindi individuate le scelte tecnologiche che hanno permesso di supportare al meglio la realizzazione di quanto progettato:

- il protocollo WSGI (Web Server Gateway Interface), interfaccia standard dei web service in Python;
- il framework FLASK, per realizzare applicazioni web che comunicano in WSGI;
- il DBMS (Data Base Management System) noSQL REDIS, per la realizzazione della "coda" che implementa la struttura del protocollo di LROP (*Long Running Operation with Polling*);
- la libreria RQ (Redis Queue), scritta sempre in Python, per la gestione della struttura *queue* di REDIS e la sua migliore integrazione nell'architettura asincrona dell'applicazione.

2.5.1 Web Server Gateway Interface

La Web Server Gateway Interface (WSGI) è una specifica che descrive come un server web comunica con le applicazioni e come le applicazioni stesse possano essere concatenate insieme per elaborare una richiesta. WSGI è uno standard Python descritto in dettaglio nella PEP[11] 3333. WSGI quindi non è un server, un modulo python, un framework, un'API o qualsiasi tipo di software: è soltanto una specifica di interfaccia che definisce come server e applicazione comunicano[12]. Se un'applicazione (un framework, un toolkit, ecc.) è scritta secondo le specifiche WSGI, allora essa funzionerà su un qualsiasi server scritto in aderenza a tali specifiche.
Le applicazioni conformi alla WSGI possono essere realizzate con approccio stacked. Quelle che si posizionano logicamente nello strato intermedio dello stack, comunemente conosciute come middleware, implementano entrambi i lati dell'interfaccia WSGI: applicazione e server. Un server WSGI, ovvero conforme a WSGI, riceve solo la richiesta dal *client*, la passa all'applicazione e poi invia la risposta restituita dall'applicazione al *client* che ha effettuato la richiesta: non fa nient'altro, tutti i dettagli implementativi devono essere forniti dall'applicazione o dal middleware.

[10] Per fare a meno del polling, il client potrebbe anche a sua volta esporre una risorsa che sia raggiungibile con un URI di callback fornito al server in fase di richiesta iniziale richiesta, dove vuole essere notificato quando il risultato dell'elaborazione risulta disponibile.

[11] PEP sta per Python Enhancement Proposal. Un PEP è un documento di progetto che fornisce informazioni alla comunità Python, o che descrive una nuova caratteristica per Python o i suoi processi o ambienti. Un PEP fornisce una concisa specifica tecnica ed una precisa motivazione per la caratteristica (o funzionalità) di cui tratta.

[12] L'interfaccia del server e dell'applicazione sono entrambe specificate nel PEP 3333.

2.5.2 FLASK framework

Flask è un micro-framework per applicazioni web WSGI scritto in Python. È stato progettato con l'intento di fornire ai programmatori uno strumento semplice ma allo stesso tempo molto potente per progettare e realizzare applicazioni web di qualsiasi complessità. Nato da un'idea di Armin Ronacher[13], si è evoluto nel tempo da semplice *wrapper* del WSGI Werkzeug ed il *template engine* Jinja, fino a diventare uno dei più popolari *framework* di applicazioni web.
Flask fornisce le basi e le linee guida, ma non impone alcuna dipendenza o *layout* obbligati di progetto. Sta allo sviluppatore scegliere gli strumenti che vuole usare e le librerie necessarie per realizzare il proprio *software*. Un notevole punto di forza è la presenza di moltissime estensioni fornite dalla comunità di sviluppatori che permettono l'aggiunta di funzionalità per soddisfare un ampio ventaglio di necessità sia progettuali sia implementative.
Una di queste estensioni, creata come fork del progetto "Flask-Classy", è Flask-Classful: un'estensione che aggiunge all'architettura di Flask "viste" basate su classi. Questa estensione permette quindi di raggruppare le viste in classi rilevanti, ognuna di essa con il proprio comportamento e contesto, molto più di quanto non si riesca a realizzare utilizzando l'approccio a BluePrints[14].
Per esempio, Flask-Classful genera automaticamente percorsi basati sui metodi nelle *views* di Flask, rendendo molto semplice l'*override* di quei percorsi utilizzando la sintassi familiare dei "decoratori", permettendo di raggruppare viste molto simili per la stessa risorsa in una singola classe.

2.5.3 REDIS

Redis è un DBMS NoSQL rilasciato per la prima volta nel 2009, di tipo *"key/value storage"*, basato su una struttura a dizionario: ogni valore immagazzinato è abbinato ad una chiave univoca che ne permette il recupero. In letteratura è spesso identificato come un "*server* di strutture dati", dal fatto che fornisce l'accesso a strutture dati mutabili attraverso un insieme di comandi, inviati utilizzando un modello *server-client* con socket TCP e un protocollo di comunicazione conciso. In questo modo diversi processi possono interrogare e modificare le stesse strutture di dati con approccio condiviso.
Un altro buon esempio è pensare a Redis come una versione più complessa di Memcached[15], dove le operazioni non sono solo SET e GET, ma operazioni che lavorano con tipi di dati complessi come liste, set, strutture di dati ordinati e così via. Redis e le sue strutture di dati astratte, quali ad esempio stringhe, *hash*, liste, insiemi, insiemi ordinati, *bitmap*, indici geospaziali etc., possiedono alcune proprietà:

- grande varietà di tipi di dato: nonostante l'architettura di Redis sia basata su una struttura a dizionario, i valori possono assumere varie forme (stringhe, insiemi, liste e molto altro);
- vengono memorizzate su disco, anche se sono sempre servite e modificate nella memoria del *server*. Questo significa che conservando i dati in memoria RAM, salvandoli in maniera persistente solo in un secondo momento, Redis permette di ottenere ottime prestazioni in scrittura e lettura;
- l'implementazione predilige sempre l'efficienza nell'utilizzo della memoria, quindi le strutture dati all'interno di Redis probabilmente useranno meno memoria rispetto alla

[13] https://github.com/mitsuhiko

[14] I *blueprint* possono semplificare enormemente il funzionamento di applicazioni di grandi dimensioni e fornire un mezzo centrale per le estensioni di Flask per registrare operazioni sulle applicazioni. Un oggetto Blueprint funziona in modo simile ad un oggetto applicazione di Flask, anche se non è effettivamente un'applicazione.

