Towards Machine-Actionable FAIR-DOs: Developing a Model for Type-Associated FAIR-DO Operations

Maximilian Inckmann

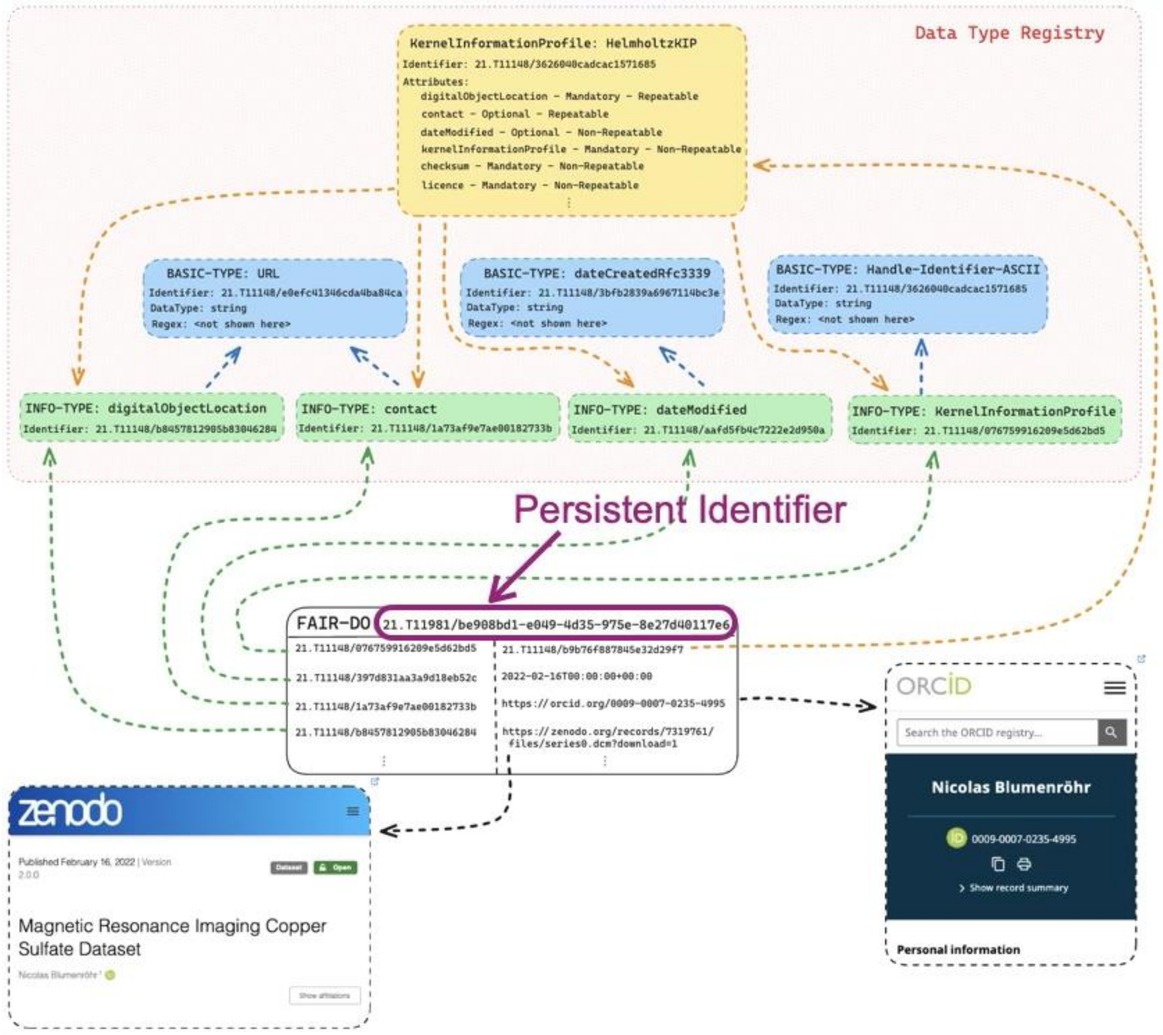
RDA FAIR Digital Object Fabric IG meeting

April 24th 2025

Current FAIR-DO realization

- Harmonized information entry-point
- References to various resources (aka. bitstreams)
- Use of persistent identifiers and Handle records
- Typed values
  → PID-BasicInfoTypes and PID-InfoType (PIT)
- Typed structure of FAIR-DOs
  → Kernel Information Profiles (KIP)
- Registry for data types and KIPs
  → ePIC/EOSC Data Type Registry (DTR)

FAIR-DOs are machine-interpretable!



Data Type Registry
KernelInformationProfile: HelmholtzKIP
Identifier: 21.T11148/3626040cadcac1571685
Attributes:
digitalObjectLocation – Mandatory – Repeatable
contact – Optional – Repeatable
dateModified – Optional – Non-Repeatable
kernelInformationProfile – Mandatory – Non-Repeatable
checksum – Mandatory – Non-Repeatable
licence – Mandatory – Non-Repeatable
BASIC-TYPE: URL
Identifier: 21.T11148/e0efc41346cda4ba84ca
DataType: string
Regex: <not shown here>
BASIC-TYPE: dateCreatedRfc3339
Identifier: 21.T11148/3bfb2839a6967114bc3e
DataType: string
Regex: <not shown here>
BASIC-TYPE: Handle-Identifier-ASCII
Identifier: 21.T11148/3626040cadcac1571685
DataType: string
Regex: <not shown here>
INFO-TYPE: digitalObjectLocation
Identifier: 21.T11148/b8457812905b83046284
INFO-TYPE: contact
Identifier: 21.T11148/1a73af9e7ae00182733b
INFO-TYPE: dateModified
Identifier: 21.T11148/aafd5fb4c7222e2d950a
INFO-TYPE: KernelInformationProfile
Identifier: 21.T11148/076759916209e5d62bd5
Persistent Identifier
FAIR-DO 21.T11981/be908bd1-e049-4d35-975e-8e27d40117e6
21.T11148/076759916209e5d62bd5
21.T11148/b9b76f887845e32d29f7
21.T11148/397d831aa3a9d18eb52c
2022-02-16T00:00:00+00:00
21.T11148/1a73af9e7ae00182733b
https://orcid.org/0009-0007-0235-4995
21.T11148/b8457812905b83046284
https://zenodo.org/records/7319761/files/series0.dcm?download=1
zenodo
Published February 16, 2022 | Version 2.0.0
Dataset
Open
Magnetic Resonance Imaging Copper Sulfate Dataset
Nicolas Blumenröhr
Show affiliations
ORCID
Search the ORCID registry...
Nicolas Blumenröhr
0009-0007-0235-4995
Show record summary
Personal information

