Towards Concern-Oriented Microservice Architecture

Hugo Monfleur, Philippe Merle

Towards Concern-Oriented Microservice Architecture

Hugo Monfleur · Philippe Merle

Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

✉ hugo.monfleur@inria.fr

Abstract. Existing microservice architecture languages do not allow for the distinction and expression of intertwined cross-cutting architectural concerns, i.e. aspects that affect several microservices, without the possibility of being encapsulated in any of them. To address this problem, this paper proposes the Concern-Oriented Microservice Architecture metamodel where cross-cutting architectural concerns are first-class citizens and show how the worldwide-used Docker Compose language can be equipped with new concern-oriented constructions. This extension is applied to the TeaStore microservice reference application.

1 Introduction

Any microservice application has to deal with cross-cutting architectural concerns, i.e. aspects that affect several microservices, without the possibility of being encapsulated in any of them. A particular concern can then be defined formally *as a predicate on units that indicates whether or not a given unit pertains to that concern*[1]. The problem lies in the fact that these concerns often cannot be cleanly decomposed from the rest of the system in both the design and implementation, and can result in either scattering (code duplication), tangling (significant dependencies), or both [7]. Separation of concerns, probably coined by Edsger W. Dijkstra in [1] and consisting in the study *of one's subject matter in isolation for the sake of its own consistency*, is then vital for concision, comprehension, maintainability, and evolvability of microservice architectures.

To help get a better grip of what a concern is and how diverse they can be in the context of microservice applications, we introduce three examples. Firstly, each application is composed of microservices. Some are dedicated to implementing business logic while others implement non-functional properties such as logging, monitoring, distributed tracing, persistence, security, or fault-tolerance. All of these relate to high level architecture decisions that determine the value of properties of microservices in regard to a particular aspect. As such they are cross-cutting architectural concerns. Secondly, various design patterns have been identified in the literature (e.g. [5]) to architect business and non-functional properties such as Service Registry, API Gateway, Circuit Breaker, or Distributed Tracing. Here again, these patterns are cross-cutting architectural concerns. Finally, each application has its own patterns for naming microservice identities, images, environment variables, etc. To the best of our knowledge, there is no microservice architecture language offering constructions for dealing with such cross-cutting architectural concerns.

[1] https://web.archive.org/web/20080122043747/http://www.research.ibm.com/hyperspace/ConcernSpaces.htm

This paper argues for making all intertwined cross-cutting architectural concerns first-class citizens of microservice architecture languages. This implies equipping microservice architecture languages with new concern-oriented constructions. In order to be language-agnostic, we propose a Concern-Oriented Microservice Architecture metamodel where cross-cutting architectural concerns are first-class citizens. In order to be concrete, we choose to extend the Docker Compose language as it is largely used in both academia and industry. A recent search on GitHub[2] returns more than 672k Docker compose files. Then we apply this concern-oriented extension of Docker Compose to TeaStore, a micro-service reference application[3] [8]. We chose this use case for validating our proposal as it has been quoted in over 100 scientific publications[4] and is also used in industry, e.g. by the Cisco Full Stack Observability Lab[5].

The rest of the paper is organized as follows. Section 2 presents our motivating example, the TeaStore application, and emphasizes its multiple scattered cross-cutting architectural concerns. Section 3 overviews the principles of our Concern-Oriented Microservice Architecture contribution, then revisits its TeaStore for making its concerns first-class citizens of Docker Compose, and sketches a toolchain for dealing with concern-oriented microservice architectures. Section 4 reviews some related work. Finally, Section 5 concludes and discusses future perspectives.

2 Motivating Example

TeaStore is a *micro-service reference application for benchmarking, modeling, and resource management research* [8]. This application, developed as a real-life application with best practices in mind, is *an online shop for tea and tea-related utilities* [8]. TeaStore is composed of seven microservices: six are represented in Figure 1, and the seventh is a database dedicated to the Persistence microservice.

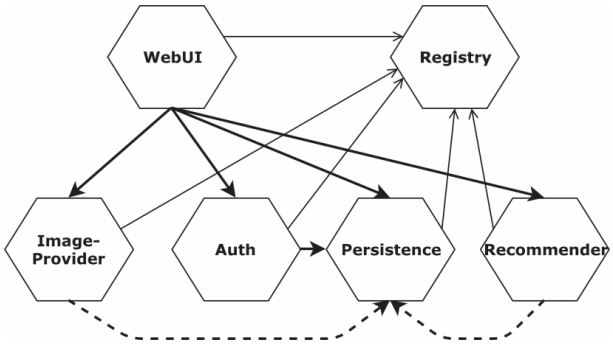


Figure 1: Architecture of the TeaStore application [8]