[15] Memcached è un sistema cache in RAM a oggetti distribuiti sviluppato da Danga Interactive originariamente per migliorare la velocità di LiveJournal.

stessa struttura dati modellata utilizzando un linguaggio di programmazione di alto livello;
- le operazioni sulle strutture sono di tipo atomico. Vuol dire che in caso di accessi concorrenti da parte di più *client*, i dati forniti risulteranno sempre aggiornati;
- possibilità di implementare configurazioni multi-nodo, *cluster* e replicazione.

Redis in REfAsE viene utilizzato come gestore della "coda" che implementa la struttura del protocollo di *Long Running Operation with Polling*, ovvero quella struttura dati dove vengono sistemati i dati ed i metadati dei *job* che richiedono un tempo di elaborazione lungo, dopo che una qualsiasi richiesta di un *client* ne ha causato la creazione. I *job* vengono movimentati sulla coda con una politica di tipo FIFO e gestiti dalla relativa logica applicativa che è implementata utilizzando la libreria python Redis Queue (RQ).

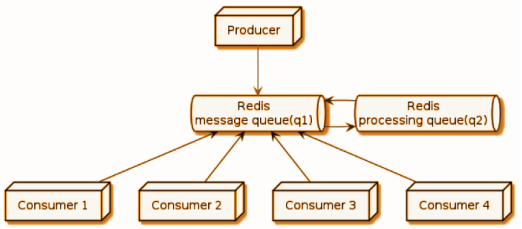


Figura 5 Schema concettuale del flusso di produzione, messa in coda e consumo delle informazioni utilizzando REDIS per il message queuing.
Figure 5 *Conceptual diagram representing the information flow related to production, queuing and consumption using REDIS as message queuing system.*

2.5.4 REDIS queue

Redis Queue (RQ) è una semplice libreria Python sviluppata da Vincent Driessen[16] per mettere in coda i *job* ed elaborarli in *background* utilizzando moduli *software* chiamati *worker*. È supportata da Redis ed è progettata per essere facilmente integrata in un progetto che ne utilizza la struttura dati astratta *queue* per implementare una qualsiasi applicazione basata sul modello dell'architettura asincrona.

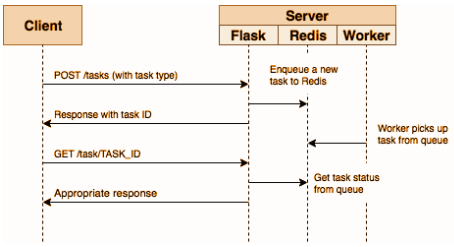


Figura 6 Sequence diagram dello scenario tipico di funzionamento in caso di accodamento di un task e successivo "consumo" da parte di un *worker*.
Figure 6 *Sequence diagram of typical scenario about queuing task on Redis and following consumption by a worker.*

[16] Profilo su https://nvie.com/about/ - Repository RQ su https://github.com/rq/rq

Le code di *job* sono un ottimo modo per permettere alle singole risorse di soddisfare le richieste dei *client* in modo asincrono al di fuori del flusso principale dell'applicazione. In dipendenza delle esigenze del progetto e dell'architettura da implementare, così come in modo congruente alle disponibilità dell'infrastruttura di calcolo, la politica che realizza le specifiche può essere vista come una combinazione lineare delle n code e degli m *worker* implementabili. In questo senso, è lecito ipotizzare una qualsiasi forma dell'architettura che si può esprimere nell'avere un qualsiasi numero di code e *worker*.

3. Protocollo e punti di accesso

In questo Capitolo vengono presentate le specifiche generali del protocollo di comunicazione. Verrà sempre omessa la parte di host e protocollo (http://server.ov.ingv.it/) perché non necessario alla trattazione. Nella descrizione vengono inoltre identificate alcune tipologie di risorse ed operazioni, tutte ben definite e determinanti per la creazione del modello:

- risorsa principale;
- risorse generiche;
- risorse di tipo: *worker*, *list*, *status* e *result*;
- operazioni: GET, POST, DELETE, PUT, HEAD.

Un aspetto importante, contemplato nell'implementazione, è che l'accodamento delle richieste, la loro elaborazione e la consegna dei risultati corrispondenti sono attività che possono essere assegnate a moduli *software* separati, in modo tale che il *polling* di un gran numero di *client* possa essere gestito con il giusto compromesso tra tempo di evasione della richiesta e sforzo computazionale.
In Figura 7, dove è presentato il modello logico dell'architettura di REfAsE, si possono vedere chiaramente i blocchi logici più importanti di tutto il sistema. Partendo dalla sinistra dell'immagine, la prima cosa che viene presentata è l'insieme deli *endpoint* esposti dal *set* di API: ognuno di essi rappresenta una risorsa di REfAsE che può essere consumata da un *client* che osserva il protocollo di comunicazione RESTful implementato.

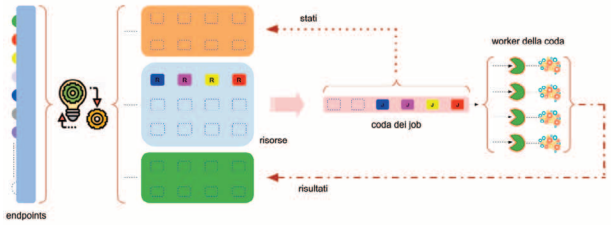


Figura 7 Modello architetturale logico di REfAsE.
Figure 7 *Architectural view model of REfAsE.*

Subito a destra, dopo l'astrazione della logica di *business*, si trova al centro (di colore azzurro) l'entità che comprende tutte le risorse di calcolo disponibili, rappresentate dai quadrati colorati etichettati con la lettera R. Ognuna di esse, se viene scelta per essere "consumata" dal *client*, viene avviata verso la coda logica e diventa un *job*. Lo stato di un *job*, via via che si aggiorna, viene comunicato e conservato dall'altra entità rappresentata con il colore arancio che lo rende disponibile al *client* che ha materialmente inviato in esecuzione la risorsa.

