A method to set up the microservice architectural style

The combination of:

- the deployment solution based on containers,
- Devops – a process favoring continuous delivery,
- and the growing expectation for business agility…

…is driving the new trend in software architecture, namely "microservices".

Nevertheless, the term doesn't designate a technical solution; rather, it is an architectural style, a certain way of structuring IT systems. As such, it raises questions that pertain to logical architecture, a discipline that complements technical skills and is required for the transformation and improvement of IT systems. The main question addressed by logical architecture is: What is the optimal structure of the IT system? This discipline benefits from a decades-long tradition, which covers SOA, object-oriented approach, structured design…

Once the technical conditions or choices have been identified, architects have to design the structure of the system. They usually start with schematic diagrams that categorize the software components, introduce technical solutions, and fix their inter-relations[1]. Beyond these instructional drawings, we need a rigorous approach that can guide the decisions we have to make, as architects, in order to effectively lead the transformation of the system. This article introduces the principles that inspire the methodology we propose.

1. Start with the business representation

First and foremost, everyone would agree on this premise:

The IT solution should be aligned with business needs and strategy.

That said, we have to draw the consequences. This implies that we need a proper representation of the business. The first thing we see in the enterprise is its organization, actors and activities. This spontaneous approach of the business reality ranks among the functionalist approaches. It gives primacy to how we perceive the activities. This is something we have to undergo, since activities – whatever you call them: functions, capabilities, use cases, processes – are part of this reality. However, it entails a difficulty: as we are considering the enterprise in its organizational aspect, we are faced with organizational variability. Organizational choices, role specifications, management style… evolve naturally, as the enterprise adapts to new conditions. Thus, the business activity model is neither stable nor sharable. In a group, different entities may have different ways of doing things and would likely struggle to agree on the same process model. For a given entity, if its activity model remained unchanged, it could underperform quickly.

As a consequence, when the purpose is convergence, simplification, agility... we need a more generic representation. We need to isolate the core business knowledge, using abstraction and expelling variability.

We must recognize, above this "pragmatic[2]" (organizational) aspect, a more abstract one, made of business objects, regardless of local habits and, of course, independent of technical choices. We call this

---

[1] The best one I know is from Pierre Bonnet, a veteran in the SOA approach. For a start, see: https://www.linkedin.com/feed/update/urn:li:activity:6498879819575623680/.

[2] From ancient Greek « *pragma* » meaning « action ».

the "semantic" aspect. The semantic model is not only a sort of conceptual data model; it aims to express the core business knowledge.

We have to isolate the business fundamentals: the essential knowledge that enables the enterprise to act in its environment.

Contrary to the pragmatic model, the semantic model is very stable, especially if the modeler has given free rein to the genericity and has remained faithful to the external reality. By doing so, the model relies on universal notions, which are constant and easy to share across the value chain.

Distinguishing between these two types of models is not a new practice[3]. It obeys a very strong and basic principle in the methodological tradition, namely the "separation of concerns" principle. It helps to analyze the business reality, and potentially to reinvent it. What concerns us here from the perspective of microservices, is that this segregation between two "upstream" aspects will result in considerable impacts, for the better, on the technical system physiognomy.

An enterprise, from a business point of view, has to be described through two types of representations, separating the core business knowledge, on the one hand, from the activity and organization, on the other.

---

2. Structure of the IT system

When equipped with the two business models – semantic and pragmatic –, we can search for a better structure for the software system.

2.1 We need an intermediary aspect

If we design the system structure directly in terms of technology and technical choices, we will get a representation which will be subjected to technical change. Also, there will be a risk of entering into excruciating detail. Such a representation will make it impossible to drive the IS transformation in the long term. Moreover, we need a representation that we can present to the decision-makers to discuss investments.

For all these reasons, our frame introduces an intermediate aspect, between business and IT: the logical aspect. This is where the structural decisions regarding the software system will be made.