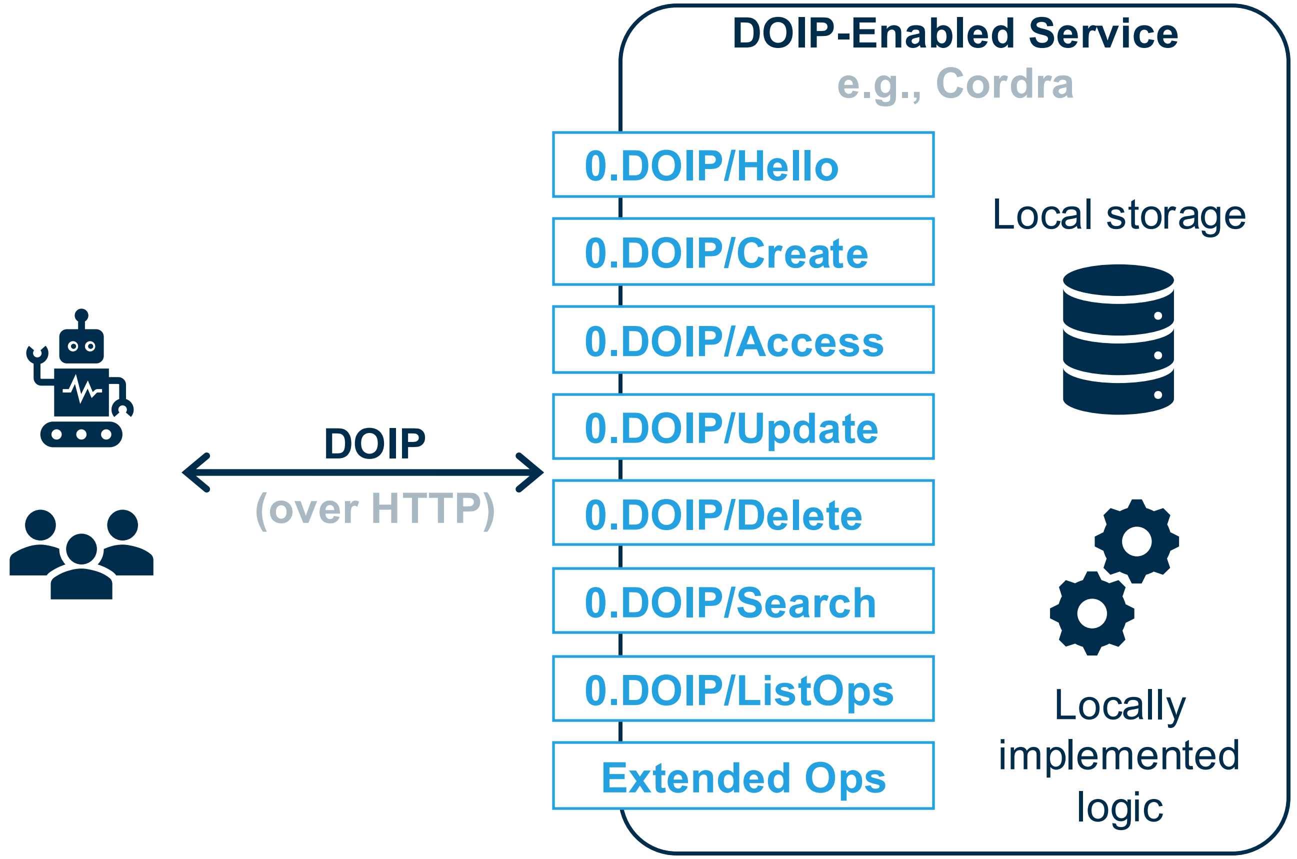
Excursion: Service-oriented Operations

Idea: Machine-interpretable API of services and enable interaction based on FAIR-DO principles

- Realized with DOIP operations
- **The logic is implemented by every service!**
- **Descriptions of the input/outputs referenced via PID and stored in DTR**
- Actions performed with/on FAIR-DOs on a specific service → usually using resources within the service
- Comparable to a typed API (e.g., tRPC, HATEOAS REST API)

→ Limited to supporting services and already done (e.g., Cordra, Nicolas's demonstrator)

Examples: retrieve and validate FAIR-DOs, get available operations for FAIR-DOs, execute type-associated FAIR-DO Operations on FAIR-DOs

