Journal Article
Journal 36

Benefits by DaaS for FSI – Principles and Limits of Modern SOAs

Ian Maravilla
Kevin Wennemuth,
Oliver Krauskopf

Journal
The Capco Institute Journal of Financial Transformation

Recipient of the Apex Awards for Publication Excellence 2002-2012

Global Finance and Regulation
#36 02.2013

CAPCO FORMING THE FUTURE OF FINANCE

Ideas at Work

Benefits by DaaS for FSI – Principles and Limits of Modern SOAs

Ian Maravilla – Senior Consultant, Capco

Kevin Wennemuth – Senior Software Engineer, Finanz Informatik Solutions Plus

Oliver Krauskopf – Senior Consultant, Capco

Introduction

The initial hype about service-oriented architecture (SOA) is over, and it seems that it is not the next revolutionary step after centralized architectures. On the other hand, the paradigm may still be of use for financial services institutions (FSIs).

Lack of consequence in some recent SOA implementations should not disqualify the paradigm, but instead we should look at lessons learned and create better architecture blueprints for future transformation projects. So from a business perspective of typical FSIs – mainly banks, asset management, hedge funds or insurance companies with considerable amount of capital markets activities – we want to see how the SOA paradigm can be applied to reap benefits.

Academia has made some progress, proposing "emergent virtual enterprises" [see Timm and Scholz (2008)] based on service-oriented architecture. Rapid project or merger-driven assembly of whole legal entities is a truly tempting scenario. However, we will stay focused on the practical aspects, i.e., how SOA applies to FSI landscapes and how data as a service (DaaS) can take place to increase the ability to manage IT complexity.

Our reflection starts with typical business requirements and metrics for business decisions – the starting point for design and planning of architecture transformation. As system environment change leads, with rare exceptions, to significant transformation costs, we want to point out reasons for transforming traditional data distribution topologies towards SOA-dominated infrastructures.

Not just project sponsors, but all stakeholders need to understand and evaluate change drivers. And it's not only the cost of core architecture transformation, such as for implementing new or changing old environments. There are also significant costs for building up new expertise on, for example, particular SOA software suites in project and line department teams, as well as often a cultural change for departments that have maintained legacy systems for years.

Whenever benefits expected by transformation seem to outweigh expected disadvantages and transformation costs, it's likely that projects or whole programs will be set up and budgeted. Already in early planning stages, in our opinion, valid architecture transformation principles are as crucial as a clear business view on cost and planning. So we will discuss principles that help in anticipating benefits, challenging architecture, and creating business roadmaps.

If there are some principles that can drive a successful transformation, then one should consider them. We want to help with that by showing where SOA/DaaS implementation can provide benefits to FSI, and by sketching how.

Although infrastructure as a service (IaaS) and software as a service (SaaS) are also specialties of SOA-backed implementations, we will be focusing on data providing architectures in capital market and FSI architectures, therefore concentrating only on data as a service (DaaS). On the following pages we will focus our examples and considerations on the

We thank our colleagues for providing us valuable feedback and constructive reflections during the development of this article, especially Alan Benson and Jan Stüve.

aspects of DaaS as a specific aspect of a SOA implementation. We will demonstrate how the application of SOA principles enables the fine-grained integration of major data providers in a given architecture.

Typical business environments and metrics for business decisions

Typical business environments

Given the high integration and consistency within modern capital market systems (CMSs), one only needs to consider changes at their interfaces, i.e., regarding CMS inbound and outbound data. However, business areas like high frequency (HFT) trading need latency figures impossible to provide with "classic" (medium or high latency) SOA, thus HFT is also off topic here.

In the following chapters we focus on architectures where SOA/DaaS can be expected to provide benefits, given that it is properly implemented, and some crucial principles are considered.

Staying focused on CMSs, we want to denote inbound/outbound interface areas around capital markets systems to get them evaluated with respect to SOA linkage. Possible linkage points/applications include:

- Static data sources
- Electronic exchange and trading platforms
- Market data feeds
- Reporting systems
- Settlement systems
- Accounting systems (if ledgers not integrated in CMS)
- Risk management systems
- Internal data warehouses (could be replaced by SOA to some extent)

Fulfilling business requirements is the sole IT driver, and the ever-changing financial industry has been always one of the most innovation and IT-resource demanding commercial industries. Thus, FSIs have often been early adopters of new techniques and paradigms, especially in order to stay ahead of competition.

