Exploiting Service-Discovery and OpenAPI in Multi-Agent MicroServices (MAMS) Applications

Eoin O'Neill and Rem W. Collier

University College Dublin, Dublin 4, Ireland
eoin.o-neill.3@ucdconnect.ie,rem.collier@ucd.ie

Abstract. One of the key benefits of the MAMS [14,19,13] architecture is to allow agents to make use of the software engineering community's industry standard technology while being deployed in a microservices architecture. This paper is going to showcase a tool that allows MAMS agents to utilise an industry standard discovery tool to interact with a microservice based on the OpenAPI Specification document that describes the service. This interaction will be based on the "shape" of the service which is identified by the accepted HTTP verbs at the various endpoints. This tool also identifies the pitfalls associated with the current industry standard with regard to service descriptions and how they could be improved through the introduction of Linked Data and use of specifications such as *Hydra* and *Hypermedia Controls Ontology* (HCTL) to make a push from ***machine-readable*** towards ***machine-understandable***.

Keywords: Multi-Agent MicroServices (MAMS) · OpenAPI · Hydra · Hypermedia Controls · Signifiers · ThingDescriptions

1 Introduction

One of the key concepts in a microservice (MS) architecture, is the notion of *bounded context*. This states that each MS, following the Domain Driven Design [5,7] principle, is to provide a singular 'business' functionality. With the shift in software engineering from monolithic software structures towards service-oriented architectures, the integration of microservices is a key issue. The standard specification for describing an API is currently the OpenAPI specification (OAS) [1]. These descriptions, although defined as being "machine-readable" are available in formats that provide no context. The documents themselves are geared towards the consumer having a level of tacit knowledge with regards to integrating the services and the protocols and domain knowledge associated with doing so. The main goal of this research is to allow agents to situate themselves in a microservices ecosystem, and through the use of a service-discovery tool,

[1] OpenAPI Specification

each service can make itself known on the network, describe itself to possible consumers of the service and provide a description so that any and all consumers can learn how to interact with this service solely based on it's description.

We have built a tool that allows agents deployed in a MS context to consume the OAS document of a service registered with an industry standard service-discovery tool in order to facilitate interaction between software agents and a service in a more generalised form in order to conform to the *loose-coupling* principle of a microservice architecture. Through the use of this tool, we can see that from an agent perspective, this standard is not fit for purpose as it does not provide enough context with regards to the interaction which led to the implementation of vocabularies such as Hydra [11] and Hypermedia Controls Ontology (HCTL)[2]. The paper is laid out as follows, Section 2 will discuss the related work in this area and why this work is relevant. Section 3 will discuss the implementation of our tool, followed by our conclusion.

2 Related Work

Roy Fielding stated in [2] "*RESTful applications are, at all times, encouraged to use human-meaningful, hierarchical identifiers, in order to maximise the serendipitous use of the information beyond what is anticipated by the original application.*" If we provide semantically enriched, "machine-understandable" descriptions of services and imbue agents with the ability to integrate them how they deem fit, then we can also 'maximise the serendipitous use of' the applications themselves. We present work that has been done on extending service descriptions in order to facilitate interaction and ease of integration. The research presented by Yang et al in [20] presents a tool called D2Spec that iterates through a Web API specification and determines the number of characteristics that the specification includes. These characteristics include the Base IRI of the Application, the HTTP methods used within the application and also generates path templates to be utilised. Guo et al[8] have established a service called *APIphany* which tries to achieve type directed program synthesis by semantically describing the types required and returned by APIs. They manage to achieve this by means of two methods; firstly, they create *witnesses* from the OAS document of the API and run a test suite in a sandbox environment and secondly, they observe live API traffic. From here, they rank the APIs suitability to that of the user's needs.

