HAL
open science

# “Functional-first” recommendations for beneficial microservices migration and integration. Lessons learned from an industrial experience

Jean-Philippe Gouigoux, Dalila Tamzalit

## To cite this version:

Jean-Philippe Gouigoux, Dalila Tamzalit. “Functional-first” recommendations for beneficial microservices migration and integration. Lessons learned from an industrial experience. IEEE INTERNATIONAL CONFERENCE ON SOFTWARE ARCHITECTURE (ICSA 2019), Mar 2019, Hamburg, Germany. 10.1109/icsa-c.2019.00040 . hal-02083127

**HAL Id: hal-02083127**

**https://hal.science/hal-02083127v1**

Submitted on 8 Feb 2024

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# “Functional-first” recommendations for beneficial microservices migration and integration

## Lessons learned from an industrial experience

Jean-Philippe GOUIGOUX
Chief Technical Officer
R&D department
MGDIS SA
Vannes, France
gouigoux-jp@mgdis.fr

Dalila TAMZALIT
LS2N – CNRS UMR 6004
Université de Nantes
Nantes, France
Dalila.Tamzalit@univ-nantes.fr

***Abstract.*** **MGDIS is a French software development company which successfully migrated her monolith application towards a microservices architecture. This migration was due to a major strategic and technical change and needed an investment of 17 300 person.days over three years. While some results were initially targeted, others were not expected at all. However, with hindsight following an afterward in-depth analysis, they eventually proved to be unexpected best-practices that should be adopted for successful microservices adoption. These lessons learned are outlined following four crucial aspects: functional approach, norms and standards, microservices granularity and their semantics, and finally technical and integration outcomes.**

***Keywords—microservices; migration; Web Oriented Architecture; business alignment; reuse by integration; business experience.***

## I. INTRODUCTION AND CONTEXT

MGDIS SA[1] is a French software development and vendor applications that target public collectivities, helping them in managing lifecycle and payments of financial subsidies and scholar grants. In this business sector, a complete Information System is typically installed on premise mainly, composed with almost two hundred different applications and used in total by around one thousand agents. The main characteristics of this market are a large importance of systems interoperability and a complete product lifecycle generally around ten years.

Due to the upcoming obsolescence of the existing applications, MGDIS board of directors decided in 2013 a complete rewrite from scratch using a modern web-based architecture, following the World Wide Web Consortium (W3C) recommendations [3]. MGDIS main goal at that time was to reduce the cost of interoperability, allowing long term improvement of the system by collaborating or embedding external software capabilities. The main strategy used to fulfill these goals was from the beginning based on Information System Alignment[2] [4, 5, 6, 7]. Namely, obtained components need to be as autonomous and decoupled as possible, so as to ease development and avoid eventual clutter from technical debt.

In 2015, the first obtained version of the new application has been tested by one pilot customer, namely a regional council with a few hundreds of agents and a few tens of thousands of external users of the associated web site. This customer operated the new version of the software and served as a beta-tester for some of the features during one year. In 2016, the application was stable enough to be deployed to other customers, some of them on premise and some of them using the same software operated directly by MGDIS as a Software as a Service [8]. These three years of development as well as eighteen months in operation provided some feedback on this software rewrite, among which the most important findings are presented in the following.

Authors’ first paper [30] gave a first round of feedback, mainly focused on lessons learned about technical issues (service granularity, deployment and orchestration). The present paper attempts at generalizing on a global best practice, lessons learnt by MGDIS from this migration process. The paper is organized as follows: after the presentation in this first section of the industrial context and objectives, the paper presents a successful approach to obtain a long-living software. Section II outlines on the importance of following a functional rather than technical segmenting. Section III presents the importance of using norms and standards. Section IV focuses on the best ways for slicing correctly microservices while Section V shows the importance of business semantics of the identified microservices. Section VI mentions the importance of the technical side during migration while Section VII presents the benefits of the software in its new shape, on the integration and finally Section VII concludes this paper by opening a couple of future works.