But there are major common architecture styles that can be found in modern FSIs as well as in most other large industries. Ignoring industry-specific business data, the following can be seen:

- Host architecture (e.g., mass data processing in OSPlus)
- Client-server architecture (e.g., Java fat clients for market data enrichment/classical three-tier architectures)
- Service-oriented architecture (exchanges being early adopters for fine grained data calculation and provisioning)

Major exchanges like Deutsche Börse and Market Data Providers were early adopters of DaaS. Many FSIs are currently evaluating the potential of further deployments of DaaS.

Scalability – how easily an architecture can be scaled from small interface amendments up to enabling whole emergent virtual enterprises [see Timm and Scholz (2008)] – is quite an interesting aspect from a strategic and particularly a mergers and acquisitions (M&A) perspective. The lower the barriers for system linkage, the lower the IT cost components in overall merger costs are expected.

Since business is always driven by cost and favors technical innovation, it comes with improved management ratios like total cost of ownership (TCO). Thus, broad TCO scenarios should to be calculated and considered before setting up a large transformation program or project.

Metrics for business decisions

To calculate the total cost of a transformation, one should not only include project budgets but also the costs for internal and external skill development, and the cost of parallel system maintenance for a period of time, for example. The major metrics for business decisions with respect to transformational changes can be listed as follows:

- Total cost of ownership
- Complexity cost (i.e., costs by reducible complexity)
- Transformation project or program cost (budget)
- Delivery frequency and latency

Total cost of ownership (TCO)

Gartner published the first approach to measuring TCO in 1987 [see Wild et al. (2000)], but there are now more formulae to calculate this operational ratio. The Gartner Group, Forrester Research, and Meta Group are three well-known publishers regarding TCO approaches [see Wild et al. (2000)]. Since this paper's main focus is not on IT cost controlling, we extract from the TCO measurement only the key aspects we need for discussion in the following chapters.

These are:

- Hardware and software costs (HS_{TCO})
- IT operation costs (IC_{TCO})
- Process, administration and training costs (PT_{TCO})
- File and data management costs (DM_{TCO})
- Cost for proprietary application development (AD_{TCO})
- Expected downtime cost/expected platform SLA (DC_{TCO})

The aforementioned can be grouped by different accountabilities (direct/indirect[1]) and by stages, but we simply decompose the overall architecture TCO in the following chapters as:

$$TCO_{overall} = \sum HS_{TCO} + IC_{TCO} + PT_{TCO} + DM_{TCO} + AD_{TCO} + DC_{TCO}$$

Hardware and software cost (HS_{TCO}) are assumed to stay constant, e.g., cost for new hardware and software should be equal to

1 Direct costs originate from purchasing, etc., while indirect costs originate especially from a lack in productivity because of insufficient skills/training levels or less than ideal processes.

the current cost. Given that the legacy architectures are not highly unstable, expected downtime cost (DC_{TCO}) is assumed to stay unchanged as well. On the other hand, one expects the IC_{TCO}, DM_{TCO} and AD_{TCO} to be significantly lower after a successful transformation, otherwise no benefits would have been reaped compared to the current situation. Only process, administration, and training costs (PT_{TCO}) are likely or expected to be higher for a while, at least when implementing new software and systems not yet well-known by internal (and external) staff.

Complexity

Complexity is a very important aspect, especially with respect to an FSI's IT cost. But it often takes considerable effort to work out realistic and applicable complexity measurements. Recent publications have shown that it is possible and certainly not trivial to measure IT complexity in a large FSI [Leukert et al. (2011)], and we would like to refer to some metrics they have worked out.

Leukert et al. (2011) relate business requirements to technical implementation using the three following parameters:

- Interface information flows (IIF): counts logical incoming and outgoing information flows of a business process/area (we use the expression "application domain") weighted by the required data latency.
- Interface intensity (II): counts all incoming and outgoing interfaces of an application, weighted by the implementation type (e.g. file transfer, webservice, direct database access, application specific API, etc.) and the level of data transformation quantified by the different data structures supported in the interface.
- Interface implementation type (IIT): mechanism to capture technical diversity. The greater the diversity of implementation types (File, View, message brokers, etc.) employed in an application, the more complex the application will be.

For simplification, we assume here that IIF stays constant over time (old requirements = new requirements), and discuss if a SOA implementation has effects on II and IIT. In addition to the aforementioned "II-parameters", some other aspects need to be considered.

