RdS
RICERCA DI SISTEMA

ENEA
AGENZIA NAZIONALE PER LE NUOVE TECNOLOGIE, L'ENERGIA E LO SVILUPPO ECONOMICO SOSTENIBILE

MINISTERO DELL'AMBIENTE E DELLA SICUREZZA ENERGETICA

Ricerca di Sistema elettrico

Sviluppo e ingegnerizzazione di un Client Open Source per la Smart City Platform (LA2.2)

Tito Brasolin

KERBEROS

RdS_PTR 22-24_PR1.7_LA2.2_354

Sviluppo e ingegnerizzazione di un Client Open Source per la Smart City Platform

SCP: Sviluppo evolutivo del Framework di Governance dei Dati Urbani (LA2.2)

T. Brasolin (Kerberos Srl, Padova)

Dicembre 2024

Report Ricerca di Sistema Elettrico

Accordo di Programma Ministero dell'Ambiente e della Sicurezza Energetica - ENEA Piano Triennale di Realizzazione 2022-2024

Obiettivo: Decarbonizzazione

Progetto: Tema di ricerca 1.7 – Tecnologie per la penetrazione efficiente del vettore elettrico negli usi finali

Linea di attività: 2.2

Responsabile del Progetto: Claudia Meloni, ENEA

Responsabile del Work Package: Claudia Meloni, ENEA

Responsabile Linea di Attività: Cristiano Novelli, ENEA

Mese inizio previsto: 19

Mese inizio effettivo: 19

Mese fine previsto: 36

Mese fine effettivo: 36

Il presente documento descrive le attività di ricerca svolte all'interno contratto dal titolo: "Sviluppo e ingegnerizzazione di un Client Open Source per la Smart City Platform"

Indice

Indice delle figure

1 Risultati attesi

Premesso che:

- Le Smart City Platform Specification[1] (SCPS) for Interoperability Layer sono specifiche pubbliche definite dai laboratori TERIN-ICER-CROSS e TERIN-ICER-SCC di ENEA per permettere una comunicazione tra sistemi in ambito Smart City. Scopo delle SCPS è permettere a una Piattaforma ICT di gestione della città (o del distretto) di adottare un approccio condiviso per comunicare con le diverse Piattaforme Verticali che insistono sullo stesso tessuto urbano e così fornire uno strumento alle municipalità, svincolandosi da soluzioni proprietarie chiuse;
- La "SmartCityPlatform" (SCP) è un software prototipale sviluppato da ENEA per dimostrare l'usabilità e l'applicabilità delle specifiche SCPS sviluppate dagli stessi laboratori di ENEA per favorire l'interoperabilità delle applicazioni verticali in ambito Smart City con una piattaforma centrale di monitoraggio su scala cittadina, denominata SCP;
- Al momento lo sviluppo di client compatibili con le SCPS in grado di comunicare con una SCP è lasciato a chi voglia utilizzare le specifiche;

si ritiene che la diffusione delle SCPS possa essere favorita dalla disponibilità di un client Open Source da affiancare agli altri strumenti già disponibili[2].

Si desidera quindi un client che, oltre a essere pienamente compatibile con le specifiche di cui sopra:

- sia multipiattaforma;
- sia facilmente installabile;
- sia configurabile mediante file di testo;
- produca un dettagliato registro di attività ed errori;

Il client dovrà implementare le chiamate ai metodi[3] **lastRequest**, **searchingRequest** e **push**.

Per quanto concerne il metodo **push**, il client dovrà potersi comportare come un processo in attesa di UrbanDataset (UD) da inviare alla SCP effettuando il **login** come da "best practice" B1[4], gestendo l'eventuale ripetizione degli invii sulla base di criteri configurabili.

Gli UD dovranno essere convalidati prima dell'invio, utilizzando le librerie JAVA per la validazione schematron fornite da ENEA.

Infine, si attende da parte del client l'invio di un report periodico via mail.