Listing 1 is the Docker Compose file of the default architecture of the TeaStore application[6].

```
1 version: '3'
2 services:
3   registry:
4     image: descartesresearch/teastore-
      registry
5     expose:
```

[2] https://github.com/search?q=path%3A**%2Fdocker-compose*.y*ml&type=code
[3] https://github.com/DescartesResearch/TeaStore
[4] https://scholar.google.fr/scholar?cites=7828615202992379775
[5] https://fso.cisco-one.com/lab-environment/
[6] https://raw.githubusercontent.com/DescartesResearch/TeaStore/master/examples/docker/docker-compose_default.yaml

```
6       - "8080"
7   db:
8     image: descartesresearch/teastore-db
9     expose:
10      - "3306"
11    ports:
12      - "3306:3306"
13  persistence:
14    image: descartesresearch/teastore-
      persistence
15    expose:
16      - "8080"
17    environment:
18      HOST_NAME: "persistence"
19      REGISTRY_HOST: "registry"
20      DB_HOST: "db"
21      DB_PORT: "3306"
22  auth:
23    image: descartesresearch/teastore-auth
24    expose:
25      - "8080"
26    environment:
27      HOST_NAME: "auth"
28      REGISTRY_HOST: "registry"
29  image:
30    image: descartesresearch/teastore-image
31    expose:
32      - "8080"
33    environment:
34      HOST_NAME: "image"
35      REGISTRY_HOST: "registry"
36  recommender:
37    image: descartesresearch/teastore-
      recommender
38    expose:
39      - "8080"
40    environment:
41      HOST_NAME: "recommender"
42      REGISTRY_HOST: "registry"
43  webui:
44    image: descartesresearch/teastore-webui
45    expose:
46      - "8080"
47    environment:
48      HOST_NAME: "webui"
49      REGISTRY_HOST: "registry"
50    ports:
51      - "8080:8080"
```

Listing 1: Default Docker Compose of the TeaStore Application

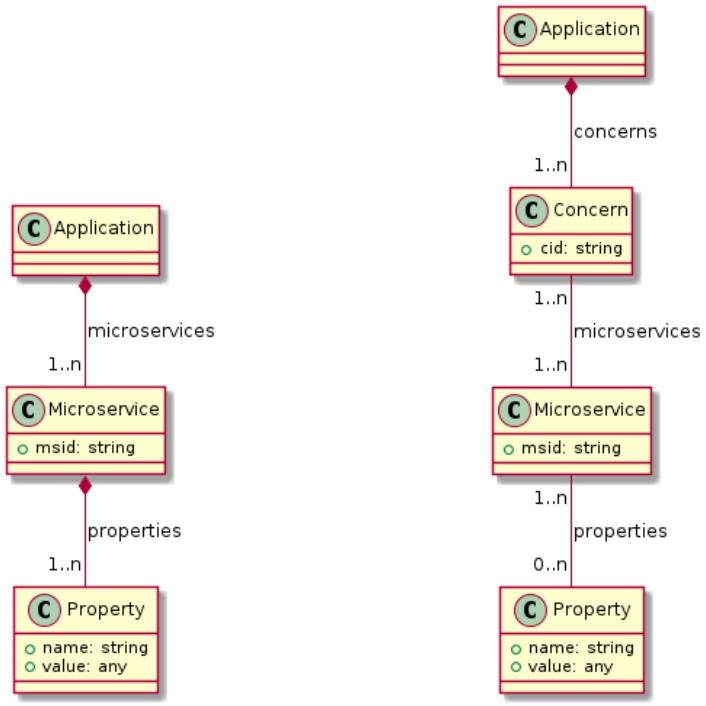
Even if this motivating example seems simple, it contains seven intertwined cross-cutting architectural concerns. The first concern is related to the identity (e.g. `registry`) of the six **business microservices** shown in Figure 1, and is scattered across lines 3, 13, 22, 29, 36, and 43. The second concern is related to the **Persistence composite microservice**, which is split into both strongly coupled `persistence` and `db` microservices (Lines 7-21). The third concern is the **Service Registry design pattern**, scattered across lines 3, 19, 28, 35, 42, and 49. `persistence`, `auth`, `image`, `recommender` and `webui` microservices can discover each others via the `registry` microservice. The fourth concern is related to the **image naming pattern**, scattered across lines 4, 8, 14, 23, 30, 37, and 44. We can notice that all seven image names share the same prefix `descartesresearch/teastore-` and their suffix is always the identity of their owning microservice. The fifth concern is the **host name regularity** scattered across lines 18, 27, 34, 41, and 48. The `HOST_NAME` environment variable is always assigned to the identity of its owning microservice. The sixth concern is the **exposed container network port regularity** scattered across lines 5-6, 15-16, 24-25, 31-32, 38-39, and 45-46. The six business services expose the same container network port, aka `8080`. Finally, the seventh concern is related to **public network ports** exposed by the TeaStore application, and is scattered across lines 11-12 and 50-51. Only ports `3306` of `db` and `8080` of `webui` are publicly accessible.