Leukert et al. (2011) also propose the following technology parameters:

- Technology infrastructure products (TIP): measures the number of different infrastructure products/components, including, among others databases, operating system and applications server.
- Technology infrastructure services (TIS): counts the number of different infrastructure services.
- Technology infrastructure requirements (TIR): measures the number of different infrastructure service level requirements to be supported.

Some authors have tried to disentangle the term "complexity" [see Mocker (2009)], and point out that one can expect lower maintenance cost and/or reap cost benefits by reducing IT complexity in terms of "interdependency".[2]

So if mainly reduced interdependent complexity and improved "II" parameters in a given architecture enables TCO reduction, these would be most important metrics for transformation business cases.

We assume in the following chapters that necessary complexity leads not to unnecessary costs, but that there is a clear relationship between interdependency complexity [see Mocker (2009)] and TCO. So, if one reduces the number of unnecessary system interdependencies by a two-digit percentage, then the related TCO components are assumed to also decrease by a significant percentage.

Transformation business case

When evaluating a transformation business case, one needs to take current TCO_{old} ("old world"), transformation cost (including staff initial training, if not to be accounted/amortized in future TCO_{new}) and the TCO_{new} ("new world" after transformation) into account. The following TCO indicates a viable transformation business case:

TCO_{old} > transformation cost (annual amortization) + TCO_{new}

Often drivers for transformational changes are not just cost ratios, but also can be regulatory or strategic. FSIs should calculate the cost that comes with regulatory or strategic-driven changes on a wide perspective.

Only when TCO_{new} and transformation significantly outweigh the current TOC_{old}, is a transformation business case viable from an IT cost perspective. It's up to the program and senior management to ensure continuing viability.

Delivery frequency and latency

Last, but not least, delivery frequency and latency is a business metric. It is mentioned here only briefly, because proper delivery frequency and latency is more a requirement to be fulfilled rather than measured. We define delivery frequency as the specific intervals between two data deliveries. We define delivery latency as a required maximum time within a given delivery system that can postpone data delivery without harming a SLA.

Motives for transforming traditional data distribution topologies

Driven by business schedule

Although technical/regulatory requirements exist in abundance, most of the changes in IT are driven by business requirements. Often these requirements do not take into

2 "Interdependency" refers to the interconnectedness of the applications in terms of the interfaces they have with each other. Applications with more interfaces to other applications would be more interdependent than those with fewer interfaces [see Mocker (2009)].

consideration any technical dependency or relationship between delivering and consuming systems. They are mostly, by definition, detached from those systems. This fact proves a challenge to most system architects because they commonly do not want to "break things," i.e., restructure a proven technical process structure that works, albeit for a dated business requirement. However, despite the fact that such requirements need to be planned carefully, most architectural and technical projects are on tight schedules, pushing the developers and architects to complete quickly. So how should the tasks be scheduled to meet that challenge?

Data governance

Direct data distribution strategies based on mechanisms like extract, transform, and load (ETL) or the distribution of batch files have, in many respects, some major flaws. If direct linking strategies like these are used on data distribution systems one quickly encounters a lack of abstraction resulting in tight coupling of these systems – too close to maintain a single change on one system's data without also changing the other. They are not coherent, meaning that they are separated from other influences. Multiplying this by the needs of big infrastructures with hundreds of systems, could mean up to a couple thousand dependencies to maintain. A single change in major provided data streams such as for capital markets may impact a hundred or more channels. In addition, there is the governance and planning overhead. How can this governance-hydra be reduced?

De-duplication

Typical requirements are tight schedules paired with high data quality, low-cost maintenance, and reusability. Low cost maintenance needs reusability to function properly. The data needs to be divided in separate and coherent entities without having an impact on the scheduled costs. High data quality also needs data to be well-defined and free of duplicated entities, a task also not scheduled on a typical project track. How can this task of data separation and quality be assured to be on track?

Connectivity

In heterogeneous system landscapes, different consumers need very different connectivities. From flat files up to data streams or RESTful interfaces, every application can have its own options to connect to the delivering system. In a small architecture this would not pose a threat to the architecture, but in large-scale heterogeneous architectures how can this connectivity requirement be addressed effectively? More to the point, how can you implement an approach to counter the exponential growth of interfaces in this environment?