Un *job* accodato viene poi prelevato dalla coda (secondo una politica di tipo FIFO[17]) da un processo *worker* che permette così la materiale esecuzione del codice contenuto nel *job* specifico (algoritmi, calcoli, utilizzo di risorse di sistema, etc.). Quando un *job* ha completato la propria elaborazione, il risultato prodotto viene inviato e conservato dall'entità rappresentata in colore verde, rendendolo disponibile al *client* che ha effettuato la richiesta iniziale.

3.1 Struttura del protocollo

Il protocollo scelto nell'implementazione del servizio REfAsE è quindi centrato su "risorse" che possono rappresentare le operazioni di base effettuabili, unità computazionali disponibili o stati di esecuzione e di elaborazione. Come descrittore di parametri di *input* e di *output* (ove applicabile) si è deciso di utilizzare JSON come formato di dati, visto che esso è correttamente interpretato da diversi linguaggi di programmazione di alto livello ed è possibile creare e gestire entità di dati in questo formato in maniera assai semplice.

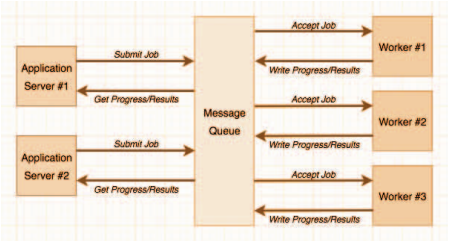


Figura 8 Vista funzionale globale del protocollo applicativo asincrono di REfAsE.
Figure 8 *Global functional view of asynchronous application protocol of REfAsE.*

3.2 Risorse

Ogni risorsa, di qualsiasi tipo essa sia, ha una forma di base composta da pochi parametri rappresentati come campi di una struttura JSON:

- *name*, il nome breve della risorsa, ad esempio *bullettins*, *locators*, etc;
- *location*, il path assoluto della risorsa, ad esempio /bulletins o /locators;
- *description*, la descrizione *human friendly* facilmente intellegibile;
- *type*, il tipo della risorsa, una stringa del tipo *"worker"* o *"list"*;
- una lista di parametri di input, descritti per tipo e nome, dove necessari.

L'operazione di base è la GET, mentre le POST, PUT ed HEAD possono essere supportate ed implementate o meno a seconda del tipo di risorsa.

3.2.1 Risorsa principale

L'*entry point* principale dell'intero sistema è definito con il path /REfAsE. Tutte le interazioni con il mondo esterno e le conversazioni con i *client* interessati ad utilizzare le funzionalità di REfAsE (le risorse rese disponibili con il set di API RESTful), devono iniziare con una operazione GET su questa risorsa.
Il risultato atteso corrisponde all'elenco delle risorse disponibili ed altre informazioni sul servizio; una tipica sessione di interazione di un *client* con REfAsE passa attraverso varie fasi:

[17] Il termine FIFO è l'acronimo inglese di First In First Out che rappresenta il metodo di transito in una coda: "primo ad entrare, primo ad uscire".

1. richiesta alla risorsa principale di tutte le risorse disponibili;
2. attraversamento dell'albero di risorse, con discesa nelle sottorisorse in cerca di quella più adatta;
3. sottomissione della richiesta tramite, in genere, un comando POST;
4. *polling* dello stato del *job* in caso di elaborazioni "lunghe";
5. restituzione e scarico dei risultati.

3.2.2 Risorse *list*

Una risorsa "list" è essenzialmente un insieme di sotto-risorse di qualsiasi tipo. Deve essere utilizzata per modellare ed identificare risorse accomunate per affinità di obiettivi, organizzazione gerarchica o insieme di funzionalità. Ad esempio, una possibile risorsa di tipo *list* potrebbe essere quella per le localizzazioni sismiche (es. /locators), tramite la quale si potrebbe accedere ad un insieme di sotto-risorse, come: /locator/hypo71, /locator/nonlinloc etc., dove ognuna di esse potrebbe fornire, ad esempio, accesso a diversi algoritmi di localizzazione.
Non c'è un limite al livello di annidamento di tali risorse: anche una sotto-risorsa può essere di tipo *list* e contenere a sua volta altre risorse di tipo *list*, e così via. In questo modo, ad esempio, un *client* in cerca di un localizzatore, richiederà la risorsa /locators ed otterrà come risposta una lista di "algoritmi" di localizzazione, per poi sottomettere la richiesta a quello che riterrà più opportuno.

Operazioni disponibili:

- GET, restituisce l'elenco delle sottorisorse.

Operazioni non supportate: PUT, HEAD, DELETE.

3.2.3 Risorse *worker*

Questo tipo rappresenta una risorsa verticale che realizza un compito ben definito da un punto di vista computazionale: esposta come *endpoint*, deve essere capace di mandare in esecuzione un *job* e portarlo a termine, sia nel caso in cui il *job* debba essere accodato per l'esecuzione asincrona, sia nel caso in cui possa essere eseguito al volo.
Il risultato di una sottomissione di una richiesta ad un worker può restituire tre risultati diversi, a seconda del tipo di worker (non si riportano gli eventuali errori):

1. se il *worker* implementa un'operazione di lunga durata utilizzando il pattern LROP, esso restituirà il codice HTTP 202 (Accepted) e, nell'*header* HTTP della risposta REST, il parametro Location con l'indirizzo di una risorsa di tipo *status* che il *client* dovrà interrogare per conoscere lo stato dell'elaborazione. Come corpo della risposta un JSON contenente lo stato iniziale del *job*, con lo stesso formato delle risposte delle risorse di tipo *status*;
2. se il *worker* implementa un'operazione breve ma con risultati multipli, esso restituirà il codice HTTP 201 (Created) e, nell'*header* HTTP della risposta REST, il parametro Location con l'indirizzo di una risorsa di tipo *result*. Il corpo della risposta conterrà un JSON con lo stesso contenuto di un *job* completato;
3. se il *worker* implementa un'operazione breve con un unico risultato (ad esempio la creazione di un'immagine), esso restituirà il codice HTTP 200 (Ok) ed il risultato atteso nel corpo della risposta.