Logical aspect: Aspect of a system that provides an abstraction of its logistic and technical means.[4]

---

[3] Only the wording has changed. For the sake of pedagogy, the term "semantic" has replaced "conceptual", just to avoid confusing the semantic model with a data model, be it a conceptual data model. A well-designed semantic model is something very different to a data model. It aims to formally express all the business knowledge. So, it contains the descriptions of information, actions and transformations, including rules, constraints and object life cycles.

[4] The definitions used in this article are excerpts from the Praxeme thesaurus, available on: http://wiki.praxeme.org/index.php?n=Thesaurus.Thesaurus.

2.2 The system is stratified

The first decision an architect has to make regarding this logical aspect is the choice of an architectural style. Within the scope of this paper, we are considering the SOA style[5]. The "microservice" approach rejuvenates the SOA philosophy, assuming the same principles.

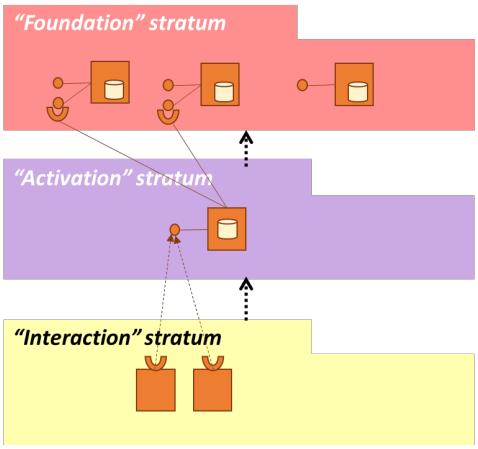
Then, when we enter the logical aspect, what do we discover at the first level? Here, the logical architect has no decision to make. The method provides the answer because it is a matter of obvious reasoning. Since we found it appropriate to separate the two business aspects, it is only natural to maintain this separation when constructing the system. This leads to the notion and identification of strata[6]. The first stratum corresponds to the semantic aspect. It is populated with the outcome of the derivation rules applied to the semantic models. We will come back to this notion of derivation later. For now, the important message is that, at the core of the IT system, we will find constituents that reflect or translate or automate the business objects and the related knowledge. We call this stratum "Foundation" as it takes up the business fundamentals. As regards the building process, it is obvious that we should start with the foundations.

The second stratum of the logical architecture accommodates the elements derived from the pragmatic aspect. Since they equip the business activity, they assume the role of orchestrators vis-à-vis the constituents of the "Foundation" stratum. As this stratum is about the business activities, we propose to refer to it as the "Activation" stratum.

A third stratum is added to take care of the constituents that handle the communication between the system and its environment. It is called the "Interaction" stratum.

As a result, the first level of the logical architecture draft is instantly in place, provided that you adopt the method. The following figure sums up the stratification principle, which also states that communication within the system is polarized. This means that an interaction constituent can call only activation constituents, which orchestrate foundation constituents.

Figure 1. The stratification of the logical architecture



Comments on the figure

From a logical standpoint, the system is stratified: its substance is segregated into three strata. The constituents can communicate inside a given stratum – not too much, as we will discuss later. They can invoke services that belong to the immediate antecedent stratum: interaction constituents call activation constituents, which can call foundation constituents.

These topological constraints are stated in the method of logical architecture and design. Then, the logical architect digs into the details. Doing so, he/she has to be cautious when establishing responsibilities and dependencies.

The stratification principle enforces the idea of aligning the IT system with business logic. Indeed, it maintains the segregation between semantic and pragmatic elements. This is of paramount importance in terms of the structural quality of the system, as well as for its evolution.