Business intelligence and accounting

Some years ago, vendors like Streambase, Progress Apama, and Bloomberg Wombat established complex event processing (CEP) as a new and fancy technique for capital markets players in search for an additional "ahead of the market advantage". But to implement CEP, a SOA-based infrastructure needs to be provided, as such CEP could be another business driver for implementing a full and "true" SOA environment. While it should be easier for smaller FSIs like hedge funds (speaking of IT infrastructure) to change the IT landscape, for FSIs with large heterogeneous IT infrastructures, a "clean" SOA seems necessary before implementing an integrated CEP solution. So could this transformation of architecture become a major ahead-of-the-market advantage?

Mission control

A tight schedule can be a tricky task when a lot of dependent systems are involved. If the systems architecture is based on a direct linking architecture such as extract transform load (ETL) or similar architectures, changes made to the data structure are rather difficult to achieve because every dependent system wants its very own data to consume. They are not interested in other dependencies or interferences. Could the responsible IT unit develop a general blueprint to achieve most or all the requirements and mission statements?

Common language

Developing such heterogeneous connectivity and data deliverance also needs to be addressed in a data translation manner. Every single consuming application will have its own format and data semantics, so a data delivery system is confronted with the many languages it must provide. A simple way is to force the consumers to understand "your" language. In this context XML provides a means but it is not an end unto itself as users will have to deal with the "extensible-ness" of the language structure between producer and consumer. But what to do if your language changes? Change all the consumers also? What if your language is too "coarse" or lacks context in order to cope with the effective depiction of business semantics?

Special event: transformation driven by M&A

The merger of disparate IT environments poses fresh challenges to enterprise managers:

- Merging or subsumption of different architectures
- Set up of new and adaption of old interfaces

Aforementioned reasons drive change usually under severe constraints of time and budget with a pressing need to retain existing business functions while somehow merging the underlying environment into a workable unit. The first question posed to the project management has got to be "How?" The answer should be: "Along a gridline of defined business functions."

A special case where the stringent application of SOA would rather be inadvisable is in the realm of systems requiring low latency such as algorithmic trading, complex error processing, etc. Here processing chains relying on the semantic protocol exchanges and such as WDSL would hamper the inherent need for high speed processing.

Ideas at Work

Transformation principles for FSI data architectures

On rather large data distribution sites, such as banks and other financial services institutions, many different types of data need to be delivered to various sites at varying times in many different formats used by all the different systems involved. The following principles should be reflected and taken into account whenever one FSI designs new target architecture specifically to set timelines and allocating budget for transformation projects and programs.

Regain of control

In one of the business case examples below, we depict a well-established but very heterogeneous IT infrastructure running various systems and applications. The main complexity results from the vast interconnections of these systems. We want to point out how to effectively reduce this complexity by using the bare principles of SOA combined with a simple TIER-architecture to separate and fulfill the heterogeneous needs.

In complex systems and IT landscapes, at some point it is inevitable that when an environment reaches a degree of complexity even those who have built it will not be able to handle it. This goes hand-in-hand with an increase of maintenance costs.

So what may be a reasonable solution for this problem on the architectural side? As far as the technical side is involved, established principles of software engineering take control: DRY (don't repeat yourself), SPOT (single point of truth), POLA (principle of least astonishment), and distribution. These are not groundbreaking new principles, these are not fancy, but proven for years [see Parnas (1972)].

Divide et impera

This first step to take is to horizontally divide your heterogeneous systems into blocks of the same concerns (see Figure 1). For example, if you have about five systems working with capital markets data, all of these would be grouped in concern aspect groups (CAGs). These groups on the business side represent a homogenous data group.

On reliable data infrastructures, the systems are tailored to the data they provide or are providing. For example, a major capital market data hub won't spread all data within the FSI, but just the data to dedicated data consumers. This reduces the overall data transported by the system and also keeps the maintenance simple.

Singularity

Another free-of-charge advantage of using separation of concerns is of course SPOT. As long as your systems infrastructure is clearly divided in concerns, there will be only one system with a distinctive concern (unique data), a single point of truth for the data it provides.

A major benefit of a SPOT system is the concrete control and governance responsibility for stakeholder data: the systems owner is also data steward, controlling the data models as well as inbound and outbound data flows. This will usually cut management and maintenance overhead induced by "foreign" system interdependencies.