Alcuni esempi di *endpoint* attualmente implementati sono:

/bulletins
/chart/frequency
/map
/notice/event/announce

Operazioni disponibili:

- GET, restituisce la descrizione della risorsa, come nell'elenco di /RefAsE;
- POST, avvia l'esecuzione del *job* o genera i risultati on the fly.

Operazioni non supportate: PUT, HEAD, DELETE.

3.2.4 Risorse *status*

Una risorsa di tipo *status* indica lo stato di un *job* precedentemente sottomesso al sistema da un *client* che ha utilizzato una risorsa di tipo *worker.* Esse vengono restituite quando si sottomette una richiesta ad un *worker* che impieghi un tempo "lungo" per produrre il risultato e devono necessariamente poter accettare le operazioni di GET e DELETE: con la prima operazione si interroga lo stato del *job* nella coda, mentre la seconda per cancellare il *job* dalla stessa.
Il formato della risorsa sarà generico: /queryjob/<jobid> dove jobid è l'identificativo univoco del *job* che interessa,sia che sia ancora in esecuzione, in caso di approccio asincrono, sia che sia già terminato.

Operazioni disponibili:

- GET, restituisce lo stato attuale del *job*;
- DELETE, termina un *job* in esecuzione e cancella ogni risorsa, anche file eventualmente già creati. Per i *job* già finiti, deve cancellare anche l'eventuale risorsa risultato e tutto ciò che sia stato generato dal *job* stesso. Successive GET ad un *job* cancellato dovranno restituire un codice HTTP 404 Not Found.

Operazioni non supportate: POST, PUT, HEAD.

3.2.5 Risorse *result*

Una risorsa "risultato" è quella restituita dopo che il *job* è terminato. Essenzialmente dovrà contenere l'elenco delle risorse create ed uno stato, ed il formato deve esplicitare il tipo di risultato esposto, ad esempio:
/bullettin/<bullid>
/location/<locator>/<eventid>

Operazioni disponibili:

- GET, restituisce la descrizione dei risultati;
- DELETE, cancella la risorsa e tutti i risultati creati, rimuove anche le informazioni del *job* che l'hanno generata. Una GET ad un risultato cancellato restituirà un 404 Not Found.

Operazioni non supportate: POST, PUT, HEAD.

Degno di nota è il fatto che se il risultato è rappresentato da un file statico, esso può essere restituito al *client* anche da un semplice web server, in esecuzione su una qualsiasi macchina dell'infrastruttura.

3.3 Worker correntemente implementati

3.3.1 Generatore dei bollettini sismici periodici

Il generatore di bollettini crea un insieme di immagini, un file HTML (*HyperText Markup Language*), documenti ODT (*Open DocumenT*) e DOCX (*MS Word™ Document*) ed alcuni documenti CSV (*Comma Separated Values*) in cui sono contenuti i dati riassuntivi per l'area richiesta o per tutte le possibili aree.
In merito agli attori coinvolti nel processo di creazione dei bollettini sismici, il seguente *sequence diagram* può essere interpretato anche come diagramma di flusso interfunzionale che modella tale richiesta.

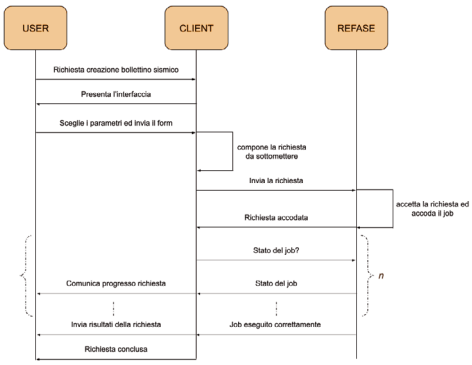


Figura 9 Sequence diagram dello scenario relativo alla creazione del bollettino sismico.
Figure 9 *Sequence diagram of the main scenario related to seismic bulletin creation.*

L'operazione di generazione di un bollettino inizia inviando la richiesta POST da parte del *client* dopo che l'utente ha scelto e selezionato i parametri obbligatori della richiesta o anche quelli aggiuntivi. Tali parametri comprendono l'area di interesse (Vesuvio, Ischia, Campi flegrei, Regione Campania), l'intervallo temporale di riferimento prestabilito (ultimi cinque anni, ultimo anno, ultimo semestre, mese, settimana) oppure un intervallo personalizzato liberamente selezionabile come "data inizio" e "data fine". È possibile anche scegliere il livello di revisione degli eventi da utilizzare nei risultati (definitivi, rivisti) ed anche selezionare uno specifico *binning*.
Nella Figura 10 è mostrata l'interfaccia presentata all'utente umano dal *client* predefinito implementato in WESSEL che, al momento in cui si scrive, è l'unico *client* esistente capace di dialogare con RefAsE.

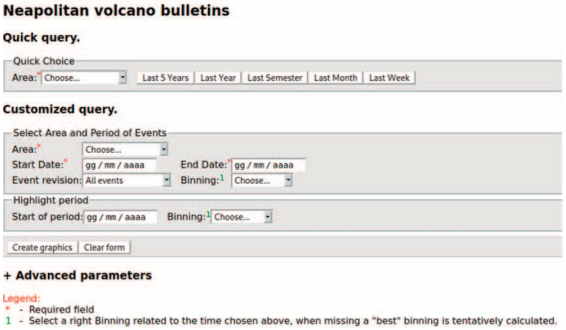


Figura 10 Interfaccia uomo-macchina del *client* implementato in WESSEL per la creazione del bollettino sismico.
Figure 10 *Human-computer interface of WESSEL client related to seismic bulletin creation.*

In seguito, poiché questo *worker* utilizza il pattern LROP sarà cura del *client* effettuare il *polling* alla risorsa *status* restituita nell'attesa che vengano completate le operazioni di creazione dei documenti. Nel seguito si riportano le schermate del *client* integrato nel sistema WESSEL per mostrare cosa un operatore vede dell'intera transazione.