The next figure positions four of the seven aspects that the Praxeme method identifies when approaching the Enterprise System[7]:

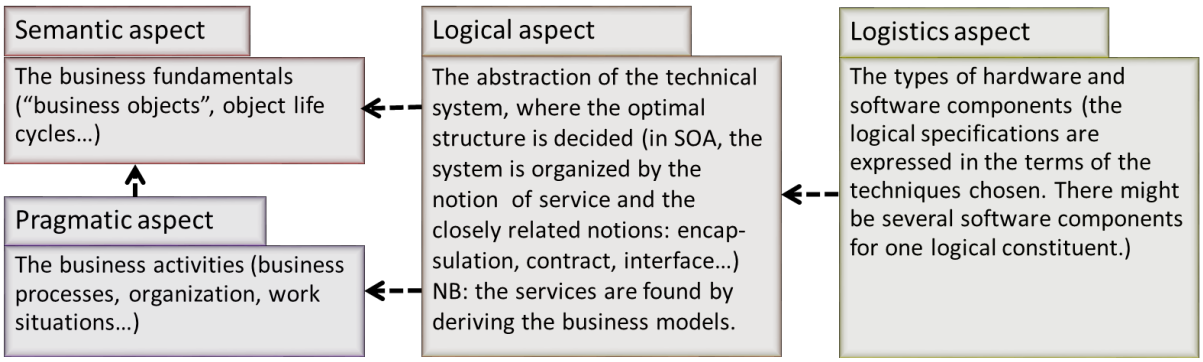
[5] Service Oriented Architecture. This style is based upon the metaphor of a consumer requesting services from a provider.

[6] The term "stratum" (plural "strata") is favored over "layer", so that there is no confusion between logical terminology and technical usage.

[7] The seven aspects, their definitions, content and relations constitute all together the representation frame, also known as the Enterprise System Topology. It provides the theoretical basis of the method.

- upstream, the semantic and pragmatic aspects, covering business objects and business activities, respectively;
- central for our purpose, the logical aspect, connected to the previous ones through derivation rules;
- finally, the logistics aspect, that includes both software components and technical solutions.

Figure 2. The positioning of the logical aspect



2.3 For describing the system, a metaphor consistent with the notion of service

The term "service" used in SOA and "microservices" has no technical ground. First of all, it stands for the analogy of the dialogue between a customer and a provider. The analogy conveys many decisive notions, like encapsulation and contract. To stay true to this metaphor, we will use the term "service" to designate an operation that replies to a customer's request and that realizes something, which the customer is expecting[8]. In this terminology, a Web Service or a microservice are not services; they are software components, which bear services, their exposed operations.

To document the entire logical architecture, the notion of service is not enough. The terminology has to be complemented with aggregates tiered on several levels. A set of coherent services is called a machine. Several machines sharing the same resources make up a workshop. A set of workshops under a common responsibility is called a factory. This is the terminology that Praxeme proposes for the logical aspect, with the concern to spin the service metaphor. Whatever the terminology you choose, you need one. The Praxeme terminology is the one we will use within the scope of this article.

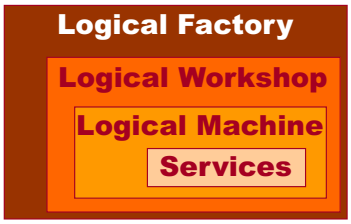


Figure 3. The metaphoric terminology applied to the logical aspect

2.4 How to discover the services… and the rest

To pursue the idea of aligning computer design with business perception, a derivation mechanism takes place in the method. In the technical sense it has in the context of MDA[9], to derive is to produce a modeling element – the target – from another modeling element – the source. The source element belongs to an antecedent aspect, while the target element is immersed in an aspect that must be connected to the first one. For example, a logical machine is derived from a semantic class or from a use case. The relative position of the logical aspect against the two business aspects allows these two derivations. This illustrates the role and importance of a framework to support the design practices.