For example, take asset data provision independent from stocks or derivatives. We are not talking about any data format aspects or technical transport protocols here, just separation of concerns as principle of DRY. A system, therefore, also may not provide another system's data, as this would interfere with the same principle. The system also does not provide mix-ups of data in its concern. This would also break DRY. Using the DRY principle as a side effect will render your systems coherent by default [Starke (2009)].

Feel the flow

The second simple but essential step is to vertically divide the now sorted systems to different layers of operation (see Figure 2). We will have systems and components that provide or receive data, we will have components that transfer data and we will have components that convert data from source to target data structures. These are all separate concerns. Based on DRY, we separate these concerns into different layers, each providing a unique service to the adjacent layers.

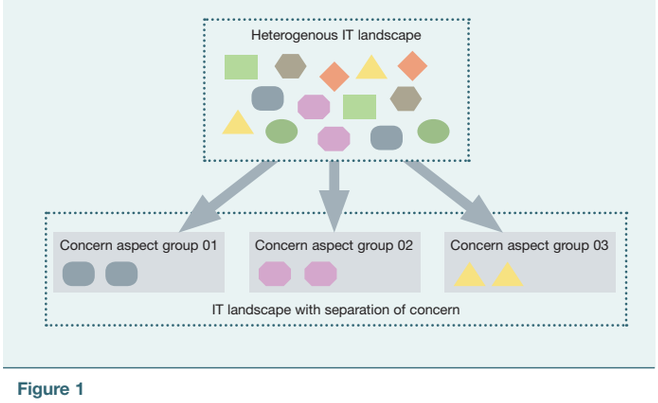

Figure 1

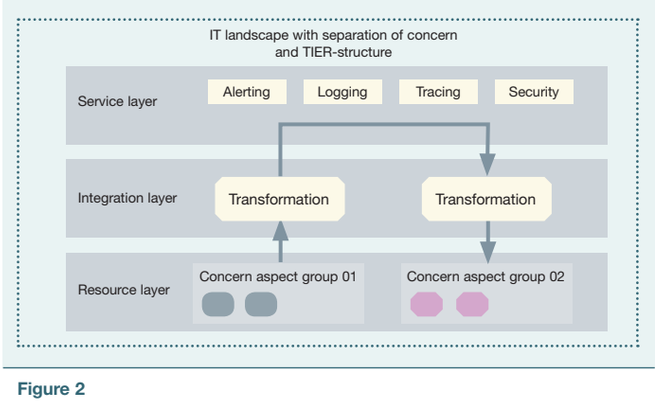

Figure 2

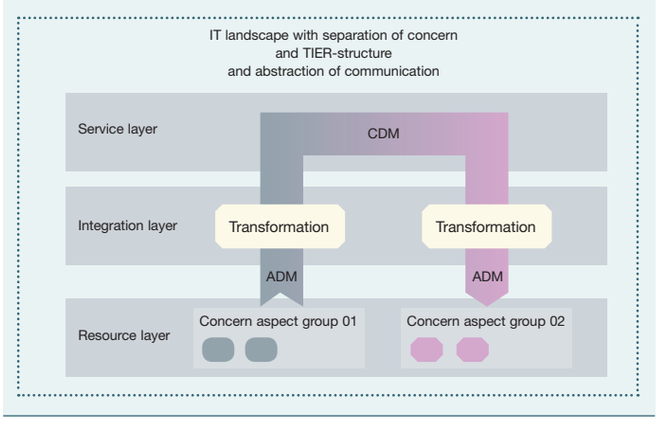

Figure 3

However, not only are the systems affected, but also the data transferred between the systems and therefore layers also need to be divided into different concerns. To take on the division of systems, we use one of the core principles of SOA: interface contracts.

Each system provides its very own data, the application data model (ADM), to the next layer based on a unique and well defined interface contract, the canonical data model (CDM). The CDM hereby is based on the POLA and is no longer application based, but business model driven.

It should be well defined and free of recurring specifications. Thus, the CDM, despite the ADM, is generalized and understandable company-wide. This abstraction implies the well-known SOA interface contract. Each new system will need to specify such a contract to the next adjacent layer (see Figure 3). This will provide each system connected to this service layer with a well-defined data pool [see Soeffky (1998)].

Adapter basing

This separation of source and target data formats provides the major benefit to tailor the subjacent architecture layout in a multi-TIER way. Each TIER and therefore concern can be handled separately – reducing complexity – and above all, are safely independent from each adjacent TIER.