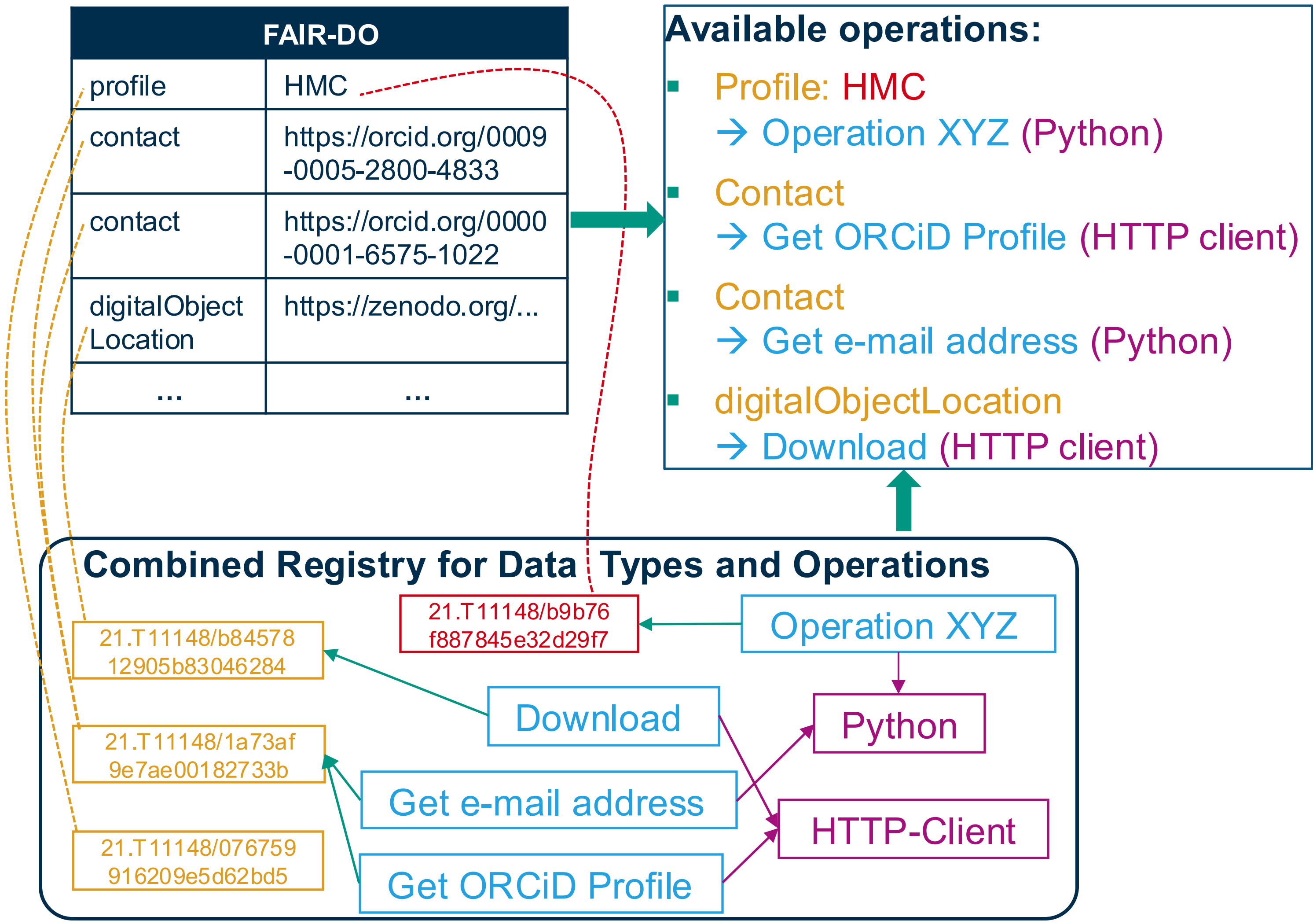


New: Type-associated Operations

Idea: Describe technology-agnostic operations in an executable manner and associate them with Data Types

- Focuses on the contents of FAIR-DOs
- Associate FAIR-DO-Operations with data types and/or KIPs in the DTR
- Dynamically add available operations for data types
- Technology- and environment-agnostic operations
- Separation of specific operations from technologies to enhance reusability

Examples: evaluate license compatibility, process data, execute arbitrary scripts



FAIR-DO	
profile	HMC
contact	https://orcid.org/0009-0005-2800-4833
contact	https://orcid.org/0000-0001-6575-1022
digitalObjectLocation	https://zenodo.org/...
...	...

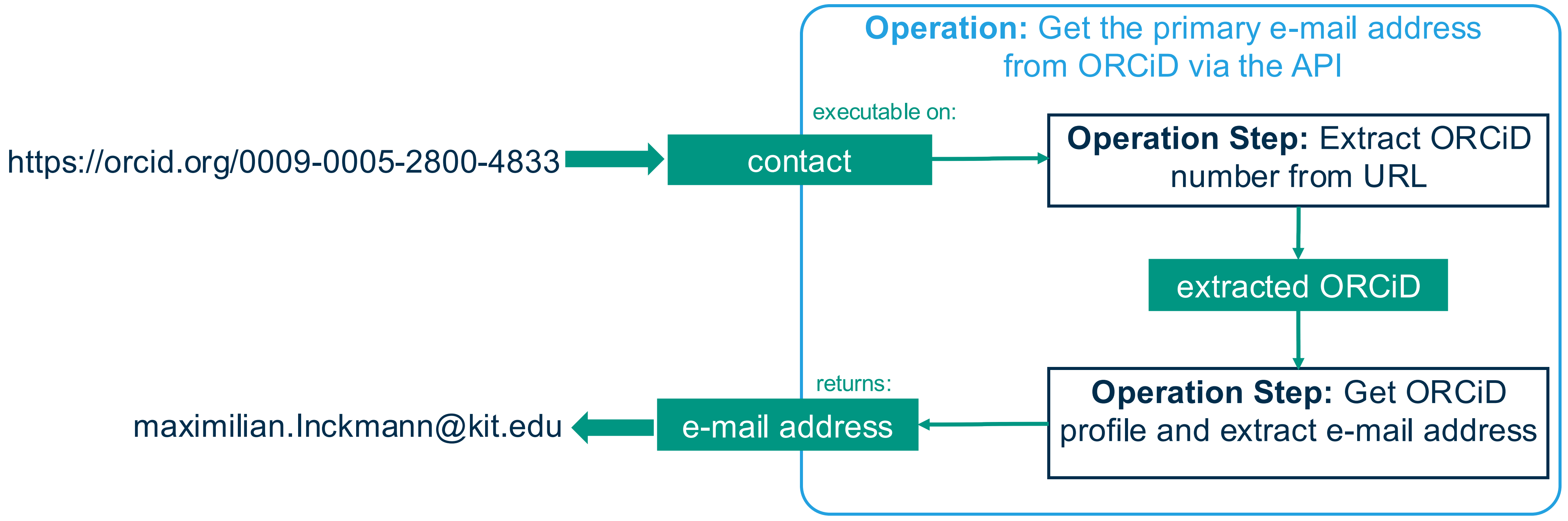
Available operations:

- Profile: HMC
  → Operation XYZ (Python)
- Contact
  → Get ORCiD Profile (HTTP client)
- Contact
  → Get e-mail address (Python)
- digitalObjectLocation
  → Download (HTTP client)

Combined Registry for Data Types and Operations
21.T11148/b9b76f887845e32d29f7
21.T11148/b84578 12905b83046284
21.T11148/1a73af 9e7ae00182733b
21.T11148/076759 916209e5d62bd5
Operation XYZ
Download
Get e-mail address
Get ORCiD Profile
Python
HTTP-Client

Data Type	Attribute	Operation
Profile	Attribute Mapping	Technology Interface

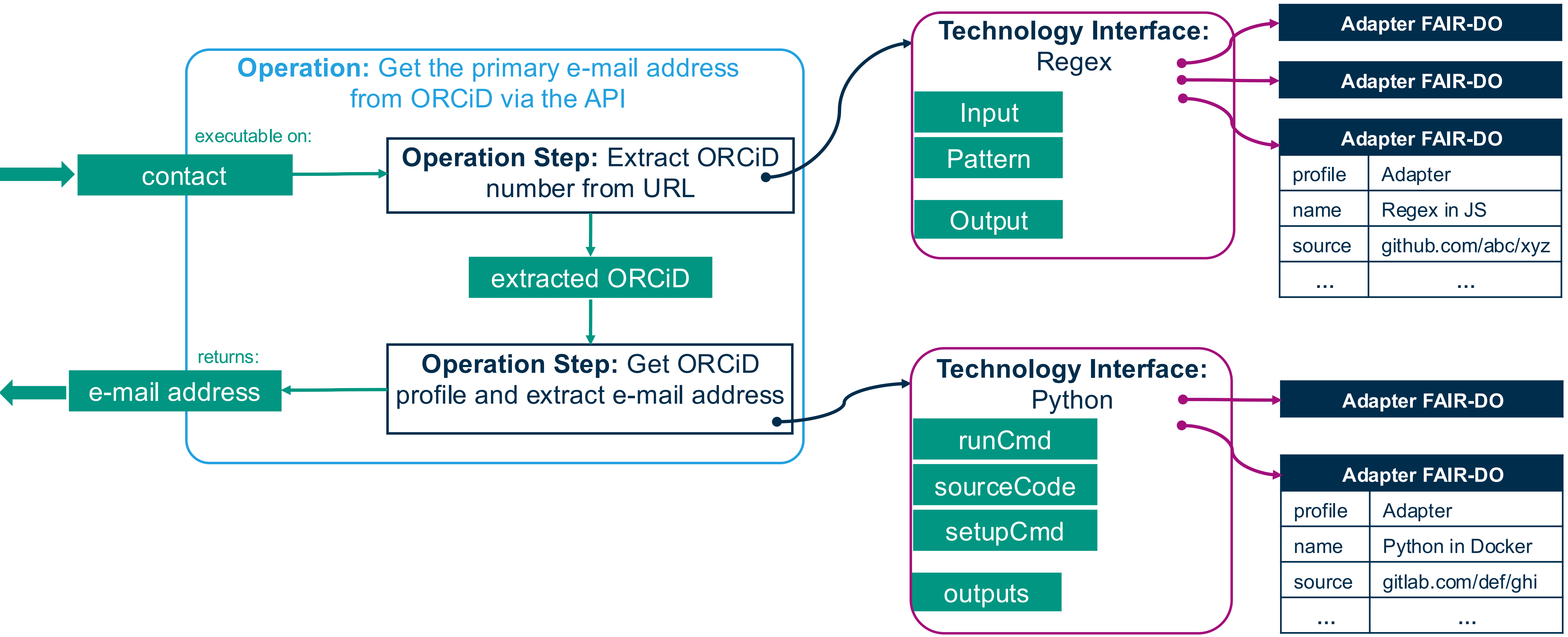
Example for a type-associated FAIR-DO Operation



Operation: Get the primary e-mail address from ORCiD via the API
executable on:
https://orcid.org/0009-0005-2800-4833
contact
Operation Step: Extract ORCiD number from URL
extracted ORCiD
Operation Step: Get ORCiD profile and extract e-mail address
returns:
e-mail address
maximilian.Inckmann@kit.edu

