The Journal of Open Source Software

FiLiP: A python software development kit (SDK) for accelerating the development of services based on FIWARE IoT platform

Thomas Storek[1,2], **Junsong Du**[1¶], **Sebastian Blechmann**[1], **Rita Streblow**[1], **and Dirk Müller**[1]

1 Institute for Energy Efficient Buildings and Indoor Climate, RWTH Aachen University, Germany **2** Drees & Sommer SE, Germany ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 25 March 2024
Published: unpublished

Summary

FIWARE, an open-source initiative providing open-source software platform components and a set of standardized APIs (Application Programming Interfaces), has been instrumental in driving digitalization across various domains and research fields. The use of FIWARE enables faster, easier, and cheaper developments of a wide range of IoT-enabled solutions (*FIWARE Catalogue – FIWARE*, 2023). Over the years, the FIWARE Next-Generation-Service-Interface (NGSI-v2) API specification has been well-developed. There is also a variety of reusable software components, so-called Generic Enablers (GEs), complying with NGSI-v2, including Orion Context Broker, IoT Agent, and QuantumLeap. These components play a pivotal role in core context information management, IoT device connectivity, and time-series database interaction (*FIWARE Catalogue – FIWARE*, 2023).

In this context, we developed FiLiP, a Python Software Development Kit (SDK) designed to accelerate the development of services that interact with the aforementioned FIWARE GEs. FiLiP emerges as a powerful tool, particularly in the domain of energy management systems, where it simplifies development, enhances efficiency, and empowers developers to create reliable and reusable IoT applications, aligning with the digital transformation facilitated by FIWARE.

Motivation

The NGSI-v2 specification defines a standardized RESTful API to represent, exchange and manage context information throughout its entire lifecycle (Cantera Fonseca et al., n.d.). Researchers or application developers can refer to the specifications and build reliable and interoperable data systems. However, NGSI-v2 is a general cross-domain specification and does not provide any domain-specific information models. Although it defines the general data structure of entities, the definition of domain-specific attributes and metadata information remains in the responsibility of the users. Hence, adopting the specification for domain-specific engineering applications requires extensive training in both data modeling and programming of IoT applications. Moreover, the specification has continuously evolved over time. Therefore, enormous efforts are required to, first, define domain specific data models and, second, to comply with the specifications and keep applications and models up-to-date with the latest version of the specification. Thus, although FIWARE provides OpenAPI specifications (*OpenAPI Specification*, 2023) (Cantera Fonseca et al., n.d.), which can be used to automatically generate API clients for various programming languages, there are still challenges that make those auto-generated API clients less reliable:

- The quality of auto-generated code strongly depends on the provided input data, i.e. Ope-

nAPI specification.

- Integrating additional features is generally not viable because manipulating the generated code can result in larger maintenance efforts.
- The generated code from the generic specification does not enable data validation and reasonable error handling, which hinders the development of reliable domain-specific applications.

As a result, these challenges continue to hinder the adoption of FIWARE-based platforms in research fields and industrial applications.

Implementation

To overcome these issues, we present FiLiP (Fiware Library for Python), developed as a reliable API client for the NGSI-v2 API standard. As the name suggests, the library is written in Python and provides a set of client classes for typical recurring GEs for IoT systems. Currently, FiLiP supports Orion Context Broker for central data management, IoT Agent for modular IoT-Interfaces, and QuantumLeap for time-series management. The interactions with the API endpoints are implemented as methods of these corresponding *"Clients"*. For example, the `ContextBrokerClient` implements typical CRUD (create, read, update, and delete) operations for the NGSI-v2 Context Broker, such as creating data entities and subscriptions and retrieving, updating, and deleting while maintaining data integrity and consistency of the underlying data models. By encapsulating API interactions in these specialized clients, FiLiP eliminates the necessity for users to create unreliable data models and consult API documentation for endpoint details. Instead, users can directly invoke clients' methods of FiLiP, thereby automating the composition and dispatch of the requisite CRUD operations. It is worth emphasizing that FiLiP's naming of methods is very intuitive and comes with detailed descriptions. This feature greatly reduces the difficulty of getting started with the FIWARE-based platform and also accelerates the development of applications. When developers need to perform an action, they can simply call the corresponding method from the FiLiP client in a very intuitive way.

To enhance the efficiency of service development, FiLiP offers a range of advanced functionalities. One of them is that FiLiP has a robust implementation of data parsing and validation via the Pydantic library ("Pydantic," 2023). The information models defined by NGSI-v2 API standards are implemented as Pydantic data models in FiLiP. For example, in NGSI-v2 API standards, an entity represents a physical or logical object. It must have an `id` and a `type` to identify itself and can contain attributes holding data about its current state or its relationships to other entities. For that, we implement a Pydantic model, `ContextEntity`, which restricts the data structure of an entity and validates the attribute values by their type. If users or developers put a string data to an attribute in the "Number" type, a validation error will be raised. By actively checking the incoming and outgoing data against predefined data types, schemas, and standards, FiLiP ensures that the data being exchanged adheres to expected formats and structures, which reduces the risk of errors and data corruption. These Pydantic based *"Models"* ensure the quality, integrity, and reusability of the data exchanged between the developed services and FIWARE GEs.