A significant reduction of complexity is achievable on the source side. On this side only a source distinctive adapter is necessary. Its only purpose is the transformation of the already mentioned source ADM to the well-known CDM. This adapter is required only once, independent from how many target systems are or get connected. It only provides the necessary data in a well-defined CDM such as simple webservice. The transport channel here is irrelevant, thus only concentrating on the definition side, e.g., XML, HTTP, and SOAP. The current trending in service providing connectivity via "RESTful" service architectures will, on a mid- or long-term strategy, also help reduce complexity. Those RESTful services will replace a lot of old fashioned webservices, making them lean again.

While sticking at the source side with simple webservices, the real work needs to done at the target side. At the target side all the target system customization (e.g., transformation of data content, data structure, or file formats) takes place, i.e., transformation according to target systems requirements. Despite the high customization factor of these adapters, most of them can also be reused, as many systems use flat files or direct database connectivity.

Again we should not succumb to the charm of putting business logic into these adapters, because again we will trigger the trap of breaking DRY. If any significant business logic transformation is required, just add another layer or simple library – to ensure reusability and stick with the principle of loose coupling [Paoli et al. (2008)].

Decentralized system authority

But can we transfer such an idea to other necessary structures in an IT world? Managing a system is never easy. Managing several heterogeneous systems can be a nightmare. Managing several heterogeneous systems in parallel is a car crash waiting to happen. But why?

Taking into account that you might have about two hundred heterogeneous systems, how many interconnections could they have? If we take for example 200 by 200 connections between systems, then a centralized IT governance unit would have to manage about 40,000 data connections, without knowledge of any particular interface content or business logic.

Now let's imagine that every application unit manages and governs its own connections to other systems. This would result in about 200 connections per system to govern. And each system owners also know about particular data structures, interfaces, and necessary steps, e.g., in failure situations. Systems and data owners should be also considered as interface authorities.

Depending on the number and complexity of the resulting set of governing units, a straightforward system of governance needs to be established to coordinate and decide on matters pertaining to the interdependency of aforementioned units. Like its constituent parts, this coordinating body should be organized in as simple and effective way as possible. Relative to actual requirements, this could be an assembly deciding by quorum or a separate managerial unit.

Ideas at Work

Re-centralizing major benefits

But decentralization at all costs also could provide a major drawback. Every new system would have to implement its own meta-infrastructure like logging, security, and equivalents. So to also gain an advantage from decentralization, the major parts of this common functionality need to be provided and maintained centrally. In this way FSI-wide critical infrastructure like security, logging, tracing, and error handling could be provided to every single new system that wants to connect to the new infrastructure. New systems solely use these infrastructure parts, if not already provided automatically, by concrete API contracts. The maintenance and governance for these components will be provided by respective units, e.g., risk/security or architecture management [Gamma et al. (2001)]

Changes and M&A scenarios

So if this system ever changes or needs to be removed from the infrastructure, only the concern it provided must be taken into account. Hassles of managing cluttered dependencies in a technical or governance manner are obsolete. An IT landscape of coherent, independent, and easy to maintain systems and interconnections is achievable.

In an M&A scenario with at least one of the units possessing a viable SOA environment, integration or subsuming of business functions already aligned by service definitions and concerns would become much easier. Thus, the main focus could be on the integration and wrapping of constituent technical and functional components into SOA units before aligning these within the existing matrix. In the same manner functionality new to the SOA environment can be more easily added.

These are the necessary components which, suitably combined, will drive a successful DaaS implementation. These components can be applied individually in order to resolve separate architectural challenges, however, only in their complete application will they develop their fullest potential.

Benefits by SOA for FSI – business case and transformation evaluation

Based on the two following transformation examples, the third part of this paper will identify the major drivers in SOA implementation successes and failures. By depicting already undergone projects from the field of FSI architecture migrations, we will single out the positive and negative aspects, deriving this from the aforementioned SOA principles.

Incomplete architecture transformation – SOA not utilized

Starting point

With an IT landscape of more than 1,000 systems overall, a large bank decided to transform the current process to collect and process data for balance sheet reporting and bank controlling. The program's scope was to link a majority of core banking, savings, credit and approximately 20 investment banking systems to a new central data warehouse, and after data normalization, to feed an accounting target system. A strict waterfall approach was chosen with tightly scheduled phases along the normal paths of analysis and design, build, test, and deployment.