[1] https://smartcityplatform.enea.it/#/it/specification/index.html
[2] https://smartcityplatform.enea.it/#/it/tools/index.html
[3] https://smartcityplatform.enea.it/#/it/specification/communication/2.0/index.html
[4] https://smartcityplatform.enea.it/#/it/specification/communication/2.0/index.html#bestpracticewscalling

2 Risultati ottenuti

- Optando per **Python** come linguaggio di sviluppo è stato ottenuto un client multipiattaforma, facilmente installabile ed eseguibile.
- Oltre alla configurabilità mediante file di testo è stata implementata l'acquisizione di parametri da riga di comando.
- Oltre al client da riga di comando (che, come da capitolato, espone in modo fruibile per l'utente finale i principali metodi di interazione con la SCP) è stata prodotta per gli sviluppatori una libreria Python che implementa esaustivamente le specifiche SCPS Communication 2.0 e di cui il client stesso internamente fa uso.

Il progetto ha quindi coinvolto la creazione di un'applicazione CLI (Command Line Interface) in Python, basata su una libreria client generata con il tool openapi-generator-cli a partire da una specifica OpenAPI 3.0 che descrive le API web esistenti. L'applicazione CLI è stata sviluppata utilizzando la libreria Click, è stata organizzata in modo da poter essere distribuita con licenza Open Source e depositata in un repository pubblico.

Degno di nota il fatto che a partire dalle SCPS è stata elaborata una descrizione dei servizi web in formato OpenAPI 3.0 e da quest'ultima è stata generata la libreria di cui sopra.

Ciò apre alla possibilità di generare rapidamente in diversi linguaggi di programmazione sia altri client, sia il codice di base ("server stub") per creare server conformi alle SCPS, riducendo il tempo e lo sforzo necessari per avviare lo sviluppo di un'API.

3 Prodotti attesi

Il prodotto finale atteso consiste in un client Open Source multipiattaforma per la SCP sviluppato in linguaggio Python, adeguatamente strutturato e rilasciato con licenza Open Source.

4 Prodotti sviluppati

Come da capitolato, il **client da riga di comando** è stato sviluppato in linguaggio Python e reso accessibile con licenza Apache 2.0 tramite il repository pubblico:

https://crossserv.bologna.enea.it/gitlab/cross/scp_udg_client_python_app

L'utente può esplorare il codice sorgente direttamente da interfaccia web o scaricarne una copia mediante il comando:

git clone https://crossserv.bologna.enea.it/gitlab/cross/scp_udg_client_python_app.git

L'applicazione eseguibile richiede Python 3.8 o superiore e Git per l'installazione da repository:

pip install git+https://crossserv.bologna.enea.it/gitlab/cross/scp_udg_client_python_app.git

Se l'installazione va a buon fine, il comando seguente mostrerà alcune istruzioni per l'utilizzo:

python -m scp_udg_client_app.cli

Un distinto repository ospita la **libreria client REST**:

https://crossserv.bologna.enea.it/gitlab/cross/scp_udg_client_python_rest

L'interfaccia web consente di esplorare il codice sorgente del client REST e la documentazione con esempi di ciascuno dei metodi disponibili. La libreria può essere installata da repository, ma è stata anche pubblicata nel Python Package Index (PyPI) e può quindi essere installata con:

pip install scp-udg-client-rest

5 Analisi degli scostamenti su attività e risultati

Il lavoro è proceduto secondo le previsioni raggiungendo gli obiettivi stabiliti.

Come unico scostamento (concordato in corso d'opera) tra i risultati attesi e quelli ottenuti si segnala la rinuncia alla convalida schematron mediante libreria JAVA, nonostante i primi test di fattibilità basati sul "bridge" JPype[5] abbiano dato esiti promettenti.

Lo sforzo necessario per rendere affidabile tale funzionalità su tutte le piattaforme è apparso eccessivo, anche in considerazione del fatto che il codice Python generato da OpenAPI effettua già una convalida degli UrbanDataset sufficientemente stringente per gli scopi correnti.

[5] https://jpype.readthedocs.io/en/latest/

6 Sintesi delle attività svolte

L'analisi preliminare ha portato ad articolare lo sviluppo in due sottoprogetti: una libreria client WEB ed un'applicazione da riga di comando che utilizza tale libreria per interagire con la SCP.