Data Type
Profile
Attribute
Attribute Mapping
Operation
Technology Interface

Example for a type-associated FAIR-DO Operation



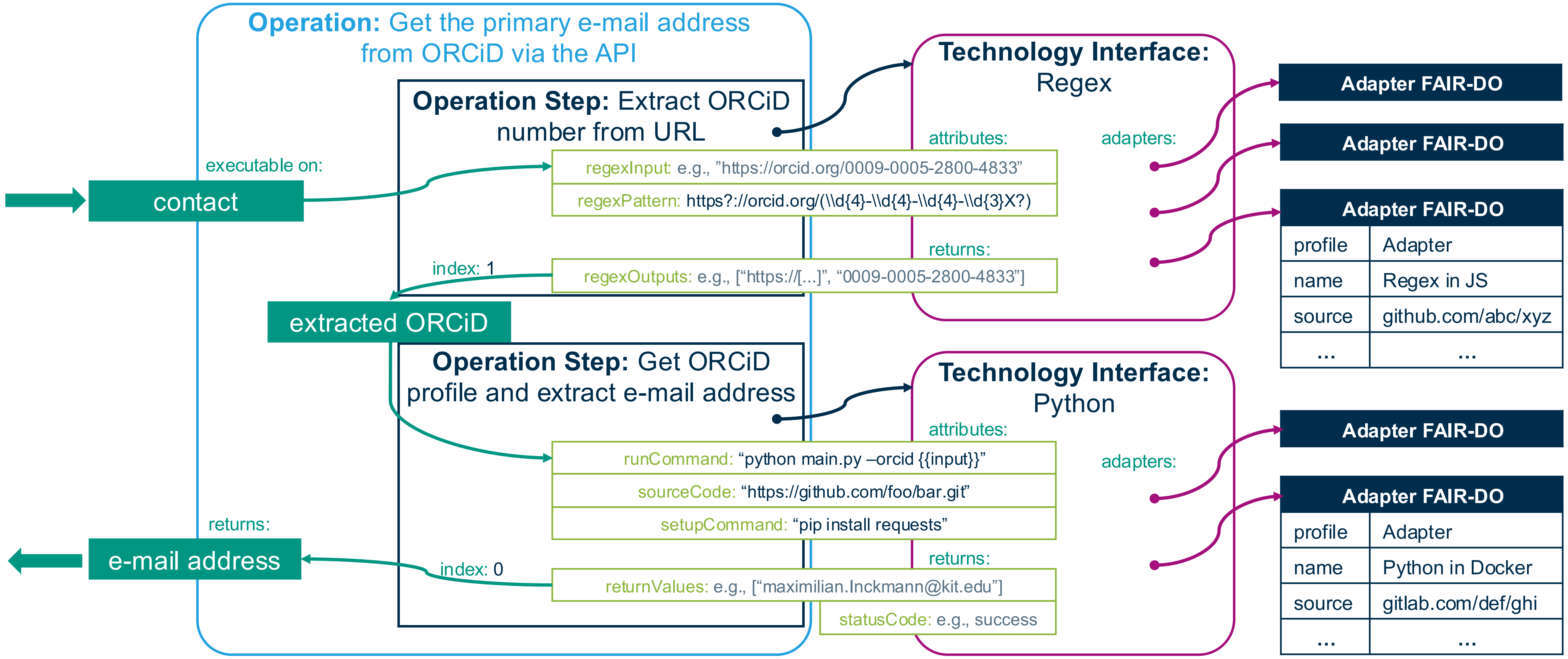
Operation: Get the primary e-mail address from ORCiD via the API
executable on:
contact
Operation Step: Extract ORCiD number from URL
extracted ORCiD
Operation Step: Get ORCiD profile and extract e-mail address
returns:
e-mail address
Technology Interface: Regex
Input
Pattern
Output
Technology Interface: Python
runCmd
sourceCode
setupCmd
outputs
Adapter FAIR-DO
Adapter FAIR-DO
Adapter FAIR-DO
Adapter FAIR-DO
Adapter FAIR-DO

Adapter FAIR-DO	
profile	Adapter
name	Regex in JS
source	github.com/abc/xyz
…	…

Adapter FAIR-DO	
profile	Adapter
name	Python in Docker
source	gitlab.com/def/ghi
…	…

Data Type	Attribute	Operation
Profile	Attribute Mapping	Technology Interface

Example for a type-associated FAIR-DO Operation


Operation: Get the primary e-mail address from ORCiD via the API
executable on:
contact
Operation Step: Extract ORCiD number from URL
index: 1
extracted ORCiD
Operation Step: Get ORCiD profile and extract e-mail address
returns:
e-mail address
index: 0
Technology Interface: Regex
attributes:
adapters:
returns:
Technology Interface: Python
attributes:
adapters:
returns:
Adapter FAIR-DO
Adapter FAIR-DO
Adapter FAIR-DO
Adapter FAIR-DO
Adapter FAIR-DO

regexInput: e.g., "https://orcid.org/0009-0005-2800-4833"

regexPattern: https?://orcid.org/(\\d{4}-\\d{4}-\\d{4}-\\d{3}X?)

regexOutputs: e.g., ["https://[...]", "0009-0005-2800-4833"]

runCommand: "python main.py –orcid {{input}}"

sourceCode: "https://github.com/foo/bar.git"

setupCommand: "pip install requests"

returnValues: e.g., ["maximilian.Inckmann@kit.edu"]

statusCode: e.g., success

Adapter FAIR-DO

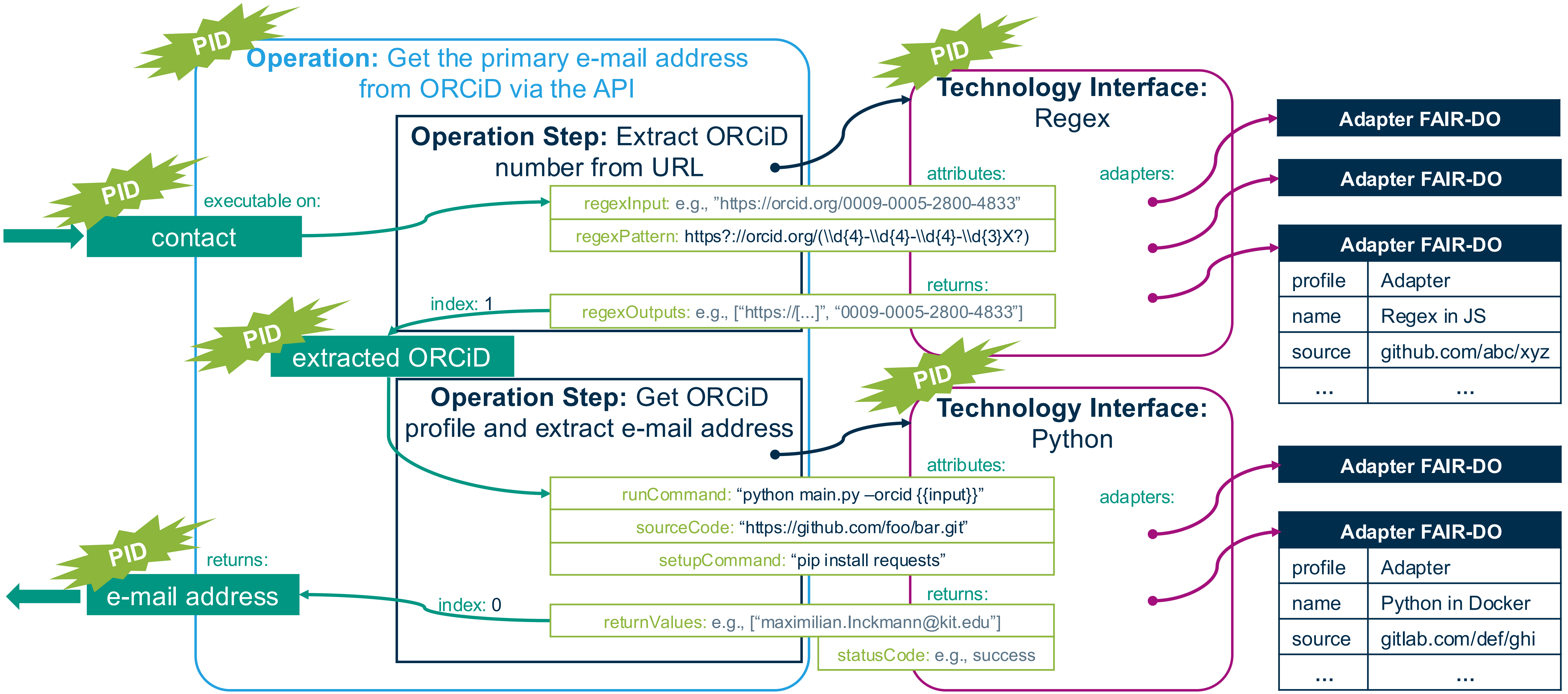
profile	Adapter
name	Regex in JS
source	github.com/abc/xyz
...	...