SPOT as architectural driver

A stringent architectural decision was the required adherence to SPOT. This stated a clear target data model closely synched with the target accounting system. The model incorporates a number of business schemas which all source systems were required to deliver against. Market and static requirements were slotted for central enhancement.

Although loose data coupling and the SPOT principle were a stated paradigm of the project, in practice a number of deficiencies were encountered. These involved the overly tight definition of the data structure with a number of source systems. In practice, this meant that a number of workarounds had to be found in order to arrive at a defined data quality. Said workarounds were rather difficult to arrive at due to the stringent target model and environment.

The underlying architecture environment was Informatica PowerCenter as the main ETL tool of choice. Requiring all source systems to deliver via strict flat file format and FTP connectors was the least-common-denominator approach chosen on data format and delivery as the technical spread of delivery systems ranged from EBCDIC-speaking mainframe systems to state-of-the-art trading systems.

Free hand was given to the source system owners to implement the required data feeds as pragmatically as possible. Considerable usage was made of the re-usage of existing feeds and feed mechanisms. The latter was especially useful for the more modern systems as these were usually equipped with standard reporting mechanisms.

Inadequate tooling

Another decision taken was to (re-)use the XML-based extract from one source system in order to satisfy the restrictive import design interface already laid out for the project. This meant an additional mapping layer from source system via XML into the flat file architecture of the target system. A problem with this was the additional model requirement of a complete transaction download which translated into a huge (+20GB) XML-extract file. This was extremely difficult for the parsing receiver to process.

Additionally, Informatica proved unable to loop XML structures rendering the traversal of tag structures to skip over irrelevant tag sections impossible. As the XML structure itself underwent frequent changes by the source system owners this meant that changes to the XML structure involved the time-costly process of agreeing to another interface definition. Since a part of the investment banking systems have been linked to other systems via message-based interfaces (MQ series), source system resistance towards ETL flat file transfer was significant.

Governance

The major stumbling block, however, proved to be the methodology chosen. Frequent changes and disagreements between source system owners and the project meant that any defined change to the content of the data feed involved a rather lengthy coordination cycle between delivery and target teams.

Although to achieve a high level of reuse was a stated goal of the program, analysis and design in earlier project stages was focused on flat file transfer design in a strict waterfall project approach. In addition to a tight project schedule, the latter made it hard to quickly establish any lessons learned and/or a feedback cycle to amend target architecture and project planning.

Future challenges

A future issue with the architectural target model chosen is the gap between the Phase 2/3 requirement of using the central data warehouse as a repository for operational intra-day reporting as this conflicts with the currently set end-of-day extractions from most of the source systems. This will require a major restructuring of downstream processing if ever the need for intraday or near-realtime feeds is to be realized. It should be noted that delivery frequency and latency is a business metric, and is to be fulfilled as such. A challenging requirement given the original design.

Checking back to the metrics mentioned earlier we find:

- TCO increase was unavoidable since a whole new data warehouse was built up including job-based ETL infrastructure to link up system. Several new interfaces had to be developed on source system side as well. All these new components incur a maintenance cost.
- Through stringently requiring feeds via a defined data structure of given content the projects IIF metrics to the minimum of one stream per product/business area holding the required information.
- By restricting itself to a few select processing systems the TIP count was reduced to the required minimum.

Inadequate SOA and business process implementation

Starting with an IT landscape with roughly about 500 systems interconnected by about 7,000 data transports, the goal of this project was to reduce the systems count by applying SPOT to identify and remove unnecessary systems and to reduce the used interconnection by implementing a standard enterprise service bus. Also the connection methods between systems and the data layouts using about 230,000 data entity fields (ADM/CDM) were to be standardized by migrating old structures to webservices.

Figure 4 depicts a typical initial systems situation in this migration scenario. Starting from various interconnections to other systems, this scenario also involves different data requirements, formats and layouts, driven by different consuming systems stakeholders.

The subsequent path to this scenario was to elaborate a successful migration plan to disjunct the targeted systems and leveraging the overall architecture by establishing a distinctive enterprise service bus (ESB), connecting all systems only to this bus. To archive this migration, the systems had to be split up and a canonical data environment had to be established. So this was an ambitious project and as every ambitious project it had flaws. Flaws that could have been prevented.