## II. ON THE IMPORTANCE OF FOLLOWING A FUNCTIONAL SEGMENTING

Like any other software company deciding to turn a monolithic application into a group of autonomous services interacting with each other, MGDIS had to address several well-known issues, among which the first and main one is choosing **how to split the former monolith into granular APIs**. Knowing the technical bits does not help in any way in knowing *where* to cut.

The main lesson learned in the past three years, and highlighted in this paper, re-architecting the software application is that the functional approach must lead the technical approach. However, most of the schemas templates in software architecture are technically-oriented. Using the

[1] http://www.mgdis.fr

[2] In France, the term “urbanization” is also widely used.

four-layer representation from the CIGREF[3]'s white paper [31] considerably helped to the success of the project by offering an appropriate way to associate the functional with the technical.

This four-level diagram, represented in Figure 1, has long been used in relation with Business / IT alignment and Information System cartography. It helped a lot in finding the right decomposition of a monolith software into modular services.

**An essential return on experience from the industrial migration is that the cutting planes should not come from the layer 3 (technical) but from the layer 2 (functional), since this embodies all business functions that are then technically implemented at the level 3.**

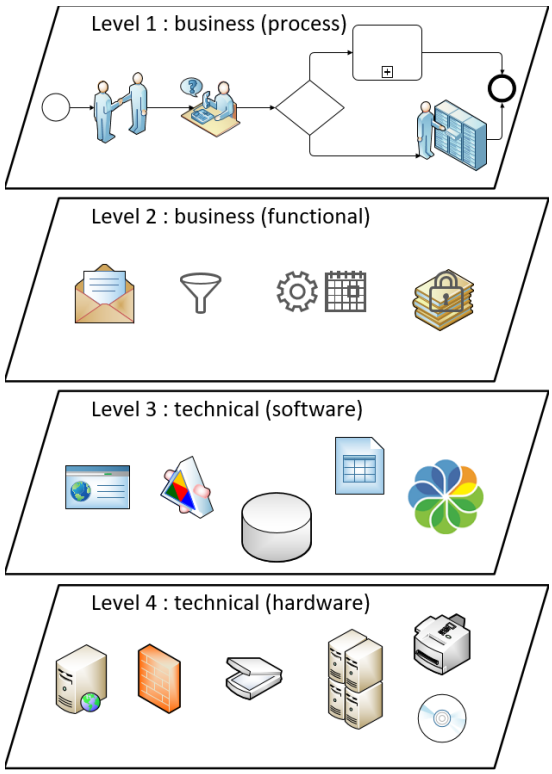


Figure 1: the four-level IT map diagram

As a software editor, the use of the four-level urbanization diagram progressively became the central point of the whole software production. It was part of every developers' objectives to reflect every step of the production info all four level of this diagram. Every single person in a feature team was asked to be able to relate his work to the process, the functionality, the software and the hardware. For example, a developer would always be able to explain how the code he or she crafted was related to which function, used by which customer process, and which infrastructure it would be installed on.

When considering technical outcomes of the three-year migration, **it turned out that most of them also derived from the functional approach**. The following will describe the most significant outcomes in further details.

## III. Going Standard / Normalization Approach

Using a functional rather than technical point of view to carve services out of the existing application led MGDIS to massive use of norms and standards where they existed and Domain Driven Design [18] to establish proper and internal pivotal format and canonical entities where no standards existed. These norms, standards, pivotal format and canonical entities are used for the definition of microservices contracts. Their use to define the architecture turned out to be the major change in terms of impact on the resulting application.

Using standards is a well-accepted requirement in enterprise architecture, but not always respected. Microservices dedicated to supporting one particular business standard make for a good alignment between IT and processes [32]. **They are in fact the glue between the functional and the technical layers in the four-layers diagram**.

