2017 IEEE International Conference on Software Architecture Workshops

# From monolith to microservices

## Lessons learned on an industrial migration to a Web Oriented Architecture

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 SA is a software editing company that underwent a major strategic and technical change during the past three years, investing 17 300 man.days rewriting its core business software from monolithic architecture to a Web Oriented Architecture using microservices. The paper presents technical lessons learned during and from this migration by addressing three crucial questions for a successful context-adapted migration towards a Web Oriented Architecture: how to determine *(i)* the most suitable granularity of micro-services, *(ii)* the most appropriate deployment and *(iii)* the most efficient orchestration?**

***Keywords—microservices; migration; Web Oriented Architecture***

## I. Context of the study

MGDIS SA[1] is a French software vendor editing 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 [3]. The initial objectives were (i) to provide a highly-ergonomic web frontend, (ii) to ensure long term evolution and (iii) to ease low cost interoperability with many other software products. The main strategy used to fulfill these goals was based on Information System Alignment[2] from the beginning [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.

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 2015. 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.

Having explained the context in the current section, the paper will present the followed approach to achieve this migration in terms of lessons learned by extracting and presenting from the return on experience important findings. Section II presents targeted technical problems, Section III explores the determination of granularity of services, namely microservices, Section IV details services deployment, Section V explains integration and orchestration concerns, while Section VI provides a first evaluation of Return On Investment, before concluding with Section VII.

## II. Targeted problems

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.

First, how to split the former monolith into granular APIs? Knowing the technical bits does not help in any way in knowing *where* to cut. A lot of literature explains the microservices approach, but this is most often cited in the context of extremely high-volume web application, which is not the case for MGDIS.

Second, when the services are defined, how can their deployment be realized, since the methods traditionally used for monolithic applications do not fit anymore?

Third, once the services are defined and installed, how to make them work together? When starting with the rewriting, nobody in MGDIS had any prior experience on neither Enterprise Application Integration [9, 10] nor Enterprise Service Bus [11, 12], not even working knowledge of service orchestration in general.

## III. Granularity of services

The first return of experience that we propose is that **the choice of granularity should be driven by the balance between the costs of Quality Assurance and the cost of deployment, as illustrated in Fig. 1.**

The cost of QA can be calculated on a given release by adding up the time spent by testers validating not only the new features but also the non-regression on existing ones with the time spent on release management. The cost of deployment is

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

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

978-1-5090-4793-2/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSAW.2017.35

the time spent by operational teams to deploy this new release, also in man.days. It decreases a lot as teams automate this operation.

Deployment time generally does not depend on the number of additional features in the release. As for QA, automation of non-regression tests help maintaining a low cost for existing features, but new ones come with costly manual test or writing of automated test scenarios. Thus, comparison of both costs must be related to the business added value of the new features, typically estimated with agile methods in man.days or feature points if one wants to be independent from the team's velocity.

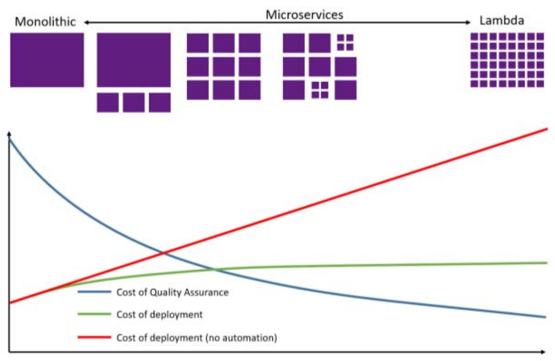

Fig. 1. Cost-based determination of granularity of services.

Defining the right granularity was a difficult try and error process, during which our Business Capability Map went up and down in terms of number of services. Technical and functional feedback led us to join some services and split some others, thus changing the granularity. For example, joining the GUI and backend responsibilities of services made the number of services in the Business Capability Map decrease from 73 to 52.

The main difficulty in searching for the right granularity was that, to our knowledge, there is no state of the art on this subject of services granularity. First, there is currently no commonly-accepted definition of the desired size of a microservice [13, 14, 15]. The measurement unit knows no consensus and the size of the service can be measured in lines of code, in number of features, in terms of the consumption of server resources under standard operational conditions, and so on.

In addition, the trend towards microservices accelerates, and newly-coming architectures tend to diminish the size of the services, with so-called "nanoservices" (as cited in Microsoft Azure Functions and lambda [16] / serverless [17] architectures (one key proponent being Amazon with its product called AWS Lambda).

It was thus clear from the beginning for MGDIS that we would have to find a way to determine the granularity of the services by ourselves. Fig. 1 is our way of formalizing this experience on granularity of service, and an attempt to make it generic.