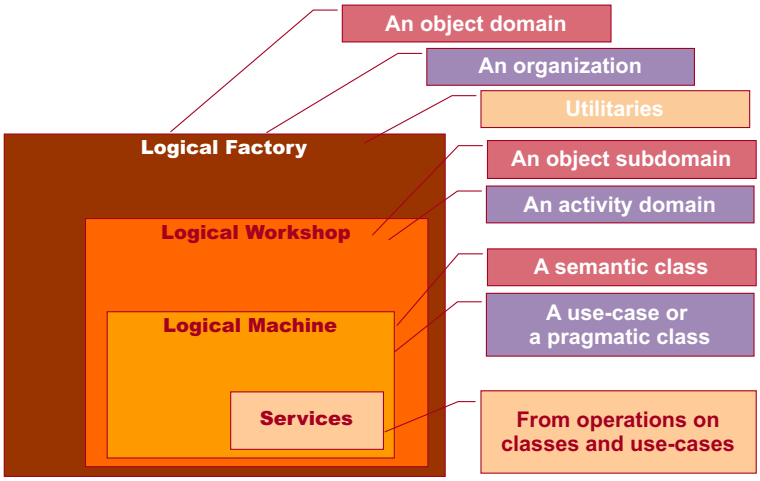
The Praxeme method for logical design and architecture provides practitioners with the derivation rules that help them to produce most of the substance in the logical aspect. Other derivation rules exist from the logical aspect toward the logistical aspect. Then, software can be generated and can enter the round-trip development cycle.

[8] When customers/clients/solicitors solicit/ask for a service, they don't expect to receive a component (the bakery, the office…). They ask the provider to do something for them (to provide the service). The service corresponds to the fact that something is done, i.e. the execution of an action. The best way to model such a fact is by using the modeling element that the UML standard calls an operation. In the Praxeme method, "service" is a stereotype associated with an operation. Not every operation is a service.

[9] Model Driven Architecture is an OMG standard, which Praxeme takes advantage of, alongside the technique of UML profiles.

More important than the logical terminology, is the way every element is obtained, depending on its nature. The figure below gives a quick idea.

Figure 4. The origins of the different kinds of logical constituents



Comment

Around 80% of the services are derived from operations found in the semantic and pragmatic models. Of course, this procedure assumes that the business models are sufficiently detailed.

The method procedures are more specific than suggested by this figure. In particular, they involve three sets of derivation rules, because the logical aspect displays three facets:

1. substance (logical constituents that specify the software components),
2. persistence (logical data models that anticipate the physical data model),
3. exchange (exchange structures to mold the flows).

There is much to be said about these three kinds of models, starting with their locations inside the system. This leads to architecture decisions being made. The method proposes the frame that helps to inventory the many issues to be addressed. It builds upon theoretical thinking, methodological tradition, as well as on practical experience.

The logical model comprises three facets and specifies the services, the persistence solutions and the pivot language. As it is mainly produced by reference to the business models, the procedure guarantees the business-IT alignment. It escapes the pitfall of services deduced from existing and often ill-structured solutions.

If we are to think the IT systems anew, for the sake of business adaptation, we need more than vague recommendations, and we have to find a starting point somewhere other than in the legacy system. This is precisely what the method proposes by starting with thorough upstream representations, then applying derivation rules and documenting architectural decisions.

3. The connection to the microservice approach

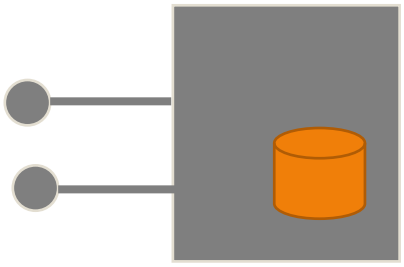
3.1 How to properly identify the microservices

To return to our topic, it is time to answer the question: how do we identify and delimit the microservices?

First, we have to admit the discrepancy between the term “service” and the use of “microservice”. A microservice has nothing to do with the notion of a service, as defined above (in the sense of a service rendered by the system)[10]. In logical terminology, the microservice corresponds to the logical workshop. We have to go back to this central notion.

