Using OpenAPI for IVOA standards

As part of our work for the SKA Regional Centre Network (SRCNet) we are developing a web-service to enable users to execute their code on compute platforms close to where their data is stored – *"moving the code to the data"*

SRC | Net
SKAO Regional Centre Network

International Virtual Observatory (IVOA) Execution Broker

Several members of the IVOA are developing science platforms to enable users to run data analysis code on compute platforms co-located with the data they publish. By proposing the Execution Broker service as an IVOA standard we hope that more sites will adopt it as an entry point for running data analysis tasks, increasing the range of science platforms that astronomers can use to perform multi-wavelength analysis of their data.

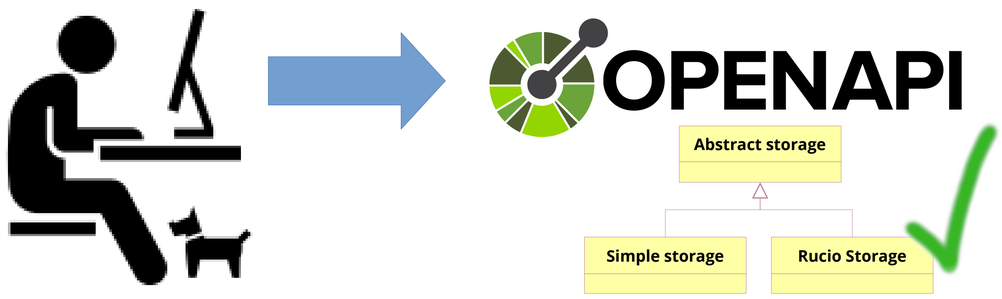
OpenAPI interface definitions

Within the IVOA the Protocol Transition Team are working on updating the IVOA processes to make them compatible with modern development tools and best practice.

As part of this process, we have been evaluating the OpenAPI interface definition language for creating machine readable service descriptions to augment and replace parts of the text descriptions that are used to define the current set of IVOA standards.

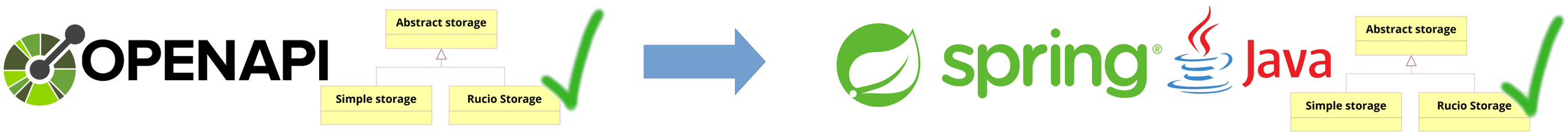
What worked

We were able to use the OpenAPI interface definition language to accurately describe the Execution Broker service interface and data model in a machine and human readable way.



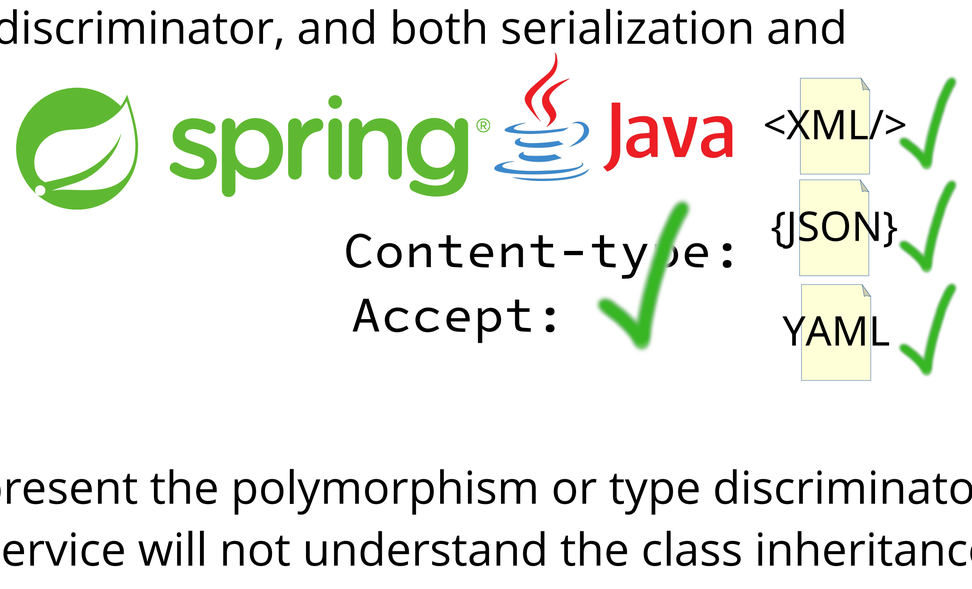
Defining a complex data model like the Execution Broker using handwritten text is extremely difficult to get right. **Using OpenAPI to define the data model made it much easier to experiment with different message structures.**