Figura 11 Interfaccia presentata all'utente dal sistema WESSEL per la conferma dei dati della richiesta per la creazione del bollettino sismico.
Figure 11 *WESSEL human interface for confirm of the bulletin data creation.*

Utilizzando il protocollo di *polling* LROP, descritto nel Paragrafo 2.4, il *client* provvede ad informarsi sullo stato della richiesta inviata a REfAsE utilizzando la risorsa di tipo *status* esposta con URI /queryjob/<jobid> (con jobid identificativo del *job* di interesse).
Quando l'elaborazione è terminata, ovvero quando i numerosi singoli passi che permettono la creazione dei grafici e dei diagrammi necessari per la produzione del bollettino sismico sono stati completati, la risorsa queryjob/<jobid> restituisce lo stato "terminato". A questo punto è compito del *client* scaricare i risultati e presentarli all'utente mediante una apposita interfaccia. Essi vengono recuperati consumando la risorsa di tipo *result* all'URI /bullettin/<bullid> (con bullid identificativo univoco della richiesta di creazione del bollettino sismico).

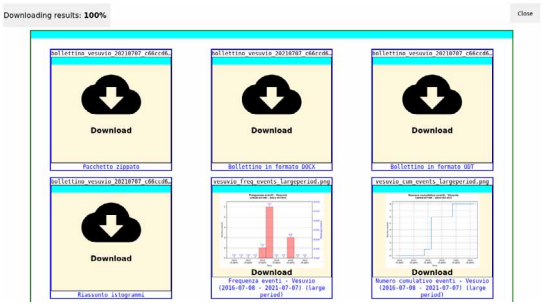


Figura 12 Come vengono presentati da WESSEL i singoli risultati relativi alla richiesta di bollettino sismico: ogni file è visionabile e scaricabile anche singolarmente (la schermata rappresenta una visione parziale dei risultati).
Figure 12 *Results presentation of the WESSEL client to the final user: each file can be viewed and downloaded (this screenshot represent a partial view).*

Da notare che anche prima che la richiesta venga completata appieno, il *client* WESSEL aggiorna l'utente con una serie di messaggi informativi che riportano sia lo stato e il progresso della richiesta complessiva, sia i risultati intermedi ottenuti.
I tempi medi di esecuzione rilevati per il completamento di una richiesta di questo tipo, ricadono in un intervallo compreso tra circa 20 secondi necessari per la creazione di un bollettino con un intervallo temporale di un mese (75 eventi sismici, 2 eventi non sismici) ai circa 40 secondi necessari per un intervallo di cinque anni (2060 eventi sismici, 43 eventi non sismici).

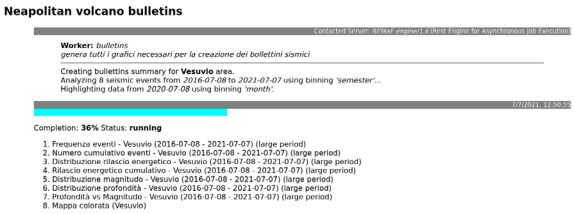


Figura 13 Il progresso dell'elaborazione mostrato dal *client* WESSEL all'utente.
Figure 13 *The progress of the request shown to the final user by the WESSEL client.*

3.3.2 Comunicati di evento sismico

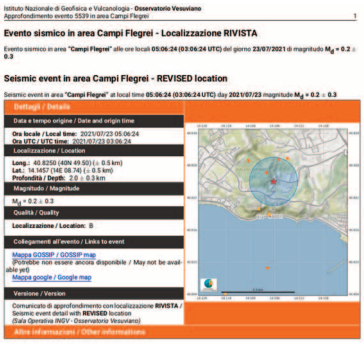
Nell'ambito dell'Accordo Quadro tra Dipartimento della Protezione Civile e l'Istituto Nazionale di Geofisica e Vulcanologia per le attività di sorveglianza Sismica e Vulcanica sul territorio Nazionale, di consulenza Tecnico – Scientifica e di studi sui rischi sismico e vulcanico, un "comunicato" identifica un particolare documento, creato al verificarsi di alcune condizioni *trigger* che contiene tutte le informazioni minime necessarie per informare il DPC qualitativamente e quantitativamente in merito alle attività dei vulcani campani, oggetto di sorveglianza da parte dell'Osservatorio Vesuviano.
La creazione di un comunicato è, quindi, subordinata al verificarsi di eventi sismici di particolare intensità e/o frequenza, le cui grandezze sono codificate e condivise proprio nell'ambito dell'accordo quadro citato in premessa.
Il punto di partenza per la creazione di tutto quanto necessario a formare il comunicato, è rappresentato dal punto di accesso raggiungibile con l'URI /notice. Il "consumo" della risorsa, ovvero l'operazione GET a /notice, restituisce altre due risorse, con relative sotto risorse:

- la risorsa event/, di tipo *list*, che implementa la gestione dei comunicati relativi ad un singolo evento, strutturata in tre sotto risorse di tipo *worker:* announce/, detailed/ e update/ che realizzano, rispettivamente, il comunicato notizia (da emettere entro 5 minuti dall'evento), quello di dettaglio (da emettere entro 30 minuti dall'evento) e quello di aggiornamento da emettere solo nel caso in cui occorra correggere alcune delle informazioni del comunicato di dettaglio;
- la risorsa swarm/, anch'essa di tipo *list* ma non implementata in questa versione, che è relativa alla gestione dei comunicati relativi agli sciami di eventi, strutturata in tre sotto risorse di tipo *worker* announce_swarm/, update_swarm/ e end_swarm/ che realizzano, rispettivamente, il comunicato notizia, quello di dettaglio e quello di aggiornamento sciame.