[10] Why “micro” in “microservice”? On the one hand, it is opposed to the usual silos that structure most of the existing systems. On the other, it denounces mishandled practices in the field of SOA. When literature compares

Figure 5. The symbol of a logical workshop



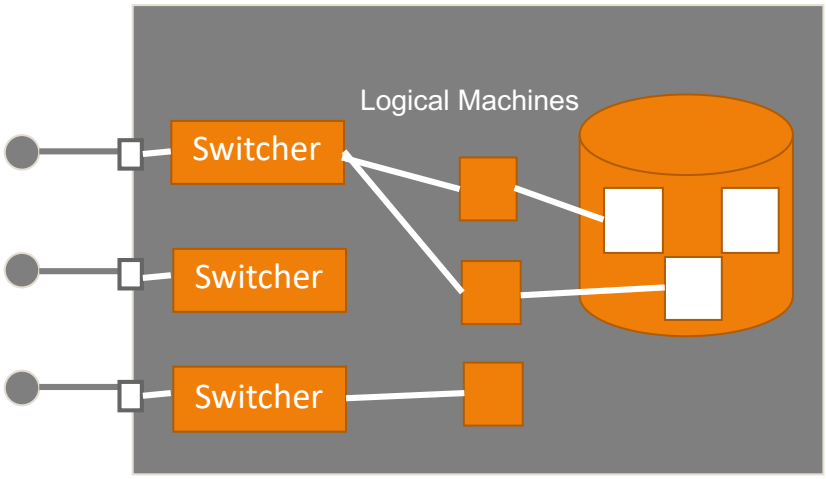
a) The logical workshop is defined as a black box: in accordance with the principle of encapsulation, it hides its resources, primarily its database or persistence solution.
b) In return, it exposes its services – what it can provide other components with – through at least one interface, most often two or three interfaces.
c) An interface is a list of operations, the exposed services. It is a selection of what the workshop can do. The way services are assembled to form interfaces determine the coupling level, as well as the dynamics at run time. So, architects have to think twice when specifying the interfaces[11].
d) To sum up the method, workshops come from two sources: object subdomains in the semantic aspect and activity domains in the pragmatic aspect.

From these features of the workshop notion, it follows that:

The logical workshop (a notion in the logical aspect) is the specification of the microservice (a sort of software component, belonging to the logistical aspect).

Thus, the method proposes a path that leads us to identify and specify the microservices. We have insisted, above, on the external properties attached to the logical workshop notion. The method also provides logical designers with procedures to design the content of the workshops. The following figure gives us a glimpse of this part.

Figure 6. Inside a logical workshop



Comment

There are as many switchers as provided interfaces. As the name implies, a switcher handles the list of the exposed services, and dispatches the requests on the internal resources.

Each table of the database is under the exclusive control of two machines: one in charge of the set behaviors, the other for the elementary behaviors (on a single object).

