A Jolie based platform for speeding-up the digitalization of system integration processes

Claudio Guidi, Balint Maschio
ItalianaSoftware S.r.l
cgudi@italianasoftware.com, bmaschio@italianasoftware.com

1 Introduction

Integrating systems is becoming a more and more urgent issue even for those companies that possess small IT systems characterized by few applications. The demand for digitalised business processes has recently increased thanks also to some governmental initiatives promoting smart factories [1-3]. The IT market offers many solutions for dealing with different aspects of a modern digital business, some of them are cloud-based solutions others are provided on premises. This abundance makes the problem of how to integrate the different digital solutions relevant because of the high costs in terms of realization projects and maintenance. Thus, it often happens that the integration project for enabling a new digital solution within an IT system costs as much as the solution itself. Hence, system integration issues usually play the role of a bottleneck w.r.t. a digital transformation roadmap because of the indirect costs it brings. From our daily experience, we noticed that large-size enterprises address integration problems with Service Oriented Architectures (SOA)[4] that are very expensive both from the point of view of the required products and consultancy, but they can afford them because of the large IT budgets. On the other hand, small and medium enterprises, whose financial and organizational capability do not allow them to implement a SOA solution, have difficulties in approaching system integration from a systemic point of view and they usually tackle these kind of problems with *ad hoc* solutions that are not replicable and raise difficulties in long-term maintenance and control. In this scenario, microservices represent a new architectural approach and a concrete opportunity for building integrated systems based on services without necessarily dealing with the high overhead imposed by a SOA solution. A microservices architecture indeed, holds some characteristics that make it ideally suited for a digital integration project: *(i)* microservices are naturally modular and reusable making the design and development of integrated processes more gradual and easy to handle; *(ii)* they can scale efficiently, making the digital integration project less susceptible to changes in request load; *(iii)* ideally, they can be adapted to run in different kind of infrastructures making them very attractive for all those enterprises where the hardware and software come at a premium. These three factors could put any enterprise in the position of developing its incremental digital integration strategy with microservices that can be scaled with the available budget without losing the benefits of a systemic approach. Despite the advantages, microservices introduce a great level of complexity in terms of governance and maintenance of the system. The number of different components available in the system dramatically increases and addressing issues like deployment, monitoring and development require different stack of technologies, which, consequently, require different skills that are not so easy to be found in the market. Thus, the main risk microservices are facing now is to appear expensive as much as a SOA solution is.

In this paper we intend to show how microservices architectures approached from a linguistic point of view can dramatically reduce costs in the development and maintenance of system integration solutions. We will provide support to our thesis by showing a real and concrete scenario in a production environment where we pervasively used the programming language Jolie[5] for implementing system integration solution. Moreover, we will show some basic features provided by a Jolie based platform called Jung[6] which can further accelerate the development and the maintenance of the discussed

solutions. Here the authors consider that, even if Jung is a commercial solution, some results about its application in real world deserve to be discussed in order to illustrate how a new generation of tools based on a new programming language like Jolie can be used.

2 The reference architecture

In Fig.1 we report the process architecture we adopt for dealing with system integration by exploiting microservices. In such an architecture all the integration processes are developed by exploiting microservices which are represented within a so called *System Integration Space (SIS).* In the system integration space each microservice is equipped with a service interface and it supplies specific functionalities that are useful to the end of system integration. A microservice is an independent process which takes the form of a service. It can be invoked by other microservices, it can be re-used in different scenarios and it can act as a microservices orchestrator by composing different microservices in order to fulfil a complex task. Microservices can interact with external applications by exploiting specific connectors that permit to connect them as they are microservices too.

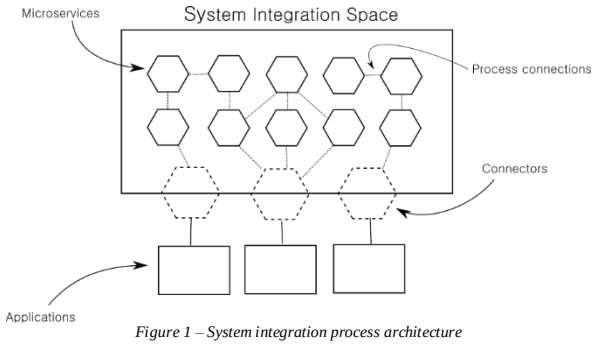

Figure 1 – System integration process architecture

2.1 Microservices categories

All the microservices within the *SIS* share some common characteristics but they can be differentiated in different categories depending on the nature of the provided functionalities.

The *shared* characteristics are:

- **Independent deployment**: each microservice can be deployed and run separately by following separate lifecycles.
- **Publication of a formal interface defined in a unique and common language**: there is a unique linguistic tool for defining interfaces within the SIS. In our case, we exploit the characteristic of the Jolie language to provide also the linguistic tools for expressing service interfaces.
- **Implementation with a unique technology**: all the microservices are implemented by exploiting a single technology that in our case is Jolie.