Per il comunicato notizia, la cui risorsa è esposta con URI /notice/event/announce, l'*input* atteso è rappresentato da un insieme di parametri obbligatori, tra cui l'area in cui ricade l'evento, le stazioni che lo hanno rilevato e quelle che appartengono alla rete dell'area, e un insieme di parametri opzionali, tra cui la magnitudo e un link pubblico a cui accedere ad una pagina di dettaglio dell'evento (WESSEL ancora non supporta questa funzionalità).
Il risultato atteso dall'elaborazione di questa risorsa è composto da 5 oggetti distinti che vengono restituiti al *client* che effettua la richiesta in modalità *on the fly* (senza accodamento):

1. una mappa dell'evento come immagine, non disponibile se la generazione del comunicato notizia è stata richiesta senza un evento nel *database*;
2. un comunicato in formato testo semplice;
3. un comunicato in HTML;
4. un comunicato in HTML da mail;
5. un comunicato in PDF.

Per il comunicato di dettaglio o di aggiornamento, le risorse di tipo *worker* (esposte con URI /notice/event/detailed e /notice/event/update) sono molto simili e devono produrre (a meno di frasi nell'*output*) il medesimo risultato che si otterrebbe dal comunicato notizia in cui sia stato passato un evento. L'*input* è praticamente identico a quello del comunicato notizia, così come il risultato atteso.
In Figura 14 è presentata la prima pagina di un comunicato di test con localizzazione "rivista" che rappresenta il modello di quanto realmente viene inviato al DPC nel caso in cui si verifichi un evento sismico "sopra soglia".
In questo caso, i tempi di esecuzione della richiesta di creazione di un comunicato sono pressoché istantanei.


Istituto Nazionale di Geofisica e Vulcanologia - **Osservatorio Vesuviano**
Approfondimento evento 5539 in area Campi Flegrei 1

Evento sismico in area Campi Flegrei - Localizzazione RIVISTA

Evento sismico in area **"Campi Flegrei"** alle ore locali **05:06:24 (03:06:24 UTC)** del giorno **23/07/2021** di magnitudo M_d = 0.2 ± 0.3

Seismic event in area Campi Flegrei - REVISED location

Seismic event in area **"Campi Flegrei"** at local time **05:06:24 (03:06:24 UTC)** day **2021/07/23** magnitude M_d = 0.2 ± 0.3

Dettagli / Details

Data e tempo origine / Date and origin time

Ora locale / Local time: 2021/07/23 05:06:24
Ora UTC / UTC time: 2021/07/23 03:06:24

Localizzazione / Location

Long.: 40.8250 (40N 49.50) (± 0.5 km)
Lat.: 14.1457 (14E 08.74) (± 0.5 km)
Profondità / Depth: 2.0 ± 0.3 km

Magnitudo / Magnitude

M_d = 0.2 ± 0.3

Qualità / Quality

Localizzazione / Location: B

Collegamenti all'evento / Links to event

Mappa GOSSIP / GOSSIP map
(Potrebbe non essere ancora disponibile / May not be available yet)
Mappa google / Google map

Versione / Version

Comunicato di approfondimento con localizzazione **RIVISTA** / Seismic event detail with **REVISED** location
(Sala Operativa INGV - Osservatorio Vesuviano)

Altre informazioni / Other informations

Figura 14 Esempio di comunicato creato da REfAsE ed inviato al Dipartimento di Protezione Civile.
***Figure 14** Example of communicate created by REfAsE and sent to the Civil Protection Department.*

3.3.3 Mappe e grafici

Oltre ai *worker* dedicati alla generazione dei bollettini sismici periodici (Paragrafo 3.3.1) e alla creazione dei comunicati di evento sismico (Paragrafo 3.3.2), sono stati implementati quelli necessari al servizio di creazione delle relazioni automatiche di dettaglio destinate ai Reperibili Sismologi.
Tale sistema utilizza alcune risorse verticali esposte da REfAsE che gli permettono di ottenere, nello specifico:

- la mappa con la localizzazione epicentrale dell'evento sismico, generata utilizzando il *worker* singev/ esposto come *endpoint* dalla lista map/ all'URI /map/singev/ dove vengono evidenziate anche le stazioni sismiche che hanno contribuito alla localizzazione;
- la mappa con la sismicità degli ultimi sette giorni, relativamente all'area dell'evento sismico, generata utilizzando il *worker* multev/ esposto come *endpoint* dalla lista map/ all'URI /map/multev/;
- l'istogramma con la frequenza del numero di eventi degli ultimi sette giorni, relativamente all'area dell'evento sismico, generato utilizzando il *worker* frequency/ esposto come *endpoint* dalla lista chart all'URI /chart/frequency/.

Come per gli altri *worker* descritti nelle sezioni precedenti, ognuno di essi richiede un insieme di parametri di *input* che vengono comunicati al richiedente (ovvero al consumatore della risorsa) alla prima operazione GET compiuta sui rispettivi URI.

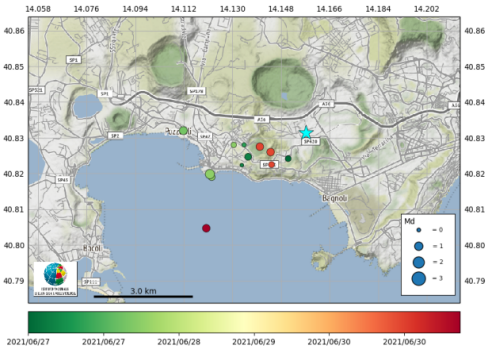


Figura 15 Esempio di mappa della sismicità degli ultimi sette giorni (dalla data evento, a ritroso).
Figure 15 *Exemple of last seven days seismic map (from event date, backward).*

4. Integrazione, test e messa in servizio

In REfAsE attualmente sono implementati e gestiti due tipi di *endpoint*:

- puramente asincroni, per la creazione dei grafici necessari alla redazione dei bollettini sismici;
- sincroni (off queue), per la creazione e l'invio dei file necessari ai comunicati di eventi sismici sopra-soglia;
- per la creazione delle mappe e istogrammi relativi ad un singolo evento sismico o ad un set di eventi multipli (*endpoint* utilizzati, ad esempio, per la creazione della relazione automatica di dettaglio necessaria ai reperibili sismologi in caso di eventi con $M_d \geq 4.0$).