In [3], Ciortea et al. present research that proposes agents creating a mashup of services and devices as a result of their goal-driven behaviour. Agents are initialised with pre-compiled mashups and cooperate at runtime in order to achieve their goals. This work showcases a similar goal of enabling agents with enough information at runtime to achieve their goals, but in an IoT context. The research presented in [16] presents a system that parses an OAS document, generates an OWL-S ontology for each service that is present in the OAS document. This

[2] https://www.w3.org/2019/wot/hypermedia

research shows the necessity for such translations and the need for a parallel standard to exist in order to establish *machine-understandability*. Furthermore, the work presented in [12] showcases a system that consumes OAS documents, stores them in a relational database and uses RDRML in order to convert the relational database entries into RDF format in order to be stored in a knowledge graph. The work detailed in [9] shows an attempt to bridge the gap between Linked Data and REST-based architectures, using the OAS document as the medium. Furthermore, Espinoza et al [4] have implemented a system that translates from the Web Ontology Language (OWL) into OAS document (OAS document) documents in order to facilitate ease of use between web developers and users of the semantic web. These works show the level of importance being placed on introducing Linked Data concepts to API descriptions.

As mentioned by Bogner et. al in their paper entitled "Industry practices and challenges for the evolvability assurance of microservices", one of the biggest challenges with regards to building Microservices architectures and assuring their sustainability is *Microservices Integration*, according to industry practitioners. This is where we see an opportunity for agents that are able to consume service description documents, such as the OAS document, and interact with resources in an ad-hoc manner to act as a bridge between services in order to enhance loose-coupling and evolvability of individual services.

3 Demonstrating the Approach

In order to allow an agent to reason about a given service, it is essential that it first be able to develop a logical depiction of that service. The shape of a resource is created based on the IRI of each of the APIs endpoints and the HTTP verbs accepted at each of these endpoints, as described in the associated OAS document. The `"paths"` section of the OAS document provides this information, an example of which is displayed below:


```
{"paths":
  {"/game":
   {"get":
    {"tags":["Gameplay"],
     "summary":"Get the current number which
     represents the state of the game.",
     "description":"Returns a number to
     the user to guess higher or lower than.",
     "operationId":"getGameplayUsingGET",
     "responses":
      { "200":
            {"description":"OK","content":
            {"application/json":
            {"schema":
            {"$ref":"#/components/schemas/Status"}}}},
```

```
        "401":{"description":"Unauthorized"},
        "403":{"description":"Forbidden"},
        "404":{"description":"Not Found"}}}}
  }
}
```

The logical model is generated by parsing the associated OAS document for each application, should it be present. It takes the form of a series of beliefs based on the following formats:

```
hasGET(<string>, <string>);
hasPOST(<string>, <string>);
hasPUT(<string>, <string>);
hasPATCH(<string>, <string>);
hasDELETE(<string>, <string>);
hasNoOpenAPIDescription(<string>);
hasOpenAPIDescription(<string>);
```

The last two belief formats are used to clarify whether or not the the service has an OAS document. If it has one, then the `hasXXX(...)` beliefs are used to describe the service's endpoints. For these beliefs, the first 'string' parameter is the service's unique identifier and the second 'string' parameter is composed of the base IRI of the application, followed by the endpoint which is being described. This allows the agent to believe that the endpoint identified by a given IRI will accept a GET request.

In the context of the Web of Things, the 'shape' of a resource is described by the ThingDescription (TD) [1], which has attributes such as 'readProperty' and 'writeProperty', and is utilized widely in the Web of Things as an enabler of interoperability [10]. An OAS document can also be seen as being quite similar to the Web of Things TD. Tzavaras et. al, in [17] showcase a comparison of both OpenAPI and ThingDescriptions based on their capacity to describe Things, stating that there is a great deal of crossover between the two. Both OAS and TD give high-level static descriptions of the underlying APIs, where as Signifiers, as defined in [18] as first class abstractions in a hypermedia environment provide a more suitable solution for providing hypermedia descriptions as they can provide contextual hypermedia based on the state of the resource (see the HATEOAS principle as defined in [6]), but also the capabilities of the agent in question. This work was a move towards internalising work similar to Signifiers within the agent based on OAS documents in a Microservices environment in an attempt to bridge the gap between the current industry standard and the hypermedia rich environment we are striving for.

3.1 Experimental Setup

As a means of evaluating this approach, we propose a game of High/Low. This game will operate with the agent requesting a number and guessing whether the next number will be higher or lower than the received number. In order to achieve this we needed to not only facilitate interaction between software agents and microservices, but we also wanted to conform to the current standards of software engineering, as well as utilising components designed for and used by the microservices community. The goal of this experiment is to get an autonomous agent to participate in a game of *high-low* by identifying the correct microservice and interacting with that resource based on its ***shape*** that is identified by parsing the OAS documents of each service.