Microservices categories:
Each microservice can be categorized as it follows:

- **Pure computational**: the microservice performs pure computation, it receives a set of parameters and it produces an algorithmic manipulation of them returning a result. *As an example let us consider the calculation of the italian fiscal code starting from personal data like name, surname, city if birth and date of birth.*
- **Pure data provider**: the microservice just provides functionalities for accessing and manipulating the data stored into a particular data source as a set of files, a database or an electronic device. *As an example let us consider a microservice connected with a database of accounts which permits to access the data of the accounts to other microservices.*
- **Pure orchestrator**: the microservice just orchestrates existing microservices by coordinating different calls to them, collecting their replies and refactoring the final response as a composition of all the received data. *As an example let us consider the case of an orchestrator which returns both the account data and its italian fiscal code obtained from the two microservices cited as an example at the previous points.*
- **Mixed microservices:** the microservice can combine more than one of the previous characteristics together. Thus it could be both computational and data provider, or it could also be a mix of all the three characteristics: computational, data provider and orchestrator.

3 Engineering system integration

On top of the process architecture described at Section 2, system integration can be engineered by exploiting a systematic approach which allows for a rational governance of its issues. The steps we follow are described in the following diagram:

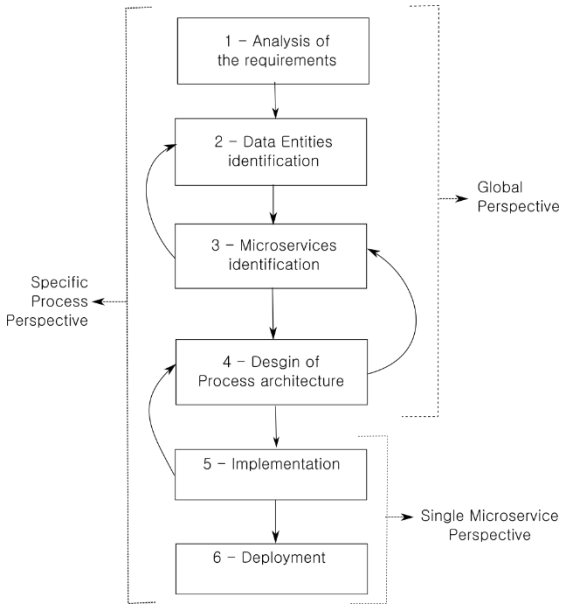

Fig 2 - System integration engineering diagram

1. **Analysis of the requirements:** this step is devoted to rationalize the requirements of the system integration request by providing a human readable documentation where all the requirements are listed in a clear way.
2. **Data Entities identification:** this step is devoted to identify all the data entities involved in the system integration process. The data entities are directly defined in the form of Jolie types. Existing data entities already exploited in other integration workflows can be selected and added to be used in the current project.
3. **Microservices identification:** starting from the list of the involved data entities, a list of microservices necessary to provide or manipulate the needed data entities is defined. Microservices can be also selected from those that are already running in the system serving other integration processes. Depending on them, the list of data entities could be modified in order take into account the data entities already in use into the system.
4. **Design of process architecture:** all the required interfaces and the workflows among the selected microservices are defined in order to achieve the target integration process. Existing microservices could require a refactor activity in order to satisfy the new requirements, new microservices could be developed. A list of implementation guidelines are provided.
5. **Implementation**: in this step all the new microservices are developed and the existing ones are modified where necessary. Unit and integration tests are performed in this step.
6. **Deployment**: all the microservices are deployed into a development environment in order to finalize tests within a production-like environment then they are moved to production.

It is worth noting that the engineering diagram moves along two main directions: the main one is the classical *top-down direction* which starts from step 1 to step 6, whereas the other is a *bottom-up direction* which works as a feedback loop from 5 to 2 that allows for a correction of the previous assumptions. Here the idea is to shorten steps 1, 2, 3 and 4 in order to focus on the implementation phase for detecting inconsistencies with the previous assumptions.

3.1 Teams and knowledge

As it is possible to note from the diagram at Fig. 2 both a global knowledge of the system and a specific knowledge about the single microservices to create, are necessary in order to engineer a system integration project. The steps from 1 to 4 involve a global knowledge of the system. The main outcome of these steps is the definition of the implementation guidelines which must reduce redundancy and improve re-use of existing microservices. Existing microservices are usually evaluated depending on the interfaces they exhibit and the workflows they are involved. A deep technical knowledge about their implementation is not required at these phases. The steps 5 and 6 on the contrary are more focused on the single microservices to be implemented and on the technical details which concerns their realization. Once reached these steps indeed, the global knowledge about the system are crystallized within the implementation guidelines and the team can focus their activities on the single microservice they are deal with. Nevertheless, the technical activities on single microservices could require to revise some decisions taken at the previous steps.