Dopo una valutazione sul carico computazionale atteso e sulle caratteristiche dell'infrastruttura di produzione, l'architettura asincrona è stata configurata per funzionare con una singola coda e 4 distinti *worker*. Degno di nota è il fatto che RQ supporta anche la gestione dei *job* falliti utilizzando una propria implementazione del *Failed Job Registry*.
Per integrare REfAsE nell'infrastruttura globale del software di Sala Operativa, è stato messo a punto un sistema di installazione custom realizzato utilizzando GNU Make[18]. La sequenza di operazioni da eseguire per costruire l'installazione del software parte da alcuni file di configurazione, i *makefile*, in cui sono elencati sia i parametri fondamentali necessari al funzionamento del sistema, sia le dipendenze e le regole che devono essere soddisfatte per la messa in esercizio in ambiente di produzione.
Viene inoltre definita anche la lista degli *endpoint* esposti dal sistema, ognuno dei quali corrisponde ad un modulo implementato nel *package resources*.
A solo titolo di esempio, tra i parametri di configurazione utilizzati dal sistema di installazione compaiono l'indirizzo del *server* Redis con la relativa porta di servizio, l'indirizzo e porta dove deve essere esposto l'*entry point* della risorsa principale di tutto il sistema e l'identificativo con il quale REfAsE risponde alle richieste. Vengono anche impostati il nome della coda di riferimento ed istanziata la connessione a Redis via RQ.
Per testare l'aderenza delle funzionalità implementate con i requisiti raccolti e con le specifiche delle API progettate, è stato utilizzato un approccio di *test by unit* verticale a riga di comando, senza quindi l'ausilio di alcun tool esterno per l'automazione dei piani di test. Tale approccio ha previsto la preparazione accurata di casi di test e scenari di *fault* mediante l'esecuzione di chiamate REST via CURL (*Command line tool and library for transferring data with URLs*) in ambiente Linux, verificando tutte le operazioni ammesse e non ammesse dagli *endpoint* ed analizzando la sintassi ed il contenuto delle risposte ottenute.

[18] https://www.gnu.org/software/make/

Per completezza, si riportano le date in cui il sistema è entrato in produzione, con i relativi servizi esposti:

- Generatore di bollettini sismici: novembre 2020
- Comunicati di eventi sismici: marzo 2021
- Mappe e grafici per la Relazione Automatica di evento per Reperibili: marzo 2021

5. Conclusioni e sviluppi futuri

Nello sviluppo del sistema si è raggiunto un buon compromesso tra le richieste che vengono elaborate e quelle che vengono accodate per una elaborazione successiva, garantendo allo stesso tempo il fatto che i *client* non restino in attesa del risultato. Grazie all'approccio di servire più richieste contemporaneamente anche in modo asincrono, i sistemi *client* di REfAsE possono sottomettere delle richieste di calcolo specifiche, fisse e dichiarate dal *server*, in modo da non dover implementare in prima persona le risorse computazionali richieste. La struttura asincrona del sistema fornisce inoltre un buon mezzo per evitare inutili attese da parte delle interfacce utente (al momento implementate solamente in WESSEL) durante le operazioni di lunga durata. Il sistema permette la centralizzazione e la riutilizzazione di codice e soluzioni offrendo al contempo un unico punto di accesso con un protocollo agile che si è dimostrato facilmente estendibile. La struttura modulare con cui è stato scritto il codice ne ha permesso facilmente l'estensione quando, ad esempio, è stato necessario implementare le risorse necessarie a generare le relazioni automatiche, riutilizzando il codice precedentemente scritto per la creazione dei bollettini periodici.
Tra gli sviluppi futuri è previsto che REfAsE venga utilizzato come interfaccia verso vari software di localizzazione (HYPO71, HYPO2000, HYPOELLIPSE, NonLinLoc, ecc.) permettendo in tal modo di avere dei punti di accesso per effettuare le localizzazioni sismiche tramite "microservizi" e dando così la possibilità al portale WESSEL di poter integrare anche la parte di localizzazione. In un secondo momento anche il servizio di generazione delle relazioni automatiche, attualmente fornito da un servizio separato, verrà integrato nel sistema RefAsE.
È prevista, inoltre, in futuro la ristrutturazione del set di API in aderenza con lo standard OpenAPI[19].
Tutto il codice sorgente è liberamente disponibile accedendo (previa registrazione) al repository GIT INGV all'indirizzo: https://gitlab.rm.ingv.it/alessandro.difilippo/rest-job-dispatcher.git

Bibliografia

AA. VV., (2020). *Progetto "Sale Operative Integrate e Reti di monitoraggio del futuro: l'INGV 2.0". Report finale*. Editors: L. Margheriti, F. Cirillo, F. Guglielmino, M. Moretti. Misc. INGV, 57: 1-186, https://doi.org/10.13127/misc/57

Accordo quadro DPC – INGV: https://istituto.ingv.it/index.php/it/2-non-categorizzato/199-accordo-quadro-2012-2021.

Berners-Lee T., Fielding R., Masinter L., (2005). *Uniform Resource Identifier (URI): Generic Syntax*. RFC 3986:1–61.

Fielding R., (2000). *Architectural Styles and the Design of Network-based Software Architectures*, PhD Thesis, University of California, Irvine.

Fielding R., Gettys J., Mogul J., Frystyk H., Masinter L., Leach P., Berners-Lee T., (1999). *Hypertext*

[19] La specifica OpenAPI (conosciuta originariamente come la specifica Swagger) è una specifica per file di interfaccia leggibili dalle macchine per descrivere, produrre, consumare e visualizzare servizi web RESTful.

Transfer Protocol - HTTP/1.1. RFC 2616:1–176.

