Airtight NFR and Cross-Cutting Concerns through a MECE Pipeline Taxonomy for a Highly Reactive Microservice-Based Architecture

Eric Chartré[1,*], Mehdi Adda[1]
[1] Université du Québec à Rimouski
*Correspondence: eric.chartre@uqar.ca, eric@chartre.net

Extended Abstract

Non-functional requirements (NFR), protocol fitness, aspects[†] and cross-cutting concerns in a microservice-based architecture are critical issues that cannot be ignored. This paper proposes a MECE[‡] pipeline taxonomy that can help to address these issues as well as introducing a standardized implementation of cross-cutting concerns. This execution pipeline taxonomy will help software developers design highly reactive and reproducible application architectures through a series of addressable filters[§] within a reusable, configurable and agnostic[**] framework.

This pipeline taxonomy organizes business informational resources and nullipotent filters in a strict predetermined sequential order. This order cannot be changed but filters can be activated or deactivated according to the needed requirements. This taxonomy defines which component can call another component, as a stage director would do in a choreography.

Amongst the NFR that are covered within the taxonomy, we find NFR regarding:

- security (access control, policy control, authentication, logging and monitoring, encryption, message integrity)
- auditability
- user experience and accessibility
- quality (error and exception handling)
- flexibility (e.g., protocol handling), reusability, modularity and extensibility
- elasticity and reactivity
- testability, robustness and recovery
- downward/upward compatibility

---

* As in Aspect-Oriented Programming
† As in Aspect-Oriented Programming
‡ MECE: Mutually Exclusive and Collectively Exhaustive
§ Pipes and filter design pattern
** Independent of the application stack and of the developer mindset or programming style

Figure 1 presents the proposed pipeline.

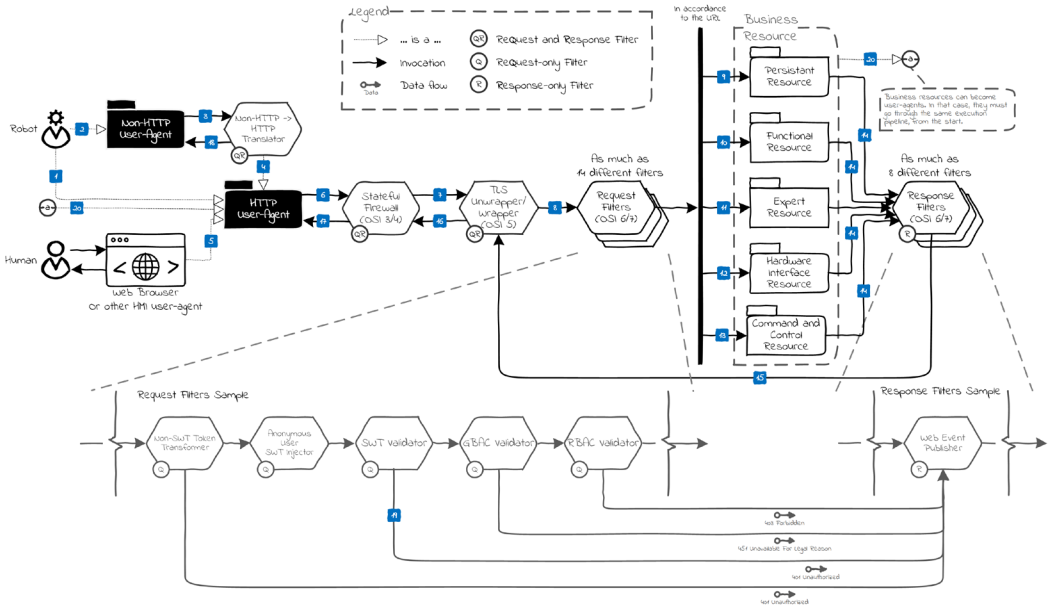

Figure 1: Execution Pipeline Taxonomy

Wherein:

A robot can be an HTTP agent [1] or a non-HTTP agent [2]. If the robot is a non-HTTP agent, a first filter oversees the translation from a non-HTTP message to a well-formed HTTP message [3]. For example, CoAP (Constrained Application Protocol for IoT), SMTP, SOAP, MQTT or Protobuf are non-HTTP protocols that can be converted easily to HTTP with their specific Translators.

Thereafter, the robot "speaks" in the pipeline's protocol: HTTP [4]. The robot could also speak natively in HTTP [1] as if it was a Web browser or any other HTTP human-machine interface [5].

The HTTP request must therefore go through a set of gates, valves, transformers, and so on, in the same fashion as Unix Pipes and Filters.

Filters cover OSI layers from 3 to 7. A stateful firewall [6] starts with the OSI layers 3 (network) and 4 (transport). The next filter is the TLS Unwrapper (OSI layer 5, session) that is activated if the packet was encrypted [7].

Fourteen (14) functional request filters have been identified so far [8] to cover OSI layers 6 (presentation) and 7 (application). Filters could use a cache, a directory or a configuration store specific to the filter, or global to the whole system. According to the URL and the HTTP method, filters can be activated or not. Beside the earlier filters, the pipeline includes Web application firewalls, URN Translator, Traffic Meter, Metadata Injectors, SWT and non-SWT tokens Validators, GBAC (geography), RBAC (role), ABAC (attribute) LBAC (license) Access Control Validators, Application Cache that accelerates the response, without having to go through processing, Circuit Breakers, and so on.

Following the request filters, the request is sent to the business informational resource described in the URL. This resource can be one and only one of these kinds:

- an idempotent Persistent Resource [9],
- a nullipotent Functional (as in a mathematical function) Resource [10]
- a nullipotent extended functional resource: Expert Resource [11] (oracle, investigator, aggregator, simulator),

- an idempotent Material Interface Resource [12] (replicators, transporters, communicators, matter consumers, matter transformers, exhibitors),
- a nullipotent Command and Control Resource [13] that will control the transactional workflow (UI or mechanized).