MGDIS' approach was that everything related to business had to be standards-driven. When a norm existed, its respect was made mandatory. When no norm existed, a canonical entity was laid out and pivotal formats open sourced so as to help standardization. The final application uses tens of business norms (vCard, OAuth2, SAML, SCIM, XACML, GeoJSON, CMIS, ODPv2 to name just a few) alongside with hundreds of technical norms (RFC, IETF, ISO, etc.), the latter being current practice but the former definitely not so. The fact that Swagger turns into Open API Initiative is also a move towards better normalization. Interestingly, norms and standards generally come from consortium making experts in the business domain agree on a precise (thus technical) representation of their field, and one can mention this initiative of *IT Standard for Business*[4].

## IV. On Microservice Granularity

The use of the prefix "micro" tends to express the fact that actual size of the service itself is of importance. Yet, opinions are multiple about what the size of a microservice should be [13, 14, 15] and there is not even an agreement on the unit that should be used (Lines Of Code, weight of the library, number of methods in the API, etc.). The field is also blurred with the appearing of nanoservices and lambda [16] / serverless [17] architectures.

The main recommendation of this article is to give up those technical considerations and base the microservices slicing on functional ones, just like for any correctly-realized componentization. Instead of using notions from level 3, this means using semantics from level 2 of the IT alignment diagram (see figure 1).

## V. Importance of Semantics

A surprising outcome of this three-year-long experience aligning all the production to standards was that semantics turned out to be a major challenge. Since every microservices had to be easily replaceable and API contracts where, as a consequence, of major importance, business semantics

[3] The Cigref, an association created in 1970, is a network of major French companies and public administrations whose mission is to make possible digital success.

[4] https://www.itforbusiness.org/

gradually took a more and more important place. This is closely related to the DDD approach of ubiquitous language, that has to reconcile business and technical users. Semantics is the basis for the second layer of the diagram.

At some point, one person among the thirty developers was dedicated to creating pivotal formats and canonical entities, with associated standardized vocabularies and definition. This allowed for precise shared understanding of the business stakes. Reasoning in terms of semantics and choosing the exact right word for every situation helped in solving more and more complex situations by the means of software. Expressions, and particularly business jargon, are often overly simplified wordings of a more complex situation, and getting better at semantics helped a lot in finding the right model for a business concept. For example, one easily assumes that an adjective on a noun designating a concept translates into an attribute of the associated entity in a model. In a fair number of situations, the attribute is in fact borne by the relationship between the entity and some other entity given by the context. To give an extremely simplified example, it is said that a person can be in credit, but adding a Boolean attribute to store this quality actually does not make sense, since a person can only be in credit towards a given other person or entity, and can very well be in debt at the same time towards others. Semantics has been such a strong move at MGDIS that modeling of its own internal process has been realized using the four-level diagram and a semantics document has been issued to align every employee with the right names to describe our commercial and technical content.

## VI. Benefits in integration

Though important technical benefits have been obtained in the design, migration and execution of the software itself as soon as 2015 (first deployments), the following years in production saw even bigger benefits when the services started to be used in integration-based scenarios. The following are examples of possible benefits when using business-oriented APIs in the design of a software application and its implementation as an embedded part of the customers IT system.

The long-awaited reuse of software components by MGDIS has been a reality in 2015, when a microservice used to manage business events has been developed in 80 days for a first application, then integrated in 10 days in a second application and finally integrated in a few days only starting from the third one. The integration was purely based on HTTP mechanisms, namely API calls, webhooks and iFrame integration. From the four-layer diagram point of view, the integration is the capability to offer additional business functions (layer 2) by technically mixing software services (layer 3).

In 2016, an application has been created purely by MGDIS marketing service, only helped by an integrator to inject data and provide the UI integration with dedicated stylesheets, etc. This is an example of “Low Code” / “Zero Code” before the terms were largely coined in the past years.

In the process of instrumenting European Subsidies distribution, a high level of traceability and auditability was required and initial analyses showed coding the corresponding features would take a few hundreds man.days. A careful decomposition in atomic features made possible to match most of them with existing software services in MGDIS’ Business Capability Map, dropping the time needed to implement to only a few tens of man.days, while increasing quality, since the corresponding components had already been used in production for more than a year for most of them.