Gilbert S., Lynch N., (2002). *Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services*. SIGACT News 33, 2 51-59. https://doi.org/10.1145/564585.564601

Gregorio J., Fielding R., Hadley M., Nottingham M., Orchard D., (2012). *URI Template*. RFC 6570:1–34.

Ivanchikj A., (2021). *RESTalk: A Visual and Textual DSL for Modelling RESTful Conversations*, PhD Thesis, USI, Lugano.

Pautasso C., (2016). *RESTful Conversation Patterns*, Talk at REST Fest, Edinburgh.

Peluso R., Benincasa A., Cirillo F., Di Filippo A., Scarpato G., (2020). *Lo sviluppo dei nuovi sistemi integrati di Sala: il sistema WESSEL ed i suoi simbionti*. In: Progetto "Sale Operative Integrate e Reti di monitoraggio del futuro: l'INGV 2.0". Report finale. Editors: L. Margheriti, F. Cirillo, F. Guglielmino, M. Moretti. Misc. INGV, 57: 1-186, https://doi.org/10.13127/misc/57/5

Peluso R., (2020). *Il database sismico SERENADE: un sistema REST per la gestione delle localizzazioni sismiche*. In: Progetto "Sale Operative Integrate e Reti di monitoraggio del futuro: l'INGV 2.0". Report finale. Editors: L. Margheriti, F. Cirillo, F. Guglielmino, M. Moretti. Misc. INGV, 57: 1-186, https://doi.org/10.13127/misc/57/7

Thomas A., (2010). *BPMN 2.0: Introduction to the Standard for Business Process Modeling*. BoD–Books on Demand, ISBN 3839149851.

QUADERNI di GEOFISICA

ISSN 1590-2595 http://istituto.ingv.it/it/le-collane-editoriali-ingv/quaderni-di-geofisica.html/

I QUADERNI DI GEOFISICA (QUAD. GEOFIS.) accolgono lavori, sia in italiano che in inglese, che diano particolare risalto alla pubblicazione di dati, misure, osservazioni e loro elaborazioni anche preliminari che necessitano di rapida diffusione nella comunità scientifica nazionale ed internazionale. Per questo scopo la pubblicazione on-line è particolarmente utile e fornisce accesso immediato a tutti i possibili utenti. Un Editorial Board multidisciplinare ed un accurato processo di peer-review garantiscono i requisiti di qualità per la pubblicazione dei contributi. I QUADERNI DI GEOFISICA sono presenti in "Emerging Sources Citation Index" di Clarivate Analytics, e in "Open Access Journals" di Scopus.

QUADERNI DI GEOFISICA (QUAD. GEOFIS.) welcome contributions, in Italian and/or in English, with special emphasis on preliminary elaborations of data, measures, and observations that need rapid and widespread diffusion in the scientific community. The on-line publication is particularly useful for this purpose, and a multidisciplinary Editorial Board with an accurate peer-review process provides the quality standard for the publication of the manuscripts. QUADERNI DI GEOFISICA are present in "Emerging Sources Citation Index" of Clarivate Analytics, and in "Open Access Journals" of Scopus.

RAPPORTI TECNICI INGV

ISSN 2039-7941 http://istituto.ingv.it/it/le-collane-editoriali-ingv/rapporti-tecnici-ingv.html/

I RAPPORTI TECNICI INGV (RAPP. TEC. INGV) pubblicano contributi, sia in italiano che in inglese, di tipo tecnologico come manuali, software, applicazioni ed innovazioni di strumentazioni, tecniche di raccolta dati di rilevante interesse tecnico-scientifico. I RAPPORTI TECNICI INGV sono pubblicati esclusivamente on-line per garantire agli autori rapidità di diffusione e agli utenti accesso immediato ai dati pubblicati. Un Editorial Board multidisciplinare ed un accurato processo di peer-review garantiscono i requisiti di qualità per la pubblicazione dei contributi.

RAPPORTI TECNICI INGV (RAPP. TEC. INGV) publish technological contributions (in Italian and/or in English) such as manuals, software, applications and implementations of instruments, and techniques of data collection. RAPPORTI TECNICI INGV are published online to guarantee celerity of diffusion and a prompt access to published data. A multidisciplinary Editorial Board and an accurate peer-review process provide the quality standard for the publication of the contributions.

MISCELLANEA INGV

ISSN 2039-6651 http://istituto.ingv.it/it/le-collane-editoriali-ingv/miscellanea-ingv.html

MISCELLANEA INGV (MISC. INGV) favorisce la pubblicazione di contributi scientifici riguardanti le attività svolte dall'INGV. In particolare, MISCELLANEA INGV raccoglie reports di progetti scientifici, proceedings di convegni, manuali, monografie di rilevante interesse, raccolte di articoli, ecc. La pubblicazione è esclusivamente on-line, completamente gratuita e garantisce tempi rapidi e grande diffusione sul web. L'Editorial Board INGV, grazie al suo carattere multidisciplinare, assicura i requisiti di qualità per la pubblicazione dei contributi sottomessi.

MISCELLANEA INGV (MISC. INGV) favours the publication of scientific contributions regarding the main activities carried out at INGV. In particular, MISCELLANEA INGV gathers reports of scientific projects, proceedings of meetings, manuals, relevant monographs, collections of articles etc. The journal is published online to guarantee celerity of diffusion on the internet. A multidisciplinary Editorial Board and an accurate peer-review process provide the quality standard for the publication of the contributions.

Coordinamento editoriale
Francesca DI STEFANO
Istituto Nazionale di Geofisica e Vulcanologia

Progetto grafico
Barbara ANGIONI
Istituto Nazionale di Geofisica e Vulcanologia

Impaginazione
Barbara ANGIONI
Patrizia PANTANI
Massimiliano CASCONE
Istituto Nazionale di Geofisica e Vulcanologia

Istituto Nazionale di Geofisica e Vulcanologia
Via di Vigna Murata, 605
00143 Roma
tel. +39 06518601

www.ingv.it

ISTITUTO NAZIONALE DI GEOFISICA E VULCANOLOGIA