On the right-hand side of Fig. 1 lies the finest-grained architecture, namely lambda architecture. On the left-hand side of Fig. 1 lies the coarsest possible service, namely monolithic architecture where one software process embeds all exposed functions. In monolithic software architecture, the coupling between the components is massive. This results in a huge cost of Quality Assurance, as one modification may have an impact on any feature of the application. The cost of running such an application is limited, though, since there is only one process to install, monitor and keep in healthy condition.

As one progresses towards the right-hand side of Fig. 1, the granularity becomes finer, which has two consequences. First, the cost of quality assurance (blue line) decreases, as it becomes possible to test and validate services one by one, since they are truly independent (at least from the point of view of unit tests and functional tests). Second, the cost of running the whole increases with the number of services (red and green lines).

If the deployment is not automated by any means, the cost increases linearly with the number of services (red line). The point of crossing of blue and red line indicates that the optimal architecture in this case is to extend the existing monolith with additional features based on independent services. This gives the advantages of keeping a battle-proofed, stable application while allowing the addition of features using the new, more evolutive, architecture. This approach has been used for many existing customers of MGDIS, who still use the old version while using alongside new REST-based services for specific features (e.g. European Funds and PKI-based financial evaluation).

If the deployment is automated (see section IV below), the cost of deployment becomes asymptotical (green line). The optimal granularity is logically higher and the resulting best architecture for MGDIS was a new application completely based on fine-grained services, which is the one operated on new customers and that we describe in this paper.

**This attempt of modeling the granularity of services has been backed in practice by the fact that it corresponded well to other criteria of granularity, for example aligning the service onto consortium-based norms and standards of the related business, or using the notion of Bounded Context in the sense defined by Evans in Domain-Driven Design [18]**.

## IV. Deployment of Services

After cutting the business functions into services comes the second phase, namely deploying the services.

**MGDIS's main finding in this area was that there is a strong link between services-based architectures and containers technology**. Deployment of the old monolithic application with all its dependencies and prerequisites could be a matter of hours, which meant deploying an application composed with 40 services would have theoretically last a week or so, which of course was completely out of question.

Container-based technologies, and in particular its most-known implementation Docker, made deployment of our new application possible through several characteristics:

- A Docker image contains all its dependencies, which means a given service can be treated as a black box, only exposing its API in exchange for resources.
- The containers are by default sealed from one to another, which results in guaranteed low coupling, without the

high cost associated with virtual machines.

- Docker Compose made it possible to easily deploy any number of services, by composing in a text file an application made of several services.
- Docker Swarm mode allows for complete decoupling of the containers and the machines supporting them. In its recent version 3, Docker Compose allows for Distributed Application Bundles, which define applications made of several services without any dependence other than the presence of a Docker host IP address and access credentials.

## V. INTEGRATING SERVICES

Once the services have been correctly designed and deployed, the question of orchestrating them needs to be addressed. MGDIS has progressed through several tries and errors, which hopefully will provide some experience for similar development teams. Before addressing this feedback, it should be recalled that the software integration landscape has greatly evolved between 2013 and 2016, which made necessary a change of approach during the rewriting of the considered application. As SOA gets slowly replaced by Web Oriented Architecture, Enterprise Service Buses fade out and leave room to smaller footprint integration techniques, such as RSS-based asynchronous communication [23, 24] or webhooks-based events [25, 26] and data exchanges between services. The following findings must be read while keeping this background in mind.

**One of the first reality we hit was that an Enterprise Service Bus is overkill for anything else than very large organizations.** MGDIS knew from the start that an ESB would be too big an integration method for her smaller customers (departmental councils, with a few hundred users) and targeted her first ESB experiences to larger ones (regional councils, with many hundred users). It turned out that only ministries and large regional councils (with an equivalent number of users, but many more IT resources) were a good fit for an ESB. In particular, since SOA is a centerpiece between functional, managerial and technical deciders, applying it with a centralized solution like an ESB is only possible if everybody in the organization is aligned on its benefits, which proved rare.

**While still using an ESB, the second lesson we learnt was that only some particularly important API streams should be considered to flow through the ESB.** The cost of associated connectors is indeed extremely high, particularly when using techniques like Enterprise Integration Patterns [19] and its Apache Camel implementation, where experimented developers are extremely scarce, at least in France.

A possible alternative to an ESB is to use a Business Process Management engine but we also experienced that it is not adapted to our context. Business Process Modeling Notation remains a great tool for business analysis and modeling, particularly since version 2.0 is now largely accepted and feature-complete, but **its use to automate orchestration routes is too complex for mid-sized applications**. If the customer IT teams are trained, successful uses for interoperating coarser-grained applications have been observed, though not released in production at the moment. But managing routes between fine-grained APIs is simply too expensive to run on a BPMN engine.

**Another difficulty with BPMN engines is the risk of re-introducing coupling** in a loose service-based application by using the same application for process design and documentation (which is functional) and business execution (which is a software application).