The penultimate example below shows a particularly high return on investment for the decomposition of the monolith into an API-based application. The business case was to automate the main business process provided by the application, in order to accommodate a particular scenario where lots of scholarship grants had to be treated in a very short delay, namely when students obtain their “Baccalaureat” with honours and earn a financial help if they stay in the region for their further studies. In order to accommodate this new customer request, it would have been necessary to add mass-treatment capability into the software application, but also the ability to generically search into a list of data where links to the students were not direct. In addition, some new business rules had to be designed into the software to take into account this new way of distributing subsidies, while making their application generic, since all French regions do not use the same mechanism. Finally, an archiving feature has to be put in place too, as the administrative requisite were strong in this scenario. All in all, the estimation of the necessary development in the application was a bit more than 400 man.days. Since this was too expensive for a development, a brainstorming session was carried which led to a fully integration-based solution around the existing API. Microsoft Excel was used to query the requests and students’ information from the software API using PowerQuery. Then, the students results listing was incorporated into a dedicated worksheet and PowerQuery used again to link the data together and apply business rules (most users being proficient in Excel would then easily change these rules without any external help). Finally, a VBA macro would loop over the prepared data to execute for every line a list of API calls to recreate the business operations needed to declare a student financial help, automatically instruct and prepare its payment. Error management was small but functional, with an additional column used to indicate whether success was reached or not. Finally, the electronic archiving feature was directly obtained by the fact that all operations were grouped in a single Excel workbook, which can easily be made read-only and sent into a dedicated server folder that would then be taken into account by the archiving process. This Excel file took less than 10 days to design, approve by the customer and document, thus proving MGDIS with the highest gain ever through API use.

Finally, the “guichet unique” example is another high-value-added scenario where the correct decomposition of APIs has led a capital role. Before the microservices architecture, the monolithic application provided SOAP web services, but those were a mere exposition on the technical innings of the application. The goal of the “guichet unique” is to provide the user with a single web application where he or she can describe a project and request funding from many administrations. Such a website had been prototyped by MGDIS around 2012, but more than 600 man.days were feed without success. Using the new architecture and in particular the fact that APIs were separated between the “project” resource and the “finance” resource (including from the REST

point of view), the same goal was obtained in 2016, with only 100 man.days (half of it being used for security enhancement), making this the first "guichet unique" in France, between the Pays-de-la-Loire Region and the Vendée Departement. Not only the right granularity of the APIs made possible to realize this breakthrough, but it allowed the company to achieve it in extremely short delays. Once again, spending much time obtaining a near-perfect comprehension of the functional and business situation still provided for an excellent return on investment in time.

## VII. TECHNICAL OUTCOMINGS

The fact that the functional point of view drove the whole migration operation does not mean that the technical point of view was forgotten and many best practices were also learned with relation to the third layer of the diagram.

**Choosing the right tool for the right function.** One of the main advantages of a microservices application in technical terms is the possibility of using the best techniques, languages, frameworks and dependencies for each services, independently from the other services. This was not possible in the monolith version of the application, where a single application forced to use a single programming platform. By opposition, the business functions for managing individuals and legal entities have been served during three years by a re-conditionned .NET service, only being rewritten in NodeJS at the very end of the project, for deployment reasons (Docker being only supported on Linux at that time). Another advantage of microservices is that it has also made it possible to let the technical teams in charge of the choice of their tools and languages and by rendering them responsible of their choices according to the whole application. In a monolithic application, this unique "one shot for five years" choice is so important that high management tends to take part in it and include strategic constraints, which is not the case with a microservices implementation.