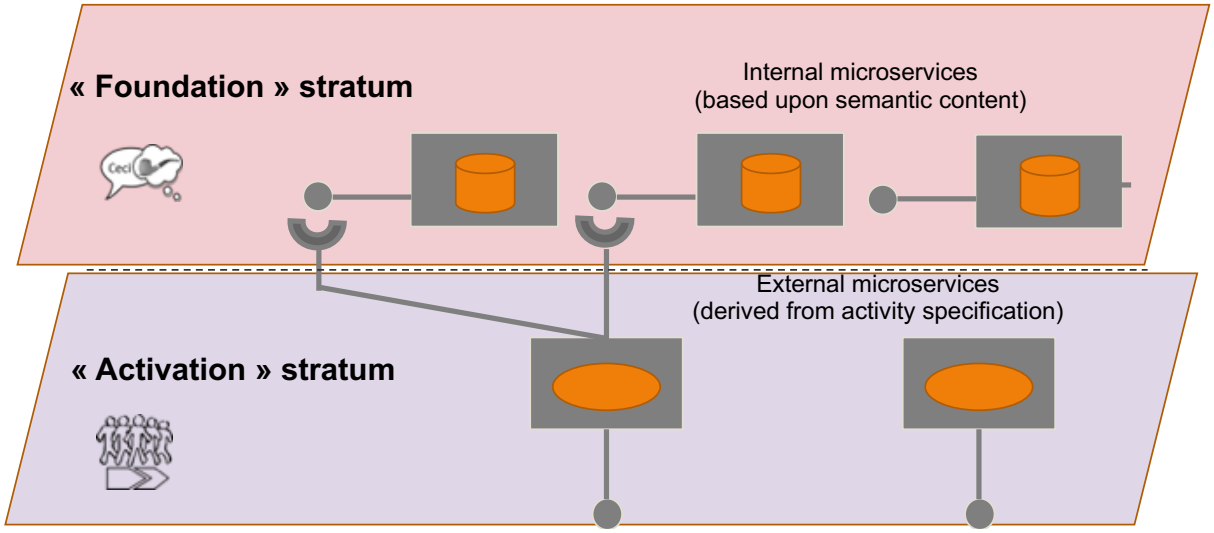
3.2 The new physiognomy of IT systems

At this stage of our reasoning, we have set up a new structure for the technical system, which the next figure summarizes.

the microservice approach to SOA with the intention of denigrating the latter and praising the former, it is thinking more of the hazardous practices purported to be service-oriented than the full-fledged SOA philosophy, which has been widely misunderstood and betrayed. That is why some have felt the need for a renewal, heralded under the microservice standard.

[11] When API programs leave the responsibility of identifying interfaces to the projects, it obviously cannot work. Proper interfaces can only arise from an overall view of the system, or even a federation of systems. Those ill-guided API programs will result in the ruin of the API approach, which is close to the SOA approach. The right approach calls for a strong architecture function, with the consequent authority.

Figure 7. A schematic diagram of the IT structure at logical level



The ideal IT system is made up of two kinds of microservices:

- in the Foundation stratum, microservices that reflect subdomains of the semantic aspect, and that have exclusive responsibility on a coherent cluster of objects (here symbolized with the database symbol);
- in the Activation stratum, microservices that are built to supply a business activity (symbolized with the use-case shape).

The latter orchestrate services that they request from foundation microservices, in exactly the same way that an elementary business activity manipulates several objects. The logical architecture tries to eliminate dependencies between microservices of the same stratum, in an attempt to reduce coupling to its bare minimum. In the same endeavor, it eliminates redundancy: in particular, objects of a specific type are located inside one and only one microservice, which drives their transformations. Through its interfaces, the objects can be shared between many activation microservices, used in various business situations.

3.3 The data architecture issue

In the initial experiences with SOA, the first brake on the approach was the habit of resorting to large databases. Cutting persistence solutions so that the workshops would hide them and handle them exclusively, was an idea admitted as a "pure" SOA conclusion, though this idea ran up against existing database practices. Nowadays, in the wake of the microservice tendency, things are changing, and IT decision-makers are less reluctant to carve up smaller databases or other persistence devices. As a consequence, data architecture can comply with the recommendations of the method, more easily than in the past. By the way, this certainly constitutes the main feature of the new approach, in practice.

The architecture of the data marries that of the services. The service plane overhangs and hides the data plane.

In addition to this architecture of the optimal system – loosely coupled and rid of redundancy –, today's styles of architecture include a subsystem, which cannot be determined by derivation. The inspiration comes from big data and business intelligence solutions, spurred on by governance concerns. This subsystem, let us call it the "data pot" or "data pool" (or "data hub", as you please)[12].

[12] It is a common pot, into which all data are poured. The image of a pool meets that of the data lake, for those who favor aquatic metaphors. Unable to decide between the two terms, we could coin the phrase "Pool Pot", except that it evokes bad memories!