In practice, individual domains often necessitate domain-specific, and even application-specific, data models to meet distinct requirements. The FiLiP *"Models"* facilitate this process by offering parent classes that ensure adherence to FIWARE API standards. Consequently, specific attributes can be defined to construct data models tailored to particular domains. These specialized models serve to validate data structures and attribute values, aligning with both domain-specific requirements and FIWARE API standards. Furthermore, the underlying Pydantic library allows for the export of Pydantic models to `json-schema` format, simplifying the generation of programming language-independent model documentation.

Since *"Clients"* and *"Models"* constitute the core features of FiLiP, the reliability of these components is pivotal to the overall usability and effectiveness of the library. In the realm of

open-source software development, maintaining code quality and dependability is paramount. FiLiP accomplishes this by implementing 82 test cases based on the Python unit testing framework, `unittest`, currently covering 82 % of the code base ("Unitest," 2023). This way, the main features of FiLiP are comprehensively tested after every code update.

In general, the abstraction of API endpoints, implementation of data parsing, validation, and the comprehensive testing workflow enable FiLiP to simplify the development process of applications by reducing the need for API specification reading, manual data checking, and functional validation within the application code. Thus, FiLiP effectively avoids the use for unreliable boilerplate code, thereby reducing overall development and maintenance costs.

To shorten the learning curve for developers and researchers, we provide examples for individual functions as entry points and comprehensive workshop materials for developing a complete FIWARE-based application leveraging FiLiP (Thomas Storek & Müller, 2023). Hence, developers can focus on building innovative IoT solutions, e.g., energy management services, without the burden of developing reliable data exchange interfaces.

Use Case

FiLiP has already been used to deploy various cloud-based building energy management systems (T. Storek et al., 2019), (Kümpel et al., 2019), (Blechmann et al., 2023). Among those, one simplified use case is to regulate the indoor air temperature of an office via IoT-enabled devices and a simple controller. Figure 1 shows an office in a building equipped with two smart devices: a smart temperature sensor and a smart electrical heater, which can send measurements or receive commands, respectively, via Message Queuing Telemetry Transport (MQTT) protocol. FiLiP provides reliable functionalities to deploy and commission a control service efficiently.

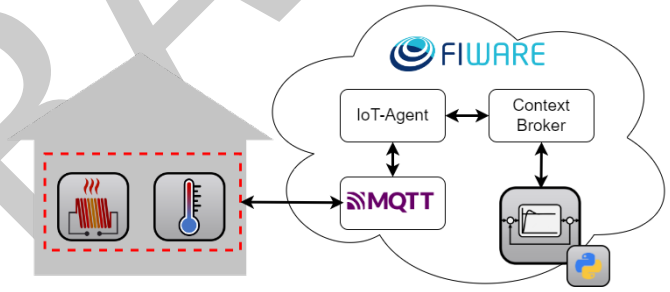

Figure 1: Use Case Overview: Intelligent Room Temperature Regulation.

To establish the communication workflow, entities and IoT devices must be registered in the corresponding FIWARE components, i.e., Context Broker and IoT Agent. In the following example, by using FiLiP, an office room can be registered using `post_entity()` method of the `ContextBrokerClient`. An entity object of the office is created using the `ContextEntity` class, which specifies those mandatory properties of the office and can validate their compliance. Similarly, a temperature sensor and a heater can also be registered as IoT devices by calling `post_device()` method of the `IoTAgentClient` of FiLiP. The classes `Device` and `DeviceAttribute` help to create the device models more efficiently and correctly. Especially for the various setting parameters of `Device`, e.g. `transport` and `protocol`, the validation can avoid ambiguity and significantly simplifies the development.

```
# create entities
office = ContextEntity(id="Office:001", type="Office")
context_broker_client.post_entity(entity=office)
```

```
# provision IoT sensor
t_zone = DeviceAttribute(name='temperature', type="Number")
temperature_sensor = Device(
    device_id='device:001',
    entity_name='TemperatureSensor:001',
    entity_type='TemperatureSensor',
    protocol='IoTA-JSON',
    transport='MQTT',
    apikey=APIKEY,
    attributes=[t_zone])
iot_client.post_device(device=temperature_sensor)

# provision IoT actuator
heating_power = NamedCommand(name="heating_power")
heater = Device(
    device_id='device:002',
    entity_name='Heater:001',
    entity_type='Heater',
    protocol='IoTA-JSON',
    transport='MQTT',
    apikey=APIKEY,
    commands=[heating_power])
iot_client.post_device(device=heater)
```