3 Concern-Oriented Microservice Architecture

3.1 Principles

As illustrated in Figure 2a, in a classical Microservice Architecture (MSA) approach, each **Application** owns a none-empty set of `microservices`. Each **Microservice** has a unique distinct identity `msid` (e.g.

registry) and owns a non-empty set of properties. Each **Property** has a unique distinct name (e.g. image) and a value (e.g. descartesresearch/teastore-registry).



(a) MSA Metamodel (b) COMSA Metamodel

Figure 2: Microservice Architecture vs Concern-Oriented Microservice Architecture

As illustrated in Figure 2b, the metamodel of our Concern-Oriented Microservice Architecture (COMSA) approach is subtly different. Each **Application** owns a non-empty set of concerns, making concerns first-class citizens. Each **Concern** has a unique distinct identity cid and references a non-empty set of microservices. A microservice can be shared by several concerns. Each **Microservice** still has the msid attribute but references a set of properties instead of owning its properties. Thus a property can be shared by several microservices. Finally, **Property** is unchanged.

The abstract concepts of our COMSA approach must be concretely implemented in target existing microservice architecture languages. The next section illustrates how the COMSA concepts can be reflected by new concern-oriented constructions in the Docker Compose language.

3.2 Motivating Example Revisited

Listing 2 revisits the motivating TeaStore example by making its seven concerns first-class citizens thanks to the new concerns construct in Line 2.

```
1 version: '3'
2 concerns:
3   BusinessServices:
4     services:
5       (registry, persistence, auth, image, recommender, webui):
6   DescartesResearchImages:
7     services:
```

```
8       ALL:
9         image: descartesresearch/teastore-{SID}
10    RegistryDesignPattern:
11      services:
12        BusinessServices:
13          expose:
14            - "8080"
15        BusinessServices \ registry:
16          environment:
17            HOST_NAME: "{SID}"
18            REGISTRY_HOST: "registry"
19    Persistence:
20      services:
21        db:
22          expose:
23            - "3306"
24        persistence:
25          environment:
26            DB_HOST: db
27            DB_PORT: "{db.expose[0]}"
28    Ports:
29      services:
30        (db, webui):
31          ports:
32            - "{expose[0]}:{expose[0]}"
```

Listing 2: Concern-Oriented TeaStore Applications

The first concern is encoded by `BusinessServices` at Lines 3-5. Line 5 shows another new construct to define sets of microservices. `DescartesResearchImages` at Lines 6-9 encodes the fourth concern previously identified. Lines 8-9 encode the fact that all microservices of the TeaStore application (`ALL` keyname) share a common image naming pattern. The `{SID}` expression refers to the microservice identity. `RegistryDesignPattern` at Lines 10-18 mainly implements the third concern at Line 18 but also the fifth and sixth concerns, Line 17 and Lines 12-14 respectively. In fact, both regularities are strongly coupled with the service registry design pattern. Line 12 shows how to refer to all the microservices of another concern (here `BusinessServices`) and Line 15 illustrates the difference operation between sets: `BusinessServices \ registry` equals to `(persistence, auth, image, recommender, webui)`. The second concern is implemented by `Persistence` at Lines 19-27. The expression `{db.expose[0]}` accesses the first item of the `expose` array property of the `db` microservice. Finally, the seventh concern is encoded by `Ports` at Lines 28-32. Both `{expose[0]}` expressions access the first item of the `expose` property of the current microservice (`db` then `webui`).

To summarize, our concern-oriented extension of the Docker Compose language provides:

- the new `concerns` keyword for making concerns first-class citizens of Docker Compose,
- a new notation to manipulate sets of microservices, and
- a powerful notion of expressions to avoid redundancy of property values.

This extension provides multiple benefits such as clear separation of microservice concerns for better comprehension, concision (32 lines instead of 51 lines, i.e. a gain of 37% for the TeaStore application), ease of maintainability, and evolvability.

3.3 COMSA Toolchain

Figure 3 shows the current architecture of our concern-oriented microservice architecture toolchain.