We created three applications and registered them with a service-discovery tool as *Application 1*, *Application 2* and *Application 3*. The purpose of this naming convention is to enforce the anonymity of the service that we are trying to allow the agent to discover and utilize. A layout of the system can be found in Figure 1. One of these applications is an implementation of a very simple, REST compliant, game of **High/Low**. By utilising CArtAgO [15] Artifacts to implement this tool, it remains agent programming language agnostic. This system is composed of three different agents, the Main Agent queries the service-discovery instance and creates an Application Agent that is instantiated with the IRI of each application registered. Once the Application Agent been created, this agent will then visit the base IRI of the application, at the */api-docs* endpoint to view the OAS document. Once the Application Agent has determined that the application has an OAS document, it begins to create a logical depiction of the resource it has been tasked with identifying. Should this application match the *shape* of the application it will create High/Low Agent to interact with the resource. The High/Low Agent has a logical depiction of how to play the High/Low game based on the "shape" of the service. Figure 1 describes the layout of the system. The code is available at the Git repo `https://gitlab.com/eoin.o-neill.3/longlivedwebopenapi` with instructions on how to run it.

4 Conclusion

In conclusion, by building this tool we have identified some of the pitfalls that exist with the current standard of service descriptions when exposing them to Web-enabled intelligent software agents and the tacit knowledge that is required when integrating microservices with one another. Section 2 has identified the integration issues that face the software engineering community when building systems of decoupled services that are required to interact in an ad-hoc manner. In order to facilitate agents being able to have a profound impact, the incorporation of Linked Data within service descriptions to define domain specific knowledge, while also providing explicit interaction definitions using vocabularies such as Hydra and Hypermedia Controls Ontology (HCTL), is paramount.

The work being done on signifiers as a first class abstraction [18] shows a path forward with regards to implementing these types of systems by providing enough context for agents at runtime to determine how to utilise a service and what the request and response requirements are in order to become the

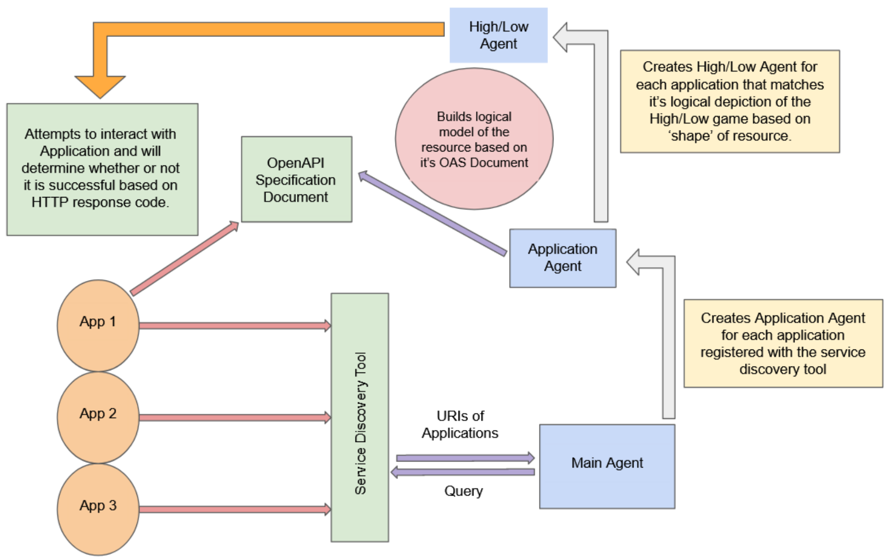

Fig. 1. System Layout

integrating bodies of these environments. As microservices architectures grow and systems begin to be made up of loosely-coupled, independently evolving services, the need for continuous, contextual integration that aligns with higher level goals provides an opportunity for hypermedia enabled agents to play a significant role. The goal of this research in the future is to implement this tool in a scenario where services are being developed and managed independent of each other with hypermedia representations providing enough contextual information for autonomous agents to act as the integrating intermediaries, a context such as large scale, smart agriculture, smart building or smart city scenario which provides enough variability and change to effectively test this tool at scale.