Adapter FAIR-DO

profile	Adapter
name	Python in Docker
source	gitlab.com/def/ghi
...	...

Data Type	**Attribute**	**Operation**
Profile	**Attribute Mapping**	**Technology Interface**

Example for a type-associated FAIR-DO Operation


PID
Operation: Get the primary e-mail address from ORCiD via the API
Operation Step: Extract ORCiD number from URL
executable on:
contact
index: 1
extracted ORCiD
Operation Step: Get ORCiD profile and extract e-mail address
returns:
e-mail address
index: 0
Technology Interface: Regex
attributes:
adapters:
returns:
Technology Interface: Python
attributes:
adapters:
returns:
Adapter FAIR-DO
Adapter FAIR-DO

regexInput: e.g., "https://orcid.org/0009-0005-2800-4833"

regexPattern: https?://orcid.org/(\\d{4}-\\d{4}-\\d{4}-\\d{3}X?)

regexOutputs: e.g., ["https://[...]", "0009-0005-2800-4833"]

runCommand: "python main.py –orcid {{input}}"

sourceCode: "https://github.com/foo/bar.git"

setupCommand: "pip install requests"

returnValues: e.g., ["maximilian.Inckmann@kit.edu"]

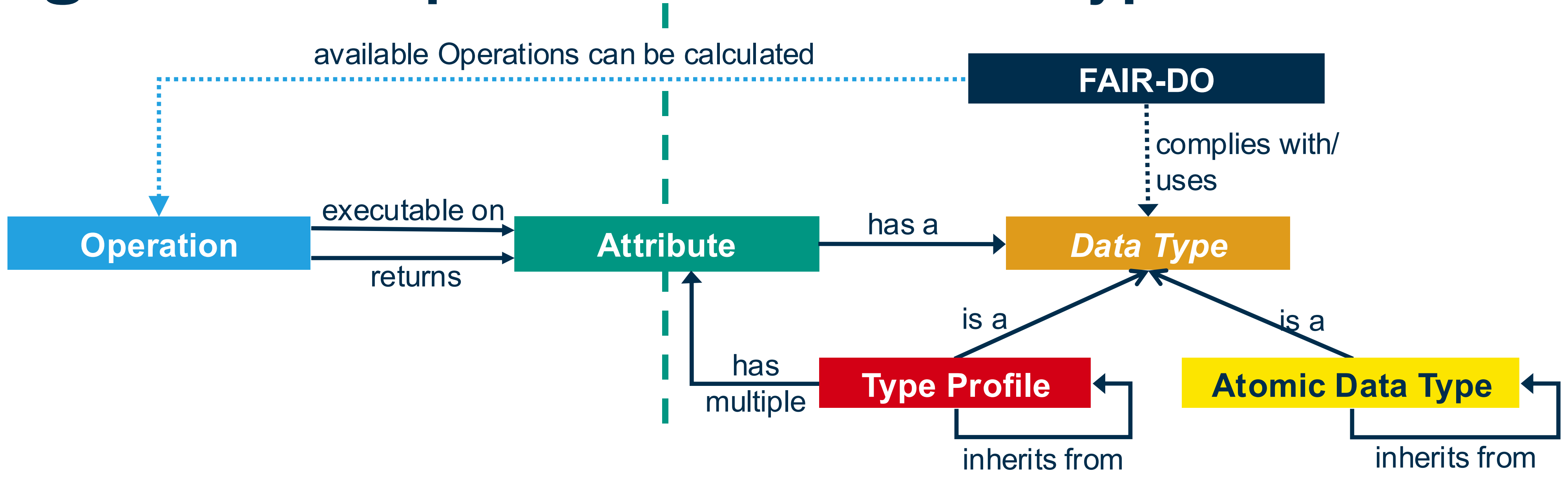
statusCode: e.g., success

Adapter FAIR-DO	
profile	Adapter
name	Regex in JS
source	github.com/abc/xyz
...	...

Adapter FAIR-DO	
profile	Adapter
name	Python in Docker
source	gitlab.com/def/ghi
...	...