After several years trying these different approaches, MGDIS found its sweet spot for microservices integration in the use of webhooks, which are a light, unobtrusive, HTTP-based way of communicating between APIs. ESBs are left to the large customers' initiative when they want to take charge of connecting the webhook events to APIs in a different, more sophisticated manner than just plugging the output of the event to the API without any mediation. Thus, the default behavior is passive choreography, and the customer is able to insert a middleware in order to switch to active orchestration.

Together with Docker Compose, webhooks-based integration proved to have the best low coupling to complexity ratio, and the advent of Swarm mode in Docker 1.13 will improve this ratio even more.

## VI. FIRST RETURNS ON INVESTMENT

Experience shows that SOA projects have often been disappointing, in contrary to what has been claimed, because of low actual reuse of services. Most of real service-oriented applications did not obtain the promised benefits of SOA [3, 5, 6]. By migrating towards web-oriented architecture and by leaning on appropriate microservices, we extracted three main returns.

**Increase of reuse.** Though most services have only been production-ready for less than a year, **a significant amount of reuse has been observed**. Without taking into account common services like authentication, Business Activity Monitoring, etc. which are almost always shared, about a third of the Business Capability Map has already more than one use, and ten out of seventy services are used in more than two different contexts, some of them in many context and in particular in contexts not known by MGDIS, as external integrators have acquired them and use business-oriented microservices for their own purpose.

Another way to evaluate reuse is to observe the time needed to create an application similar to another one, once the microservices they are based on have been created. It turned out MGDIS had this kind of experience, since three applications that are extremely close have been released in the past three years. The first one, which took in charge most of the business microservices development, used 160 days of development. The second, which added some features to existing services, took 50 days of development. The third one actually was completely designed by integration and configuration of existing services and thus cost only 10 days of production, none of which were assigned to actual programming of services.

Finally, a strong financial gain happens when the microservices produced by a given company for its own software are reused by other software editors or integrators for a completely different domain. This has happened to MGDIS for a few low-level services that have been purchased by partners and external companies for their own integration in software. Interestingly, this was not an initially thought-of business case.

**Replacement facilitated.** Reuse is definitely the most sought-after characteristic of a service, because it brings an automatic reduction of cost, but **ease of replacement is an additional positive side effect of well-built services**. In particular, MGDIS has observed in production that microservices based on norms or pivotal formats were quicker to validate from a quality assurance perspective, but also produced much fewer bugs when replacing another version. The former monolithic application was typically verified by QA in a few weeks, while a single service is currently validated in a day or so. Of course, the total amount of time for QA is roughly the same, but the microservices architecture has made it possible to spread it at will along time, bringing a better Time To Market. As a bonus, customers are much more willing to replace a service on a well-known and restricted domain by its newer version than they were to qualify and put in production a complete monolithic application with all associated potential impacts to be solved at once.

**Performance increases.** After reuse and replacement, **performance is the third main advantage that has been observed in production** by MGDIS on its new architecture. Health diagrams are currently exploited by support and hotline at MGDIS where an error is brought up when a given service exposed online has an average response time of more than 1 000 milliseconds. In 99% of the case, the actual response time is between 70 and 300 ms (metrics for December 2016). To give a comparison, the announced objectives of maximum response time for a SOAP call in the old application were to be lower than 3 000 ms, and this could only be achieved in 95% of the cases. In a particularly coarse and complex web method, the average observed time was 11 000 ms (this value was recorded after a performance-optimization campaign, and could not be reduced further in the old architecture, despite many efforts).

**Finally, much lower levels of support have already been empirically observed** on the new application, but the metrics are currently mixed with bugs from the older version of the software, and cannot be correctly interpreted at the moment. The only statement that can be said for now on this subject is that, while customers have transferred to the new application and a burst in defects were expected, MGDIS actually removed one person from the support team while the stock of defects went on diminishing. This is considered as a financial improvement roughly equivalent to 70 000 € per year. The average bugs stock decreased during these three years from 320 to 90 only. For reasons of confidentiality, we cannot give more detailed numbers about support.

Being able to add Business Activity Monitoring by pure integration (acting only on the messages) and without releasing any new version of the related microservices is foreseen as a source of benefits for the near future, but this feature has not been exploited yet and thus cannot be stated for sure. Yet, it has been asked eagerly by several customers.

## VII. Conclusion

The numerous advantages obtained through the change of architectures are currently compensating the huge investment granted by MGDIS, and the financial return over investment is expected in less than five years' sales (the estimated lifecycle of the product being ten years). Thus, while not being a proven success yet, our experience might still prove useful for others in similar context, and expectations are high in the success of the products, with an encouraging rate of sales.

An important return of experience of the few years recently passed aligning the functional and technical capabilities is that this is not a technical issue only. To achieve our goal necessitated to jointly tackle the problem from the technical, methodological and management points of view. It was a deliberate choice in this article not to develop the adaptation to agile methods, the management approach and the attempt of using Conway's law to choose the right team for the right service implementation, but there is a lot to be said on these subjects. This may be treated in further articles.

## 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/