Acknowledgement : Eoin O'Neill and Rem W. Collier are supported by the Science Foundation of Ireland Strategic Partnerships Programme (16/SPP/3296) and is co-funded by Origin Enterprises Plc.

References

1. Web of things (wot) thing description, `https://www.w3.org/TR/wot-thing-description/`
2. Yahoo — mail, weather, search, politics, news, finance, sports amp; videos, `http://groups.yahoo.com/group/rest-discuss/message/3232`
3. Ciortea, A., Boissier, O., Zimmermann, A., Florea, A.M.: Responsive decentralized composition of service mashups for the internet of things. In: Proceedings of the 6th International Conference on the Internet of Things. pp. 53–61 (2016)

4. Espinoza-Arias, P., Garijo, D., Corcho, O.: Mapping the web ontology language to the openapi specification. In: International Conference on Conceptual Modeling. pp. 117–127. Springer (2020)
5. Evans, E., Evans, E.J.: Domain-driven design: tackling complexity in the heart of software. Addison-Wesley Professional (2004)
6. Fielding, R.T.: REST: Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation (2000), `http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm`
7. Fowler, M.: Domain driven design (Apr 2020), `https://www.martinfowler.com/bliki/DomainDrivenDesign.html`
8. Guo, Z., Cao, D., Tjong, D., Yang, J., Schlesinger, C., Polikarpova, N.: Type-directed program synthesis for restful apis. arXiv preprint arXiv:2203.16697 (2022)
9. Idehen, K.U.: Swagger, the api economy, rest, linked data, and a semantic web (Aug 2018), `https://medium.com/openlink-software-blog/swagger-the-api-economy-rest-linked-data-and-a-semantic-web-9d6839dae65a`, accessed=07/04/2022
10. Kaebisch, S., Anicic, D.: Thing description as enabler of semantic interoperability on the web of things. In: Proc. IoT Semantic Interoperability Workshop. pp. 1–3 (2016)
11. Lanthaler, M., Gütl, C.: Hydra: A vocabulary for hypermedia-driven web apis. LDOW **996**, 35–38 (2013)
12. Muhamad, W., Bandung, Y., et al.: Transforming openapi specification 3.0 documents into rdf-based semantic web services. Journal of Big Data **9**(1), 1–24 (2022)
13. O'Neill, E., Lillis, D., O'Hare, G., W Collier, R.: Delivering multi-agent microservices using cartago. In: International Workshop on Engineering Multi-Agent Systems. pp. 1–20. Springer (2020)
14. O'Neill, E., Lillis, D., O'Hare, G.M., Collier, R.W.: Explicit modelling of resources for multi-agent microservices using the cartago framework. In: Proceedings of the 18th International Joint Conference on Autonomous Agents and Multi-Agent Systems, Auckland, NZ, 2020. International Foundation for Autonomous Agents and MultiAgent Systems (IFAAMAS) (2020)
15. Ricci, A., Viroli, M., Omicini, A.: Cartago: A framework for prototyping artifact-based environments in mas. In: International Workshop on Environments for Multi-Agent Systems. pp. 67–86. Springer (2006)
16. SILVA, S.: REST Service Discovery Based on Ontology Model. Ph.D. thesis (2021)
17. Tzavaras, A., Mainas, N., Petrakis, E.G.: Openapi framework for the web of things. Internet of Things **21**, 100675 (2023)
18. Vachtsevanou, D., Ciortea, A., Mayer, S., Lemée, J.: Signifiers as a first-class abstraction in hypermedia multi-agent systems. arXiv preprint arXiv:2302.06970 (2023)
19. W Collier, R., O'Neill, E., Lillis, D., O'Hare, G.: Mams: Multi-agent microservices. In: Companion Proceedings of The 2019 World Wide Web Conference. pp. 655–662. ACM (2019)
20. Yang, J., Wittern, E., Ying, A.T., Dolby, J., Tan, L.: Towards extracting web api specifications from documentation. In: 2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR). pp. 454–464. IEEE (2018)