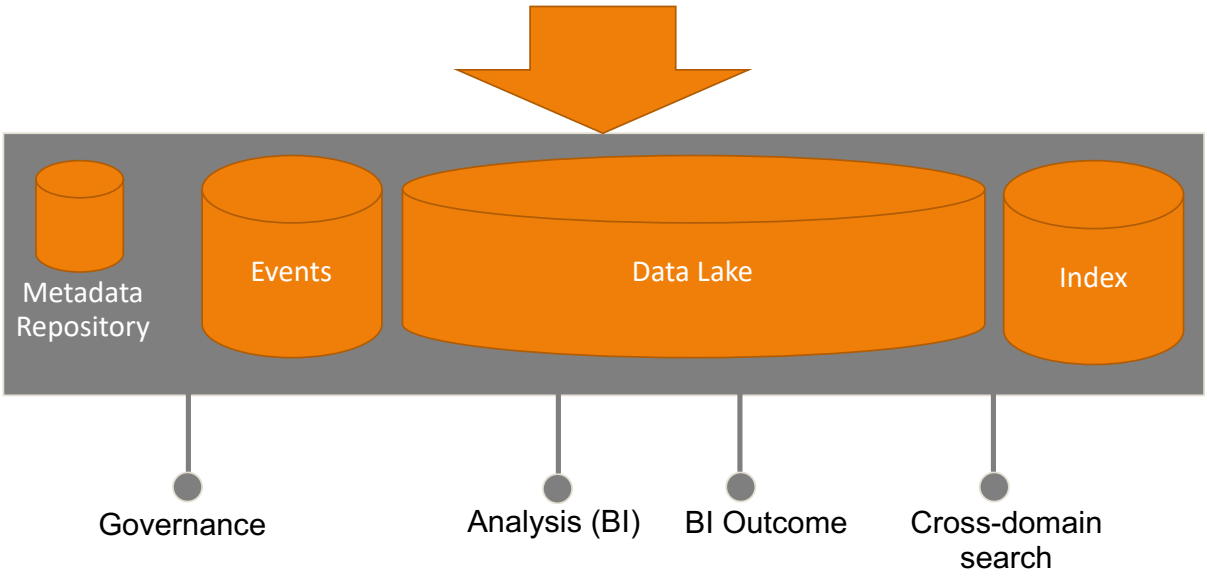
This subsystem violates the rule of the single location of each datum (no redundancy). It precisely exists to duplicate every bit of information, and to record every change. This chosen and voluntary redundancy can be explained by the following reasons:

- coherence, which requires managing the transactions across several databases – since the smaller databases are distributed among many microservices;
- governance, to check that regulations and rules are applied effectively;
- performance analysis, namely BI;
- performance in search, by complementing the data architecture with a cross-database index function, in order to accelerate requests.

The Data Pot is a subsystem that shadows the rest of the system, and serves transversal purposes.

It incorporates mechanisms such as event management, data lake and data analysis, and the metadata repository. These solutions may seem disparate, but they have in common the need to take the information and subject it to transverse uses; simply speaking, it is about data management. For this reason, and for architecture considerations too, they are gathered together in one single subsystem, even though, from a deployment perspective, they could remain separate. How we can integrate this subsystem in the whole system is a big topic for architecture design. The first recommendation is to limit the contact point: ideally, there should be only one downstream line, feeding the data pot from all legal sources. Then, it is possible to publish several interfaces; some turned toward special activities related to governance (data and rules), others used to inject analysis lessons into the operations. Due to the scope of this article, we will not pursue this topic further.

Figure 8. The Data Pot, for an integrated data management, in principle



Caveat: this design mentions technical devices, which the enterprise may benefit from; but one must not forget that such a design infringes the principles we have referred to, so far. As a reminder, the best architectural solution to these concerns (governance, transaction management, etc.) is one located within the normal microservices. To take an example, a transaction implies several foundation microservices; therefore, it cannot be handled by only one of them; but, since the transaction occurs in the span of a certain use case, the activation microservice that equips this use case could handle the transaction.

With this discussion, we sit in the middle of a dispute: one trend drives us toward the technical panacea; the other one pulls us toward the logical architecture thinking.

4. Implement and deploy the IT System

Another key figure of the logical workshop is:

> *The notion of logical workshop is built to serve as a deployment unit.*

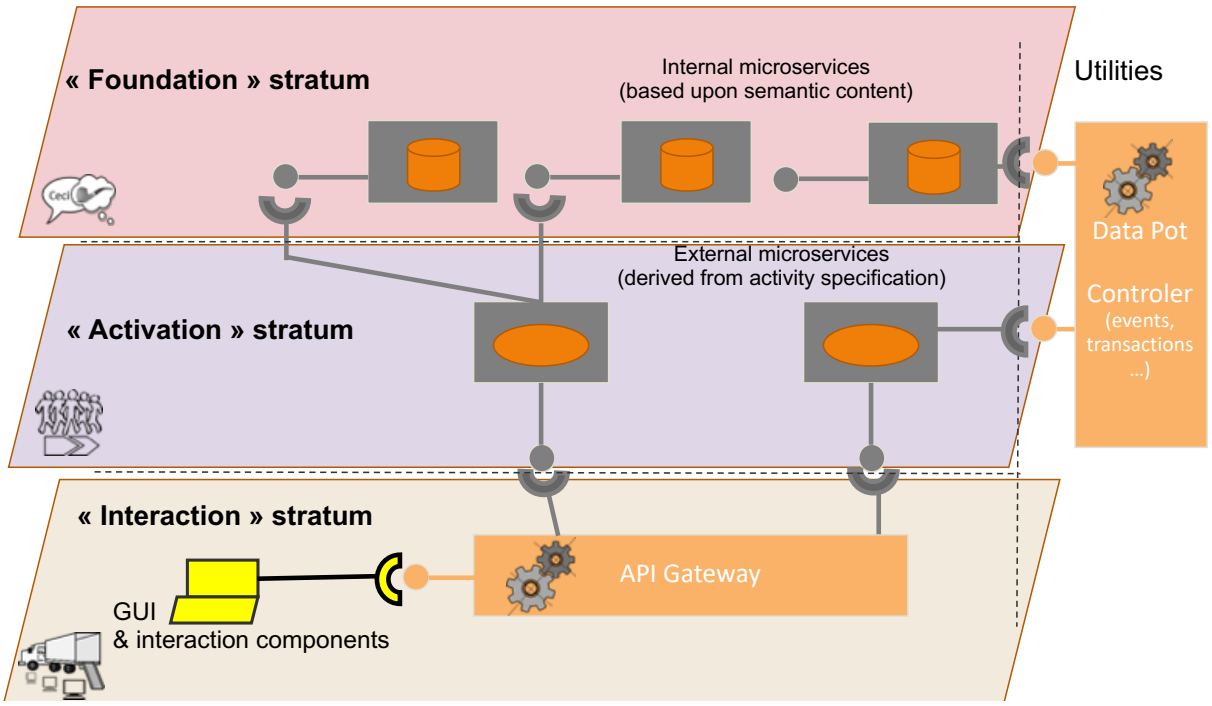
As such, it merges with the notion of microservice. As a workshop/microservice encapsulates its resources, it can be instantiated and located easily, through the deployment process.

Deployment pertains to the physical aspect. It requires its own representations, mainly based on the UML deployment diagram.

At this level, some additional appliances appear:

- service mesh, which deals with communication between microservices;
- interaction components, such as GUI, Web Services for external communication, batch programs coordinating activation services;
- API Gateway, an apparatus that optimizes the choice of the more appropriate interface instance.

Figure 9. The schematic diagram covering the three strata and utilities



To go further on this subject, see the paper of Pierre Bonnet at https://www.smart-up.org/micro.html. Pierre Bonnet pioneered SOA and MDM in large companies, starting in the 2000s. I owe him my initiation to the SOA approach. His technical skills, his visionary inspiration, as well as his obvious stamina and strong personality make him a true leader, the kind you need to succeed in ambitious transformations.

As a conclusion, the microservice approach rejuvenates the SOA philosophy. It can be implemented with technical devices, which:

- exist mainly in the open source domain;

- are lighter than what has been implemented in the previous SOA attempts (ESB, Web Services).

An essential success factor when adopting this approach is the method. Without a thorough and documented procedure, the general discourse may delude many, and lead to wasted investments. Praxeme is an enterprise transformation method. It encompasses every aspect of the Enterprise System, pulling together the many disciplines needed to transform a system. Among them, logical design and logical architecture bring the detailed specifications and the overall plan for building or transforming the IT system. In order to guarantee the business-IT alignment, it promotes business modeling. Not only is the outcome harnessed to design the business, but it is also used to guide the software design, by means of derivation rules.

The released method documents are available on: http://www.praxeme.org.