MEISTER

ON INTEGRATING SAP WITH ANYONE

MEISTER V2.0 ORCHESTRATION ENGINE

Andre Rosenthal
Gateway Architects, LLC
2600 Dallas Parkway suite # 600, Frisco, TX

GA
Gateway Architects

Table of Contents

Product Document

Document Name:	On integrating SAP with anyone
Document ID:	MV2INSP0001
Document Owner:	Andre Rosenthal
Document Version:	1.0
Document Date:	Monday, 26 August 2019
Document Status:	Draft

MEISTER

Introduction

Much was said in the literature about integration points and the processes entailed in a successful marriage between discrete points of execution of a business process, so it is expected, at least cursorily, that the topic is at its exhaustion, or at least near its peak of understanding. However, the journey of integration needs to be understood in terms of time as well as technology, and as expected, time has a tendency of altering the reality of technology deployments, so much so that, if ignored, one expects only the status quo to be retained, and disruptive technologies, however amazing they may be, are discarded since their focus seem to imply a complete new set of rules for business processes, thus threatening the leverage of these in favor of old, traditional proven processes.

Meister 2.0 is an example of such disruption, and consequently introduces all threats to the status quo, both in terms of technology acumen as well as process execution. This paper analyses the comparison of status quo between the perceived standards of integration with its dominant members, and Meister, the disruptive technology.

What is integration?

We define integration as a function

$$f(x) \mid \exists\ x\ in\ X \rightarrow\ \forall\ x_i\ in\ X\ \exists\ y\ in\ Y\ that\ maps\ x_i\ to\ y_j\ for\ any\ i\ and\ j.$$

If such function exists, it states that, regardless of the values of the variable x_i and y_j, a data parity mapping is achieved and unique. It is obvious that such function is of little values if it depends on the indices of the variables, as it would mean that one has to create an infinite set of such functions in order to integrate disparate system. That is, one is forced to craft by hand such functions for each particular case and maintain these functions with respect to the life cycle of solutions. Not only this is labor intensive and time consuming, but it also injects a costly change management into the overall solution. One only needs to see that when a new set of functionalities requested by the business, the revision of these functions and the (almost always) crafting of new ones becomes a requirement, yielding in an ever increase in the total cost of ownership and maintenance of the solution.

Now consider a small revision on the function declaration; we propose the change of definition from the traditionally expected declaration as given before into the following new function:

$$g(f(x)) \mid \exists\ x\ in\ X\ and\ y\ in\ Y \rightarrow\ \ \exists\ map\ M(f,g)\ defining\ a\ transition\ from\ t\ to\ g.$$

This new function $g()$ becomes a mapping integrator which is now independence of $x\ and\ y$ and only provides a guarantee that, if such function exists, it will be able to achieve the exact same thing as the first version, but this time without being dependent on the path taken. Further, if we augment this new definition to allow $g()$ to operate on set of all possible $f(x)$

functions, we have indeed provided a generic mapper of all possible transitions from a data in a system into a (possibly new) data on another, and the process of execution of such map becomes fully encapsulated into a logical unit not dependent of the chosen path, so long as a path (which could be the best possible or unique) is found. As such, the new definition gets improved, and now looks as the one defined below:

$$g\left(\sum f_i\left(x\right)\right) \mid \exists\, x\ in\ X\ and\ y\ in\ Y \rightarrow\ \exists\ map\ M(f,g)\ defining\ a\ transition\ from\ t\ to\ g.$$

We call this function $g()$ above the Integrator function, or in simple form, the orchestration engine behind the mapping of discrete data from different systems.

Orchestrations Disruptiveness

Meister 2.0 disrupts the technology in the following aspects:

1) It is not bound by time: that is, it requires no batch processing to be implemented. This means that at any given time, data between the systems are kept in sync, so long as the time factor is at least larger than the lowest time-quanta required to transact the data within the boundaries of the unique system.
2) It is not based on offline processes: that is, it does not require a drop of a snapshot of data into specific repositories to be effected by the either participants. This means that, regardless of the frequency chosen to time factor mentioned above, the data is not pre-stored anywhere.
3) It requires nothing else but Meister as transaction coordinator, and the nodes where the data is perceived to be the system of records: that is, no need for external ETL or parsing of data is required. This means that Meister 2.0 becomes the overseer of the orchestration between the nodes and manages the life cycle of the operation on its entirety.

One needs not to dive deep to realize that, amongst all the solutions present in the marketplace as the time of this writing, Meister is the only one that breaks down the notion of batch processing between nodes and merges the requirement of data synchronicity as nothing else but another ACID transaction between data points. The impact of this change is immense: the overall vision that businesses need to provide different paths of execution (with their own set of limitations and breaking points) in order to coexists between different data repositories is eviscerated and, as a consequence, creates a cloud of confusion. As an example, consider the following technology stacks for traditional orchestration:

1) SAP PI/PO
2) BizTalk Server with SAP Adaptor
3) The SAP ERP
4) A 3rd party partner

This stack is commonly seen across landscapes as the de-facto proven solution for integration (viz. the I and O in the SAP product.) However, this stack exists solely to allow for data movements between parties where an expensive human interaction is required; after all, one needs to create the orchestration portion of the solution by hand, and once done, it is locked down and versioned for future needs, with the repetition of all efforts done before for new functionality.