**Evolving business should be supported by evolving technical tools.** Each technical service has to constantly follow the evolution of its dependencies in time. This is easy when the technology is stable enough and takes this issue into account. An example of a stable platform is Java and its tooling. This is much more a challenge when the technology is young and changing rapidly. Such an example is the deployment, which not only followed 11 versions of Docker[5], but also followed every technology evolution, from simple use of Dockerfile to Docker Compose, then Consul as a services registry, followed by a traditional coupling with HAProxy[6], and finally support of Traefik[7] in order to solve this problem. When the implementation is modular and closely aligned to the business functions, changing the former is not an issue, as the latter stays stable meanwhile.

**The technical format should describe changes and not states.** Master Data Management, from a functional point of view, deals with business entities, which change constantly. JSONPatch (RFC 6902) proved extremely useful to implement a semantics-correct referential. Indeed, traditional approach to Master Data Management is to store the different versions of an entity. This leads to more volume, and has always caused trouble in development because of the complexity of dealing with concurrent modifications. Coming back to the semantics of an entity modification, which basically is a set of atomic operations of creating / value modification / deletions, accompanied with metadata like the value date of the change, the origin and author of the change, etc. helped creating referentials that generate no locks, hence a better performance. In addition, this approach made it extremely easy to comply with regulations about traceability and auditability, since all modifications are stored. Pushing this atomicity to the client also helped reduce bandwidth use and greatly improve the GUI performance by extending asynchronicity.

**Security also benefited from a functional approach**. Though security is often seen as a purely technical issue, cutting down services from their business domain function helped achieving Defense In Depth with multi-layers dependencies without jeopardizing users comfort. Indeed, stopping a service software process for security reasons would lead to only one function being unavailable to the users, thus strongly limiting the impact on business. The fact that microservices allowed for many security curtains made them great as an implementation of the security in depth paradigm. ANSSI Principal Engineer Philippe Wolf[8] illustrated this pattern as "mango versus coconut", where the external services are soft and easily targeted but the more one approach the core if the information system, the tougher it gets. By opposition, a coconut is hard to break in, but will release all its juice once a simple hole is made. Where security in depth has typically been realized with one to two levels of firewall or dedicated networks, microservices frequently allow for three to five levels of security.

## VIII. CONCLUSION AND FUTURE WORK

Many additional technical outcomes have been collected along MGDIS' three-year long journey into microservices adoption, but the present article focused on the majority that were related to the "functional first" approach.

Correct management of multitenancy, challenges on microservices security (and also great advantages that their compartmenting provides), specific use of general technical tools like messaging queues, schemaless databases and retry policies made for lots of returns that may be addressed in future works. We will first follow our research work on identifying microservices granularity. Despite the common acceptation of DDD [18] and norms as criteria for services granularity, there is still work for additional approaches, and works like [29] have already studied the field with an approach based on performance improvement by finer granularity, providing a quantitative view of the above qualitative proposal for optimal granularity searching.

Though the investment in this architecture has been considerable for a company the size of MGDIS, it should be noted that a vast majority of legal tenders and prospects have

[5] Docker: Enterprise Container Platform https://www.docker.com/

[6] HAProxy - The Reliable, High Performance TCP/HTTP Load Balancer http://www.haproxy.org/

[7] https://traefik.io/, the Cloud Native Edge Router.

[8] Philippe Wolf (IGA / French Government Cybersecurity Agency) - Conference at ENSI Bretagne Sud on state of the art in cybersecurity – Vannes, 25 September 2014.

been won over in the last two years, and the expected return over investment is currently diminishing to an estimated five years sales, the estimated lifecycle of the product being ten years, not taking into account the better durability of a modular software.

## REFERENCES


[1] [Henderson &al. 93] Henderson, J. C., & Venkatraman, H. (1993). Strategic alignment: Leveraging information technology for transforming organizations. IBM systems journal, 32(1), 472-484.

[2] http://fr.slideshare.net/ewolff/nanoservices-and-microservices-with-java [last accessed on January 12th, 2017]

[3] https://www.w3.org/TR/2004/NOTE-ws-arch-20040211

[4] Hirschheim, R., & Sabherwal, R. (2001). Detours in the path toward strategic information systems alignment. California management review, 44(1), 87-108.