Besides, FiLiP can be used to add semantic information to the data points. In the example below, the temperature sensor is associated with the office through a relationship labeled `hasSensor`. Another relationship `locatedIn` can further link this office with a building, provided that the entity of that building already exists.

```
# add semantic information
hasSensor = NamedContextAttribute(
    name="hasSensor",
    type="Relationship",
    value=temperature_sensor.entity_name)

locatedIn = NamedContextAttribute(
    name="locatedIn",
    type="Relationship",
    value=building.id)

office.add_attributes(attrs=[hasSensor, locatedIn])
```

To this end, data can be exchanged between the FIWARE-based platform and the smart devices. A PID controller can then be deployed as a cloud service to regulate the indoor temperature of the office. With the help of FiLiP, a communication interface between the controller and the FIWARE GEs can be efficiently established as well.

```
# retrieve data from the temperature sensor
temperature = context_broker_client.get_attribute_value(
    entity_id=temperature_sensor.entity_name,
    entity_type=temperature_sensor.type,
    attr_name=t_zone.name)

# calculate new set point with controller logic
heating_power.value = controller_logic(temperature=temperature)
```

```
# send command to the heater entity
context_broker_client.post_command(
    entity_id=heater.id,
    entity_type=heater.type,
    command=heating_power)
```

Conclusion

The presented use case exemplifies how FiLiP plays a pivotal role in the implementation of IoT-enabled applications using FIWARE GEs. The primary strengths of using FiLiP encompass the following key aspects:

- **Simplified Development**: FiLiP offers developers a straightforward and user-friendly approach to register entities and IoT devices on FIWARE GEs. By providing classes like ContextEntity and Device, FiLiP streamlines the development process. It ensures that essential attributes and parameters are validated, empowering developers to make well-informed decisions without requiring an advanced level of expertise.
- **Efficient Deployment**: FiLiP helps to establish communication interfaces between the FIWARE-based platforms and IoT devices efficiently. Additionally, FiLiP's data validation mechanisms ensure that entities and devices comply with expectations, significantly reducing ambiguity and the risk of errors or misinterpretation.
- **Maintainable Applications**: The dynamic evolution of the NGSI-v2 API poses challenges in managing ad hoc API clients (e.g., those solely relying on the Python `requests` package (Reitz, 2023)). FiLiP addresses this complexity by encapsulating the majority of API interactions within class methods, thereby disentangling application logic from the intricacies of the API. Consequently, applications built with FiLiP inherently exhibit higher levels of reusability and maintainability.
- **Reliable Functionalities**: The reliability of FiLiP's core functionalities is enhanced through a comprehensive testing workflow. A total of 82 test cases are meticulously implemented to ensure that the majority of features undergo rigorous testing after every code update. This proactive approach to testing not only guarantees code quality but also allows developers to focus more on the development of FiLiP-based applications.

In conclusion, FiLiP is a versatile and dependable tool that simplifies the development process, enhances efficiency in deploying applications, ensures their maintainability, and provides a strong foundation for reliable functionalities. With FiLiP, developers can confidently create innovative and IoT-enabled applications within the landscape of FIWARE. Researchers can investigate smart solutions in various domains, including energy management and beyond.

Acknowledgements

We gratefully acknowledge the financial support provided by the Federal Ministry for Economic Affairs and Climate Action (BMWK), promotional references 03ET1495A, 03ET1551A, 0350018A, 03ET1561B, 03EN1030B.

References

Blechmann, S., Sowa, I., Schraven, M. H., Streblow, R., Müller, D., & Monti, A. (2023). Open source platform application for smart building and smart grid controls. *Automation in Construction*, *145*, 104622. https://doi.org/10.1016/j.autcon.2022.104622

Cantera Fonseca, J. M., Márquez, F. G., & Jacobs, T. (n.d.). *FIWARE-NGSI v2 Specification*.

FIWARE catalogue – FIWARE. (2023, June 26). https://www.fiware.org/catalogue/

Kümpel, A., Storek, T., Baranski, M., Schumacher, M., & Müller, D. (2019). A cloud-based operation optimization of building energy systems using a hierarchical multi-agent control. *J. Phys.: Conf. Ser.*, *1343*(1), 012053. https://doi.org/10.1088/1742-6596/1343/1/012053

OpenAPI specification. (2023, November 6). https://www.openapis.org/

Pydantic: Data validation using python type hint. (2023). In *GitHub repository*. GitHub. https://github.com/pydantic/pydantic

Reitz, K. (2023). Requests: HTTP for humans. In *GitHub repository*. GitHub. https://github.com/psf/requests

Storek, T., Lohmöller, J., Kümpel, A., Baranski, M., & Müller, D. (2019). Application of the open-source cloud platform FIWARE for future building energy management systems. *J. Phys.: Conf. Ser.*, *1343*(1), 012063. https://doi.org/10.1088/1742-6596/1343/1/012063

Storek, Thomas, & Müller, D. (2023). *FIWARE for Energy System Engineers: An Introduction to FIWARE-based Applications in Python*. https://doi.org/10.18154/RWTH-2022-11779

Unitest: Unit testing framework. (2023). In *Python Documentation*. Python. https://docs.python.org/3/library/unittest.html