Data Type	Attribute	Operation
Profile	Attribute Mapping	Technology Interface

Associating FAIR-DO Operations with Data Types



Data Type

- Define a generic term for all typing mechanisms → reduces redundancy
- Added inheritance mechanisms for better reusability
- **Atomic Data Type** → syntax of single values (string, number, integer, boolean) formerly: PID-BasicInfoType
- **Type Profile** → combination of multiple Attributes formerly: PID-InfoType and KIP

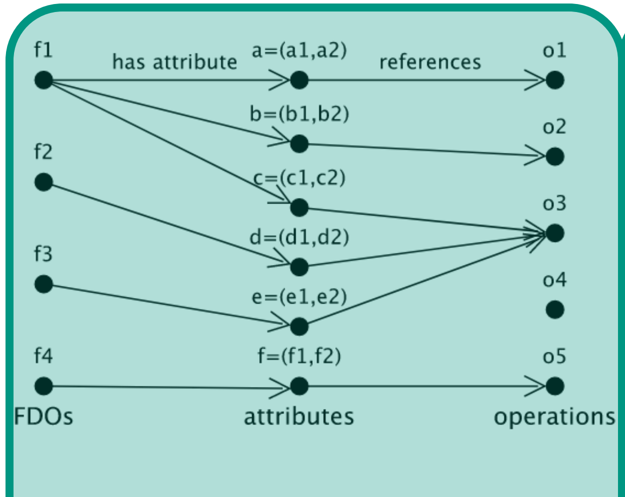
Attribute

- Name, description, cardinality
- Refers to a Data Type (Atomic Data Type for simple values; Type Profile for complex values)
- Single connection between operations and data types → association mechanism

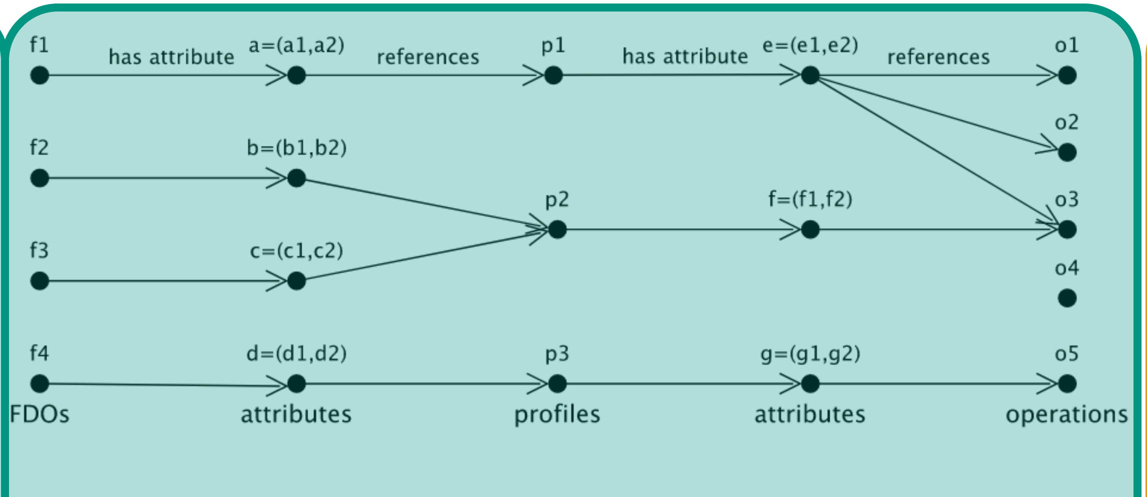
FAIR-DO

- Complies to a Type Profile and uses Atomic Data Types
- Set of available Operations computable via Data Types and Attributes

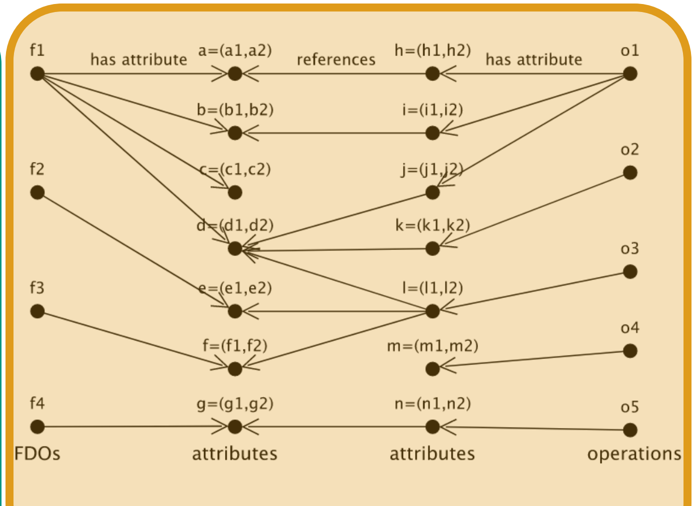
Associating FAIR-DO Operations with Data Types



(a) Record typing



(b) Profile typing



(c) Attribute typing

Source: Blumenröhr, N., Böhm, J., Ost, P., Kulüke, M., Wittenburg, P., Blanchi, C., Bingert, S. and Schwardmann, U., 2025. *A Comparative Analysis of Modeling Approaches for the Association of FAIR Digital Objects Operations*. https://doi.org/10.48550/arXiv.2504.05361.

- Reference FAIR-DO Ops from PID records via their PID → Record Typing ✓
- Operations executable on a single Attribute pointing to a Type Profile → Profile Typing ✓
- Operations executable on a single Attribute pointing to a Data Type → Attribute typing (✓)

Still missing: Operations executable on multiple Attributes → Duck-typing 🦆

What would we need (to change) to achieve this?