Generating the Java classes to implement the service API and data model in the Spring framework worked well. Including support for polymorphic types in the data model and messages.



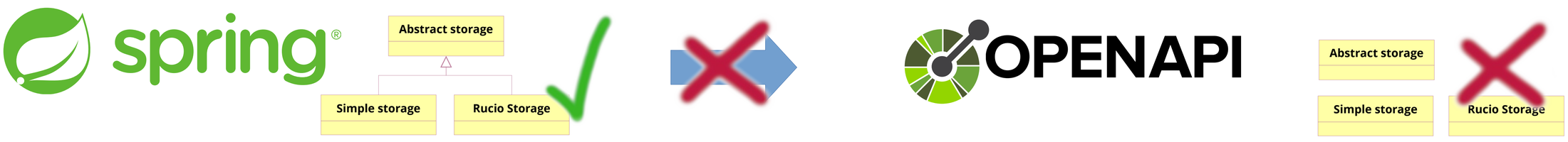
OpenAPI made it easy to define polymorphic types using a discriminator property. The OpenAPI tooling generated code with Java annotations to implement the type discriminator, and both serialization and deserialization worked automatically.

OpenAPI has good support for HTTP content negotiation, making it easy to create a service that can handle inputs and outputs in JSON, YAML and XML from the same data model.

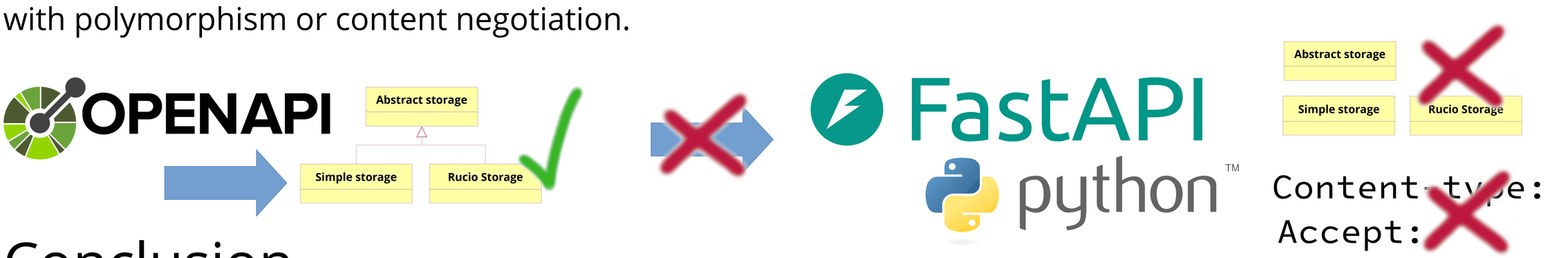


What didn't work

The OpenAPI generator in the Spring framework does not represent the polymorphism or type discriminator in its output. Which means that clients generated from the live service will not understand the class inheritance.



The code generator for the Python FastAPI framework does not produce valid code for a complex data model with polymorphism or content negotiation.



Conclusion

Defining a complex data model like the Execution Broker using handwritten text is extremely difficult to get right. Using OpenAPI to define the data model made the standard document much more accurate and clearer to understand. **This alone was worth the investment in time to write the OpenAPI definition.**

The code generation tools for Java are mature and produce repeatable and accurate results.

The code generation tools for Python are still evolving and don't (yet) produce good results

OpenAPI initiative
https://www.openapis.org/

OpenAPI specification
https://swagger.io/specification/

Java Spring framework
https://spring.io/

Python FastAPI framework
https://fastapi.tiangolo.com/

Dave Morris
dave.morris@manchester.ac.uk