6.1 Client WEB

1. Elaborazione sulla base delle specifiche SCPS Communication 2.0 di una descrizione esaustiva del web service Urban DatasetGateway, in formato OpenAPI 3.0[6]
2. Produzione del client web in linguaggio Python a partire dalla descrizione di cui sopra. Il codice con documentazione ed esempi è stato depositato nel repository:

   https://crossserv.bologna.enea.it/gitlab/cross/scp_udg_client_python_rest

6.2 Applicazione CLI[7]

1. Identificazione delle funzionalità principali necessarie (test, login, push etc.) e creazione di un piano di modularizzazione del codice.
2. Creazione di una struttura standard per il repository Python:

   https://crossserv.bologna.enea.it/gitlab/cross/scp_udg_client_python_app

3. Implementazione dei comandi con l'ausilio della libreria open source Click ("Command Line Interface Creation Kit") https://github.com/pallets/click/

[6] https://swagger.io/specification/v3/
[7] Interfaccia a riga di comando (*command line interface*)

7 Dettaglio delle attività svolte

OpenAPI è una specifica standardizzata per descrivere le API RESTful in modo formale e strutturato. Precedentemente nota come Swagger, OpenAPI permette agli sviluppatori di definire in modo chiaro e comprensibile le funzionalità delle API, i loro endpoint, i parametri di input e output, i tipi di dati e molto altro.

Utilizzando strumenti come **openapi-generator-cli** è possibile generare automaticamente client e server in vari linguaggi di programmazione risparmiando tempo e riducendo gli errori manuali nella scrittura del codice *boilerplate* ed è per questo che si è seguito tale approccio nello sviluppo del client web.

7.1 Client WEB

A partire dalla documentazione delle API web è stata creata manualmente la relativa descrizione in formato OpenAPI 3.0 e da questa è stato generato un client Python mediante openapi-generator-cli[8].

7.1.1 Analisi della Documentazione delle API

1. La documentazione delle API web esistenti è stata analizzata attentamente per comprendere la struttura e le funzionalità offerte.
2. Sono stati identificati i metodi HTTP utilizzati (GET, POST), i parametri richiesti e gli schemi di risposta.

7.1.2 Creazione della Specifica OpenAPI 3.0

1. La descrizione delle API è stata redatta utilizzando il formato OpenAPI 3.0, che include informazioni dettagliate sugli endpoint, i parametri, i tipi di dati e i codici di risposta.
2. È stato utilizzato YAML come linguaggio di markup per scrivere la specifica, garantendo che fosse ben strutturata e conforme allo standard OpenAPI.

7.1.3 Utilizzo di openapi-generator-cli

1. È stata impiegata l'utility openapi-generator-cli per generare automaticamente un client Python basato sulla specifica OpenAPI creata.
2. Questo ha permesso di ottenere un client Python preconfigurato per interagire con le API, riducendo significativamente il tempo e gli errori rispetto a una scrittura manuale del client.

Sono stati eseguiti dei test per verificare che il client Python generato funzionasse correttamente e fosse in grado di interagire con le API come previsto.

Al fine di esemplificare la possibilità di generazione di codice in vari linguaggi è stato redatto un tutorial sulla creazione di un client Bash:

https://crossserv.bologna.enea.it/gitlab/cross/scp_udg_client_python_app/-/wikis/home/openapi-cli-bash