[5] Chan, Y. E., Huff, S. L., Barclay, D. W., & Copeland, D. G. (1997). Business strategic orientation, IS strategic orientation, and strategic alignment. Information systems research, 8(2), 125-150.

[6] Galliers, R. D., & Leidner, D. E. (2014). Strategic information management: challenges and strategies in managing information systems. Routledge.

[7] Cassidy, A. (2016). A practical guide to information systems strategic planning. CRC press.

[8] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., & Zaharia, M. (2010). A view of cloud computing. Communications of the ACM, 53(4), 50-58.

[9] Linthicum, D. S. (2000). Enterprise application integration. Addison-Wesley Professional.

[10] Keller, W. (2002). Enterprise Application Integration. Erfahrungen aus der Praxis. dpunkt.

[11] Chappell, D. (2004). *Enterprise service bus*. " O'Reilly Media, Inc.".

[12] Schmidt, M. T., Hutchison, B., Lambros, P., & Phippen, R. (2005). The enterprise service bus: making service-oriented architecture real. IBM Systems Journal, 44(4), 781-797.

[13] Newman, S. (2015). Building microservices. " O'Reilly Media, Inc.".

[14] Fowler, M., & Lewis, J. (2014). Microservices. ThoughtWorks. http://martinfowler. com/articles/microservices. html [last accessed on February 17, 2015].

[15] Namiot, D., & Sneps-Sneppe, M. (2014). On micro-services architecture. International Journal of Open Information Technologies.

[16] Marz, N., & Warren, J. (2015). Big Data: Principles and best practices of scalable realtime data systems. Manning Publications Co.

[17] Diot, C., & Gautier, L. (1999). A distributed architecture for multiplayer interactive applications on the Internet. IEEE network, 13(4), 6-15.

[18] Eric Evans (2003). Domain Driven Design, Tackling Complexity in the Heart of Software.

[19] Gregory Hohpe & Bobby Woolf (2004). Entreprise Integration Patterns.

[20] http://download.microsoft.com/documents/australia/soa/gartner.pdf

[21] O'Brien, L., Brebner, P., & Gray, J. (2008, May). Business transformation to SOA: aspects of the migration and performance and QoS issues. In *Proceedings of the 2nd international workshop on Systems development in SOA environments* (pp. 35-40). ACM.

[22] Rosen, M., Lublinsky, B., Smith, K. T., & Balcer, M. J. (2012). Applied SOA: service-oriented architecture and design strategies. Wiley & Sons.

[23] http://ieeexplore.ieee.org/document/5476743/

[24] N. Bieberstein, R. Laird, K. Jones, T. Mitra (May 5, 2008). Executing SOA: A Practical Guide for the Service-Oriented Architect. IBM Press.

[25] http://apiux.com/2013/09/12/webhooks/

[26] https://sendgrid.com/blog/whats-webhook/

[27] https://opensource.com/article/17/7/software-standards

[28] https://www.enterprisetech.com/2017/06/14/well-enslaved-proprietary-clouds-unless-collaborate/

[29] O. Mustafa, J. Marx Gómez (2017). Optimizing economics of microservices by planning for granularity level. http://soft.vub.ac.be/~cfscholl/ProWeb17/ProWeb_2017_paper_8.pdf

[30] J.-P. Gouigoux and D. Tamzalit. From monolith to microservices: Lessons learned on an industrial migration to a web oriented architecture. In Software Architecture Workshops (ICSAW), 2017 IEEE International Conference on, pages 62–65. IEEE, 2017.

[31] https://www.cigref.fr/cigref_publications/RapportsContainer/Parus2003/2003_-_Accroitre_l_agilite_du_systeme_d_information_web.pdf

[32] P. Edwards, M. Peters, and G. Sharman. The effectiveness of information systems in supporting the extended supply chain, Journal of business logistics, Wiley Online Library, vol. 22, Number 1, pages 1--27, 2001.