Microservices are usually developed by small teams where both global knowledge and specific technical knowledge must be present. Since there is always a high level of interconnections among the microservices which participate in an integration workflow, a communication link among all the teams involved in the same integration process must be kept alive during the implementation phase in order to catch those local modifications which have impacts also at the global one.

4 Jolie and Jung

Jolie is a microservice specific language that allows for a fast and reliable definition of orchestration processes. Its Java/C like syntax is coherent with the microservice paradigm and crystallizes core concepts like interfaces, endpoints, communication primitives and workflow-like behaviours. In Jolie everything is a service or a composition of services. Such a feature represents one of the main differences with other programming languages where services are always obtained as a result of an abstraction of the application of a given framework, e.g. Java[7] and Spring Boot[8]. In Jolie a developer does not deal with external frameworks but she only exploits the primitives offered by the language producing clear and simple scripts for defining microservices. Interface, communication primitives and business logics are available within a unitary linguistic domain which provides a compact and easy means for directly operate on microservices programming without requiring any additional intermediator tool.

Jung is a commercial digital platform that provides a powerful deployment chain tool for Jolie-based microservices. This tool enables to handle multiple deployment nodes (called Cloud Nodes), allowing the programmer to upload, move, delete and configure instances of Jolie microservices. Each microservice can be deployed independently from the other with different versions of the same microservice coexisting in the same cloud node. The platform is equipped with a scalable monitoring tool, that can track messages throughout all the microservices involved in a specific behaviour.

In the following table we report the main advantages we experimented by using Jolie and Jung w.r.t. the engineering diagram presented at Section 3.

Phase	**Jolie**	**Jung**
1. Analysis of the requirements	-	-
2. Data Entities identification	Data Entities are directly expressed in the format of Jolie types which can be used in the next phases	Existing microservices interfaces and types are registered into Jung and they can be retrieved and consulted
3. Microservices identification	-	Existing microservices are registered into Jung and they can be searched and analyzed
4. Design of process architecture	Microservices interfaces and types are directly written in Jolie. They are part of the implementation guideline.	-
5. Implementation	All the microservices are implemented in Jolie. No special frameworks are needed for obtaining the service artifacts.	A specific cloud node is dedicated to development tests. Microservices can be deployed in it in order to be tested.
6. Deployment	-	All the microservices are deployed and monitored within Jung.

5 A real scenario

At the present, Jung based solutions are adopted in more than ten different production environments for dealing with archiving documentation flows from SAP[9] to third party applications. Here we discuss the impact on a specific customer where their usage had strong results on system integration projects from the point of view of time to market, costs and human resources organization. In the table below, we report the time to market acceleration obtained w.r.t. the estimated days provided by direct competitors.

Project	Direct competitor estimation	Jolie based microservices approach
DMS Project	60 days	15 day
Maintenance Management	40 days	10 day
Marketing API	20 days	7 days
DMS Project (upgrade)	1 day ~ 3 days	3 hrs ~8hrs

6 Conclusions

In this paper we presented our experience in the engineering of system integration solutions for SMEs by exploiting microservices architectures together with specific technologies like Jolie and Jung. We described the engineering process model we follow during an integration project and we show how the technologies we selected can allow for a simplification of the project phases because they favorite an easiest management of the team knowledge. In particular, Jolie as a unique programming language for dealing with microservices development facilitates the communication among the team and its artifacts like types and interfaces can also used during the design phase as implementation guidelines. Moreover, Jolie does not need any specific framework for converting the business logic programming paradigm into a service one because programmer directly develops in microservices[10]. Jung is an administrative suite which help teams in the management of the Jolie artifacts and microservices by allowing navigation of interfaces and deployment of microservices.

Such an approach allowed us to speed-up the design and the development of system integration solutions in the customers we collaborate with. In particular, we enabled small teams to be autonomous in the development and the management of a microservices based system targeted to achieve system integration.

As a future work we intend to enrich Jung with specific choreography tools which can help in the design of process architecture by formally defining global view compositions.

2. References

[1] “INDUSTRIE 4.0” - http://www.plattform-i40.de/I40/Navigation/EN/Home/home.html
[2] “Nouvelle France Industrielle” - https://www.economie.gouv.fr/nouvelle-france-industrielle/
[3] “Fabbrica Intelligente” - http://www.fabbricaintelligente.it/
[4] “SOA Reference Architecture” - http://www.opengroup.org/soa/source-book/soa_refarch/p7.htm
[5] “Jolie” – http://www.jolie-lang.org
[6] “Jung” - http://www.italianasoftware.com/jung.html
[7] “Java” – http://www.java.com
[8] “Spring Boot” - http://spring.io/projects/spring-boot
[9] “SAP” – http://www.sap.com
[10] “A Linguistic Approach To Microservices” – Microservices Community Conference, Odense 2017 – “https://github.com/klag/talks/blob/master/conf-microservices/2017/LingusiticApproach.pdf”