[8] https://openapi-generator.tech/docs/installation

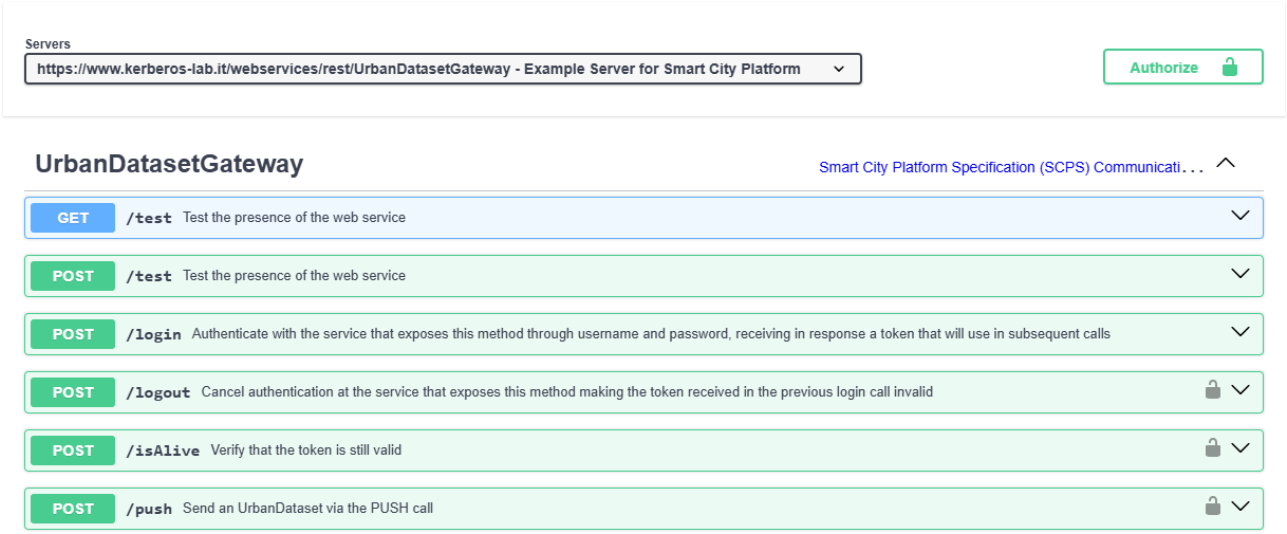

Figura 1 - GitLab supporta la visualizzazione dell'interfaccia Swagger per i descrittori OpenAPI direttamente all'interno dell'interfaccia web del progetto. Questo consente agli sviluppatori di esplorare e testare le API descritte senza lasciare GitLab, migliorando così la documentazione e la facilità di utilizzo delle API.

https://crossserv.bologna.enea.it/gitlab/cross/scp_udg_client_python_app/-/blob/main/schema/openapi3_0.yaml

7.2 Applicazione CLI

Per sviluppare l'applicazione CLI è stata utilizzata la libreria Click[9], che facilita la creazione di interfacce a riga di comando in Python. Click è stato scelto per le sue caratteristiche di semplicità, flessibilità e capacità di gestire facilmente comandi annidati e opzioni. I comandi implementati nell'applicazione includono “is_alive”, “last_request”, “login”, “push”, “searching_request” e” test”.

Altre librerie sono utilizzate per gestire la configurazione mediante file di proprietà JAVA, il logging e altre funzionalità.

Il codice è stato organizzato in modo modulare, in modo da separare la logica della CLI dall'interazione con la libreria client. Sono state create funzioni specifiche per ciascun comando CLI, che chiamano i metodi appropriati della libreria client.

Il codice è stato preparato per la distribuzione con licenza Open Source. È stata scelta e aggiunta al repository una licenza appropriata (Apache 2.0). Il repository è stato reso pubblico e include una documentazione dettagliata su come installare e utilizzare l'applicazione CLI.

[9] https://click.palletsprojects.com/en/stable/

7.2.1 Organizzazione del codice

```
scp_udg_client_python_app
├── LICENSE
├── pyproject.toml
├── README.md
├── requirements.txt
├── schema
│   └── openapi3_0.yaml
├── scp_udg_client_app
│   ├── cli.py
│   ├── commands
│   │   ├── generate_whatever.py
│   │   ├── __init__.py
│   │   ├── is_alive.py
│   │   ├── last_request.py
│   │   ├── login.py
│   │   ├── push.py
│   │   ├── report.py
│   │   ├── searching_request.py
│   │   └── test.py
│   ├── helpers.py
│   ├── __init__.py
│   ├── __main__.py
│   ├── options.py
│   └── services
│       ├── api.py
│       ├── auth.py
│       ├── __init__.py
│       └── token.py
└── setup.py
```

Figura 2 - Il file "setup.py" include tutte le informazioni necessarie per l'installazione e la distribuzione del modulo, ciascun comando è stato implementato separatamente all'interno della cartella "commands".