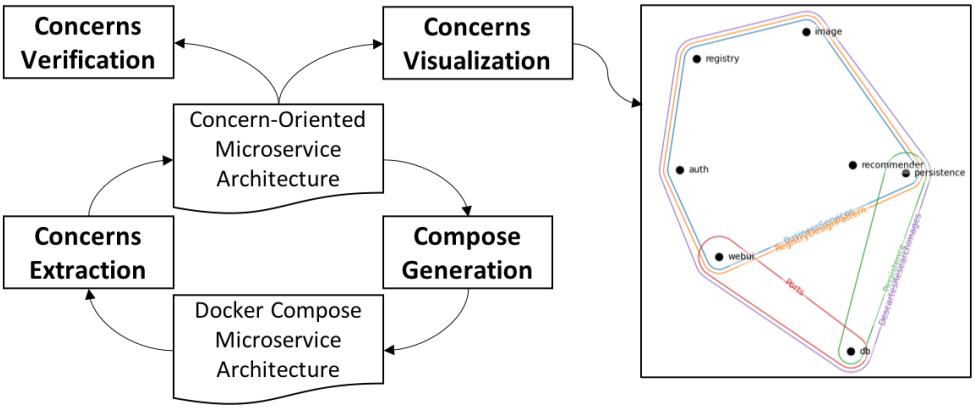


Figure 3: Concern-Oriented Microservice Architecture Toolchain

Concerns verification targets the verification and validation of COMSA files, i.e. syntactical and semantics checking of `concern` declarations, microservice sets, and property value expressions. **Concerns Visualization** generates visual concern-oriented diagrams. Hypergraphs are particularly well suited to visualize concern-based applications as illustrated in the right part of Figure 3, i.e. a hypergraph representing the concerns of the TeaStore application. **Compose Generation** generates standard Docker Compose files from COMSA files, i.e. compose the concerns expressed in Listing 2 to generate Listing 1 automatically. **Concerns Extraction** does the reverse job of extracting concerns from any existing Docker Compose files. Here the challenge is to find the right heuristics to automatically build pertinent concerns.

4 Related Work

To the best of our knowledge, no MSA language provides concern-oriented constructions even though some MSA languages deal with specific concerns. LEMMA [4] is a modeling MSA language ecosystem including four viewpoints (Domain, Operation, Service, Technology), each addressing a specific set of stakeholders' concerns. Silvera [6] is a Domain-Specific Language for modeling MSA including constructions to deal with only five MSA design patterns, i.e. Service Registry, API Gateway, RPC and Messaging Communication Styles, and Circuit Breaker. MicroART [3] provides a MSA recovery metamodel including an attribute to qualify the nature of each microservice, i.e. functional (what we name business), monitoring, system-level management, service orchestration, service brokering, security, service proxy, and data storage. All these can be easily represented by our MSA concerns.

5 Conclusion and Perspectives

The key idea of this paper is to make any cross-cutting architectural concern a first-class citizen of MSA languages. For this, the paper introduces the COMSA approach, its metamodel and its toolchain, proposes a concern-oriented extension of the Docker Compose language, and applies it to the well-known TeaStore microservice reference application [8].

COMSA work is still in its infancy. Perspectives encompass enhancing COMSA principles, metamodel and toolchain, applying COMSA to various large-scale MSA applications such as those of the DeathStarBench benchmark suite [2], developing a library of reusable cross-cutting architectural concerns, and applying COMSA to other existing MSA languages.

Acknowledgments

This work was partially funded by the French ANR SCALER project.

References

[1] Edsger W Dijkstra. On the role of scientific thought. *Selected writings on computing: a personal perspective*, pages 60–66, 1982.

[2] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, et al. An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 3–18, 2019.

[3] Giona Granchelli, Mario Cardarelli, Paolo Di Francesco, Ivano Malavolta, Ludovico Iovino, and Amleto Di Salle. MicroART: A Software Architecture Recovery Tool for Maintaining Microservice-Based Systems. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pages 298–302, 2017.

[4] Florian Rademacher. *A Language Ecosystem for Modeling Microservice Architecture*. PhD thesis, Kassel, Universität Kassel, Fachbereich Elektrotechnik / Informatik, 2022.

[5] Chris Richardson. *Microservices patterns: with examples in Java*. Simon and Schuster, 2018.

[6] Alen Suljkanović, Branko Milosavljević, Vladimir Inđić, and Igor Dejanović. Developing Microservice-Based Applications Using the Silvera Domain-Specific Language. *Applied Sciences*, 12(13):6679, 2022.

[7] Peri Tarr, Harold Ossher, William Harrison, and Stanley M. Sutton. N degrees of separation: Multi-dimensional separation of concerns. In *Proceedings of the 21st International Conference on Software Engineering*, ICSE '99, pages 107–119, New York, NY, USA, 1999. Association for Computing Machinery.

[8] Jóakim von Kistowski, Simon Eismann, Norbert Schmitt, André Bauer, Johannes Grohmann, and Samuel Kounev. TeaStore: A Micro-Service Reference Application for Benchmarking, Modeling and Resource Management Research. In *Proceedings of 2018 IEEE 26th International Symposium on the Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, MASCOTS'18, pages 223–236, September 2018.