The business resource process result (HTTP response or HTTP error response) has the potential to go through nine (9)[††] new response filters: Application Cache, HTML Portal Wrapper, Event Registrar (message queues, pub/sub, CQRS), Web Event Publisher, W3C Logger, Header Cleaner, HTTP/2 Rewriter, Exception and Error Handler. [14]

As for the request, the HTTP response must be well formed.

The HTTP response goes back to the TLS Wrapper [15] to continue its course to the stateful firewall [16] all the way to the HTTP user-agent [17]. If the initial user-agent was a non-HTTP agent, the response is retranslated in the right protocol [18].

All filters are independent. But they are executed in a strict order so it can make sense without creating security holes or making cross-cutting concerns step over each other's responsibilities. For example, if an anonymous user accesses a business resource and the SWT Validator is activated, the SWT Injector must add a SWT token into the HTTP header. Without it, the SWT Validator won't let the request go to the next request filter, generating an exception (`401 Unauthorized`) and bypassing the business resource completely [19]. As another example, access control must be executed in the following order GBAC → RBAC → ABAC → LBAC to avoid security problems.

In that respect, if a component generates an error, the pipeline can be partially short-circuited.

Finally [20], a business resource can become user-agents by calling another business resource. In that case, that business resource must absolutely go through the same execution pipeline without any exception or shortcuts. This ensures NFR and cross-cutting concerns airtightness if filters are configured properly, by the least privilege principle.

By mandatorily using this execution pipeline on all microservices internal and external calls, it allows developers to focus on the design of business informational resources without being preoccupied by the implementation of the pipeline's cross-cutting concerns within the business microservices. Furthermore, if filters are configured to be mandatory by default, it will add constraints to the system design that are not easily circumventable; if they are deflected, it will be by thoughtful actions on the constraints. This standardization will also obviate and mitigate avoidable risks regarding NFR.

This pipeline taxonomy is part of a new psychotechnology and metamodel for designing highly reactive microservice-based architectures[‡‡]; the other parts of the psychotechnology being a set of architectural design patterns, a hierarchical informational resource taxonomy and a simple notation to express that architecture efficiently.

References

Ambler, S. W. (2013). Technical (Non-Functional) Requirements: An Agile Introduction. Retrieved from: http://agilemodeling.com/artifacts/technicalRequirement.htm

Bass, L., Clements, P., & Kazman, R. (2012). Software Architecture in Practice (Third Edition). Addison-Wesley Professional.

Bonér, J., Farley, D., Kuhn, R., & Thompson, M. (2014, September 16). The Reactive Manifesto v2.0. Retrieved from: https://www.reactivemanifesto.org/

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). Pattern-Oriented Software Architecture Volume 1: A System of Patterns (Volume 1 edition). Wiley.

Chartré, E. (2021, November 11). Psychotechnologie en conception logicielle de systèmes de gestion de l'information réactifs et distribués, conception homogène et reproductible basée sur des microservices [Presentation]. Conférence IEEE Québec Chapitre Ordinateur.

[††] Number of identified response filters so far.

[‡‡] Chartré, E. (2022) Highly reactive microservice-based architecture psychotechnology. Master's Thesis. To be published.

Chartré, E. (2022). Psychotechnologie de découpage en microservices réactifs [To be published]. Master's Thesis. Université du Québec à Rimouski.

Chong, J., Faria, A., Nadathur, S., Yi, Y. (n.d.) *Pipe and Filter | Our Pattern Language*. Retrieved from https://patterns.eecs.berkeley.edu/?page_id=19

Dalbey, J. (2002). Nonfunctional Requirements. Retrieved from: http://users.csc.calpoly.edu/~jdalbey/SWE/QA/nonfunctional.html

Davies, R. (2010). Non-Functional Requirements: Do User Stories Really Help? Retrieved from: https://www.methodsandtools.com/archive/archive.php?id=113

Microsoft. (n.d.). *Pipes and Filters pattern—Azure Architecture Center*. Retrieved from https://docs.microsoft.com/en-us/azure/architecture/patterns/pipes-and-filters

Enterprise Integration Patterns—Pipes and Filters. (n.d.). Retrieved from https://www.enterpriseintegrationpatterns.com/PipesAndFilters.html

Harrison, N. B., & Cockburn, A. (2007). Learning the lessons of architecture patterns. Journal of Computing Sciences in Colleges, 23(1), 198–203.

International Organization for Standardization. (2014). ISO 25010. Retrieved from: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

International Organization for Standardization. (2020, October 29). ISO 27001 Security Requirements of Information Systems. ISO 27001 Guide. Retrieved from: https://iso27001guide.com/iso-27001-security-requirements-of-information-systems-iso27001-guide-iso27001-guide.html

Nadareishvili, I., Mitra, R., McLarty, M., & Amundsen, M. (2016). *Microservice Architecture: Aligning Principles, Practices, and Culture* (1st edition). O'Reilly Media.

Newman, S. (2015). *Building Microservices: Designing Fine-Grained Systems* (1st edition). O'Reilly Media.

Wikipedia (n.d.) Non-functional requirement. Retrieved from Wikipedia: https://en.wikipedia.org/w/index.php?title=Non-functional_requirement&oldid=1059099044

Whitfield, M. (2019, November 14). Non-functional requirements – NFRs – About. Mark Whitfield. Retrieved from: https://mark-whitfield.com/about/non-functional-requirements-nfrs-about/