7.2.2 Principali librerie Python utilizzate

1. **click**: Permette di creare interfacce a riga di comando in modo semplice e componibile.
2. **click-config-file**: Un'estensione di Click che permette di caricare configurazioni da file.
3. **click-log**: Un'estensione di Click che integra facilmente il logging nelle applicazioni CLI.
4. **javaproperties**: Una libreria Python per leggere e scrivere file .properties di Java.
5. **outgoing**: Fornisce un'interfaccia comune per gestire diversi metodi di invio di e-mail.
6. **simple-file-poller**: Libreria Python per monitorare e rilevare nuovi file in una directory.

7.2.3 Comandi implementati

Click consente di creare comandi annidati che possono essere eseguiti tramite il terminale. L'utente finale invoca l'eseguibile principale "scp-udg-client" seguito dal comando di primo livello e può passare opzioni e argomenti.

Elenchiamo nel seguito i comandi di primo livello, fornendo una breve descrizione e un esempio di utilizzo minimale per ciascuno. Per maggiori dettagli, si invita a consultare la documentazione.

7.2.3.1 Comando "test"

1. **Descrizione**: Verifica la presenza online del web service.
2. **Utilizzo**: scp-udg-client test —udg-endpoint <udg-endpoint>

7.2.3.2 Comando "login"

1. **Descrizione**: Si autentica presso il servizio e memorizza il token di autenticazione.
2. **Utilizzo**: scp-udg-client login —udg-endpoint <udg-endpoint> —username <username> —password <password>

7.2.3.3 Comando "is_alive"

1. **Descrizione**: Verifica che il token sia ancora valido.
2. **Utilizzo**: scp-udg-client is_alive —udg-endpoint <udg-endpoint>

7.2.3.4 Comando "last_request"

3. **Descrizione**: Richiede e salva localmente l'ultimo UrbanDataset.
4. **Utilizzo**: scp-udg-client last_request —udg-endpoint <udg-endpoint> —resource-id <resource-id> —inbox-dir <inbox-dir>

7.2.3.5 Comando "push"

1. **Descrizione**: Invia al servizio gli UrbanDataset presenti in una cartella locale.
2. **Utilizzo**: scp-udg-client push —udg-endpoint <udg-endpoint> —outbox-dir <outbox-dir> —continuous
3. **Note**: In modalità "continuous" questo comando resta in esecuzione e monitora la cartella degli UrbanDataset da inviare, altrimenti si arresta immediatamente dopo l'eventuale invio.

7.2.3.6 Comando "searching_request"

1. **Descrizione**: Richiede uno o più UrbanDataset con raffinamento della ricerca a livello di context.
2. **Utilizzo**: scp-udg-client searching_request —udg-endpoint <udg-endpoint> —resource-id <resource-id> —inbox-dir <inbox-dir> —period-start <period-start> —period-end <period-end>

7.2.3.7 Comando "report"

3. **Descrizione**: Genera un rapporto sullo stato degli invii che può essere spedito per email.
1. **Utilizzo**: scp-udg-client report —email-to <email-to> —continuous
2. **Note**: In modalità "continuous" questo comando resta in esecuzione inviando periodicamente il report con frequenza programmabile, altrimenti esce immediatamente.

7.2.4 Configurabilità e logging

Ciascun comando legge da un file in formato JAVA .properties i parametri di esecuzione, i quali possono essere sovrascritti dai parametri forniti direttamente nella linea di comando. Il percorso di tale file può essere indicato tramite il parametro *—config* in assenza del quale si presume il default config.properties

Inoltre, è possibile abilitare la scrittura su file del registro degli eventi verificatisi durante l'esecuzione, con vari livelli di dettaglio, mediante appositi parametri dell'eseguibile principale.

Ad esempio, il comando seguente monitora continuamente la cartella OUTBOX registrando gli eventi su file con il massimo livello di dettaglio:

scp-udg-client —verbosity DEBUG —log-file push.log push —config push.properties —udg-endpoint https://www.kerberos-lab.it/webservices/rest/UrbanDatasetGateway —outbox-dir OUTBOX —continuous

8 Contributo delle eventuali consulenze alle attività sopra descritte

Il lavoro è stato svolto dalla Kerberos Srl di Padova senza l'ausilio di ulteriori consulenze.

9 Pubblicazioni scientifiche

Dall'attività svolta non sono risultate pubblicazioni.

10 Eventi di disseminazione

Non sono stati organizzati eventi specifici riferiti all'attività.