Now consider the Meister 2.0 approach:

1) Meister 2.0 installed on the SAP ERP
2) The 3rd party partner

Since there is no dependency between the content of the data transmitted by the 3rd party partner (per augmented function defined above), Meister would pick the request and process it as if it was any regular transactional request for the sake of the SAP ERP system. That is, anything that is capable of being handshake between partners is done as if it is a transaction over the SAP ERP system. The simplicity of this statement is overwhelming: it just eliminated two touch points of the ecosystem with a generic one that is already bolted in the SAP ERP and runs natively there. The costs savings is realized immediately: no longer the business needs to procure expensive and lengthy projects in order to change requirements. This simplicity comes with its own set of uncertainties: how is it possible that Meister 2.0 encapsulates the process of a generic orchestration which works regardless of the path taken? In summa: companies have been told for years they need expensive and convoluted solutions to do the most trivial operations and accepted prima facie that this is indeed the truth. The fallacy of the previous statement is the focus of the introduction: the time effect.

The meaning of the time effect

As data is parked at a repository waiting to be executed, it becomes stale by definition. Consider the simple exchange of a non-stock catalog between partners. Ariba, the leading solution of the market, imposes the following:

1) Each member of the chain must define their own translation scheme between their unique data structure into Ariba's internal data scheme.
2) Each member of the chain (supplier and consumer) must adhere to the internal scheme
3) At time of data exchange, it goes from internal structure to Ariba's purified one, then from the purified one into another structure.

What is wrong with this picture? At surface, nothing stands out! It seems logical that partner A has data in a different format (say XML) and partner B wants data in yet another format (say iDoc). So, perfunctory, the process above works very well. That is, until partner A introduces a new token in the data chain. Then, for each possible version of the XML produced by partner A,

there must exists a new translation scheme, which in time, would spaghettify the whole process. Given enough time the cost of maintaining integrity and efficiency skyrockets and the only one having a good time is Ariba, which cashes in each change done to each map.

What if Ariba is not used then? In this case, we are back to the BizTalk/SAP PI/PO paradigm: let's write ourselves new orchestration solutions, regression tests and all, and wait several months for testing to show no more issues were injected by this change, and if all goes as planned, finally go-live. And the time effect is here once again: from the time the business realizes it needs a change until the time the change goes into effect, it is again necessary to do this process repetitively for each and every new change required. This iterative process, when seen isolated, becomes the status quo imposed by marketing machines of software vendors to sustain the idea that integration is always a batch, catch-up, after thought process, and therefore you are better of conforming with this idea since nothing else is feasible.

The Salesforce impact

All was well and dandy in the SAP ecosystem until Salesforce disrupted the scenario. As a self-contained solution capable of UI augmentation via customization[1], the need to coordinate between transaction units becomes a requirement for business, which falls out of the status quo messages banged on the drums of software vendors for years. As a measure of resistance, vendors (SAP and others) stressed the fact that here too, you need batch processing to be successful. Forget about Master Data replication in real time, which is a no sequitur vision of the real world. Consider the always faithful offline batch a night and all will be good. We wonder why this message is perceived as so strong a statement that, even today, many companies tend to regard the data governance as a mere inconvenient and not a real problem.

Meister 2.0 fits into this situation like a glove: just plug from Salesforce lightning code a real-time call to Meister, and SAP will be kept coordinated as Salesforce manages its own realm of influence. However, this is not a mandatory statement either: it becomes a matter of decision at the transaction time which system of records is the dominant data governance entity: sometimes it would be both, and sometimes either! So long as the orchestration at the transaction level is kept with the clear idea of data governance in mind, the residence of the data within any system of record becomes a matter of choice and not an architecture pillar.

Meister 2.0 as the conductor

All previous paragraphs in this white-paper lead to the following question: it is feasible to expect that a generic orchestration engine could be the conductor of the transaction regardless of what is being done? In fact, the answer is yes: by standardizing in one standard (in Meister case, Json Documents) and by encapsulating each transaction node into a specific generic

[1] Itself nothing new since SAP is customizable the exact same way from the early days of R3

mapper, Meister delivers the power of orchestration without having to be aware of what is the content of the Json Document. So long as it is in Json, all is fine. It becomes a matter of implementation at the localized node the inspection of the content of each Json Document, making everything else a matter of transaction coordination agnostic of the data content. The choice of governance becomes a selection of best-fit and not necessarily an architecture decision, capable of morphing in time to adjust to companies' agility and growth.