An Integrated Data Type and Operations Registry with an Inheritance System

- Central management and association of Operations and Data Types
- Inheritance mechanisms and polymorphic behavior → enhance reusability and flexibility
- Complex logic beyond the capabilities of JSON Schema → e.g., rule-based validation; inheritance
- Optimization based on the ePIC/EOSC DTR’s data models
- Provide FAIR-DOs for entries in this new registry → improve FAIRness and consistency
- Some of these features are prototypically realized in IDORIS

FAIR-DO Operation execution service with Adapters

- Computation of Attribute values for the Technology Interfaces
- Execution management and scheduling
- Adapters for multiple technologies (e.g., Python, JavaScript, Regex, HTTP)
- Ideally able to communicate via DOIP

Conclusions

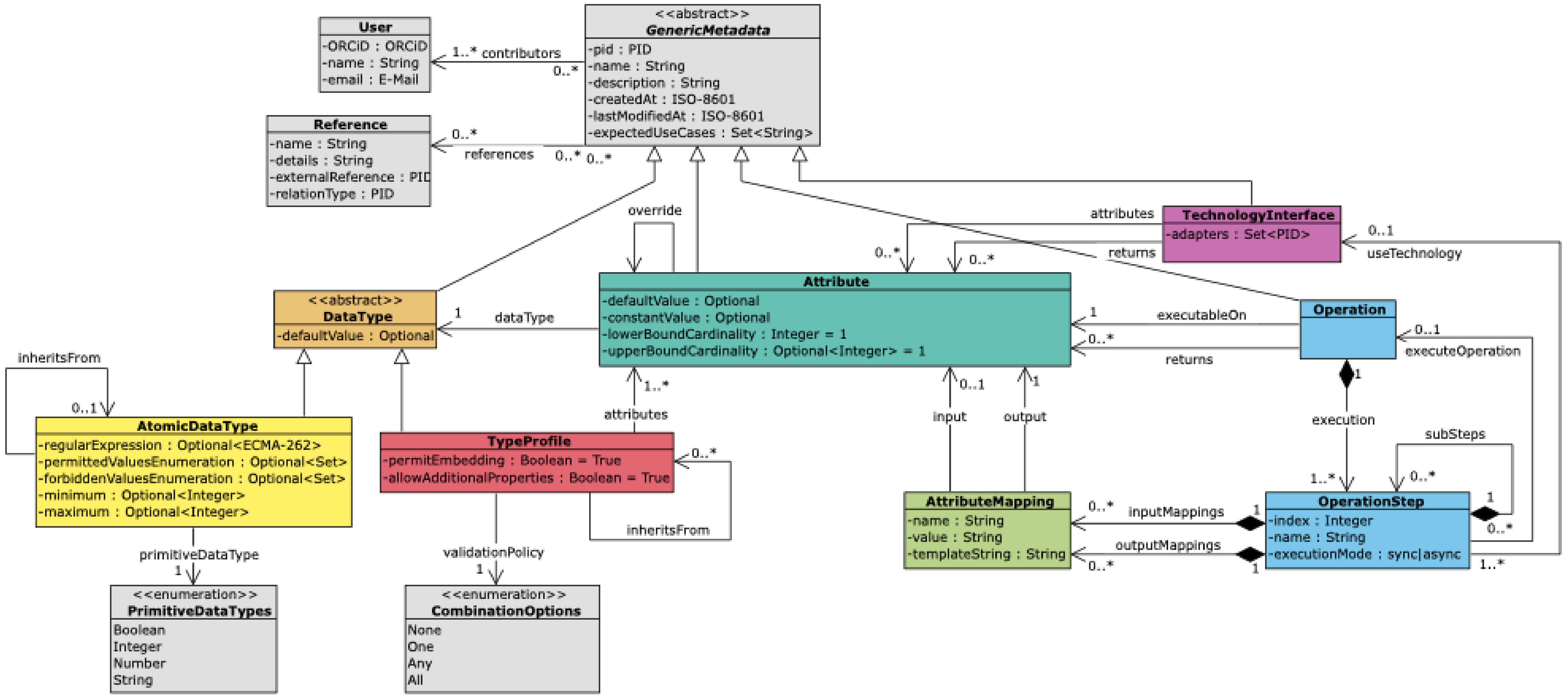
- Presented a model for technology-agnostic and reusable FAIR-DO Operations
- Machine-executable description of Operations with multiple steps → Workflows
- Developed an association mechanism between Operations and Data Types

Research aspects not (explicitly) shown in this presentation:

- Attachment slide: Complete UML diagram of the typing model
- Type Profiles for validated inter FAIR-DO relationships
- Attribute-overriding and polymorphic behavior
- Technical details of IDORIS (prototypical implementation)

Status: Realization of Prototype is work in progress

Attachment: Complete UML diagram of the data model


User
-ORCiD : ORCiD
-name : String
-email : E-Mail
1..* contributors
0..*
<<abstract>>
GenericMetadata
-pid : PID
-name : String
-description : String
-createdAt : ISO-8601
-lastModifiedAt : ISO-8601
-expectedUseCases : Set<String>
Reference
-name : String
-details : String
-externalReference : PID
-relationType : PID
0..*
references
0..*
0..*
override
attributes
TechnologyInterface
-adapters : Set<PID>
0..1
useTechnology
returns
0..*
0..*
Attribute
-defaultValue : Optional
-constantValue : Optional
-lowerBoundCardinality : Integer = 1
-upperBoundCardinality : Optional<Integer> = 1
<<abstract>>
DataType
-defaultValue : Optional
1
dataType
Operation
1
executableOn
0..*
returns
0..1
executeOperation
inheritsFrom
0..1
AtomicDataType
-regularExpression : Optional<ECMA-262>
-permittedValuesEnumeration : Optional<Set>
-forbiddenValuesEnumeration : Optional<Set>
-minimum : Optional<Integer>
-maximum : Optional<Integer>
1..*
attributes
TypeProfile
-permitEmbedding : Boolean = True
-allowAdditionalProperties : Boolean = True
0..*
inheritsFrom
0..1
input
1
output
1
execution
subSteps
1..*
0..*
AttributeMapping
-name : String
-value : String
-templateString : String
0..*
inputMappings
1
outputMappings
0..*
1
OperationStep
-index : Integer
-name : String
-executionMode : sync|async
1
0..*
1..*
primitiveDataType
1
<<enumeration>>
PrimitiveDataTypes
Boolean
Integer
Number
String
validationPolicy
1
<<enumeration>>
CombinationOptions
None
One
Any
All