Failed scoping

Simultaneous to this system architecture migration, the complete infrastructure, e.g., security management, transportation, operations, administration, hardware and database infrastructure, was on the changing roll too. This meant that too many systems, not tailored by the already shown principles, were on the schedule to be migrated, all at the same time and not, as previously dictated, in separated, coherent, and therefore more manageable sets. This was one of a major cause of being not successful. On the strategic side of this migration, every system involved was on the test bed. This involved an analysis of the systems' usability and necessity in the FSI infrastructure, putting pressure on the systems' IT and business stakeholders (with the threat of being eliminated). The inevitable result of this allover change was met with considerable resistance by the respective business and IT units.

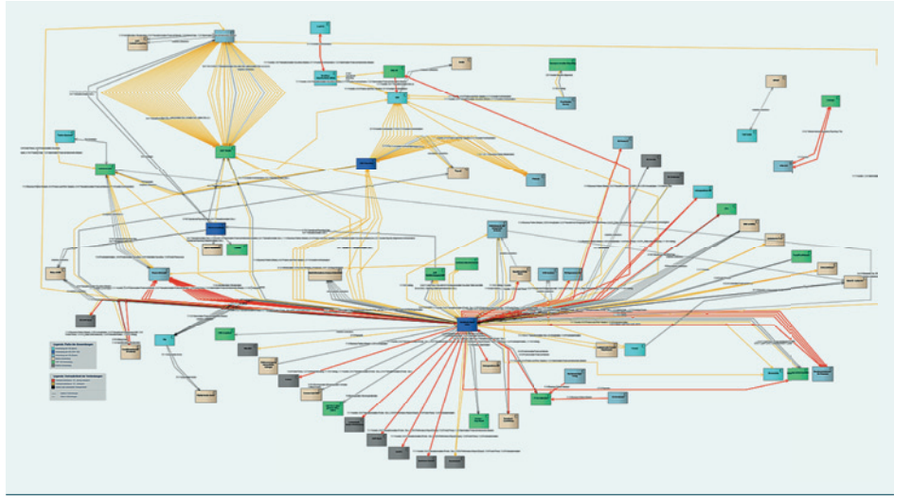

Figure 4

Premature infrastructure

On the architectural scope of the migration as already depicted, an ESB as central distribution and transport layer was implemented. Though implementation of the ESB itself was successful, the necessary infrastructure and concepts for failover, logging, error handling, and load balancing was not given beforehand. Even necessary shared libraries for logging, security, etc., were not available or planned. Instead of providing this central infrastructure separate (budget-wise) from the ongoing migration projects, this lack of planning was put on the project's payroll without adjusting the project scope itself. This resulted in project overtime and over-budget situations leading to conflicting issues between units.

The lack of a stable development and release infrastructure also led to conflicting situations: those systems had not reflected the real productive situations like failover scenarios, connectivity or security implementations. This inflicted massive time and budget issues because every new project needed to solve the same problems with release patches and system adaptions over and over again.

Halfway governance

Like every major FSI project, a whole lot of stakeholders from different systems and/or business units are involved that need to be managed, coordinated, and most importantly moderated. At an early stage, the whole migration project was tailored by programs, containing business units, and/or business interests such as capital markets or master data and their respective stakeholders. Underneath this global planning infrastructure, each system was tied to a team of business unit and IT. This teaming of units was not, regarding the governance, successfully implemented. Each team claimed the lead of governance, system and data wise, leading to confusion and even destructive behavior in the unit itself and even towards other stakeholders. This simple but very stressful and money-burning behavior could have been prevented by conferring the governance to either team, and not letting other stakeholders interfere with that governance.

Misplaced POLA

The lack of centrally maintained enterprise architecture management systems (EAM) and thus the missing centralized repository of available or developed data infrastructures (CDM) and services (DaaS) led to false or duplicated concepts developments. This in turn resulted in unclear or, even worse, duplicate responsibilities of ownership of the CDM structures by different units. This finally led to conflicts between different stakeholders delaying or finally stopping whole projects.

Checking back to the metrics mentioned earlier we find:

- In this case, TCO decrease was clearly intended, i.e., by consolidating legacy interfaces in standard services for different asset classes. By simplifying the historically evolved entanglement of system linkages, etc. (all three "II" parameters), reduction of complexity [see Mocker (2009)] was one of the main targets.
- To date, this has only been partially achieved as the aforementioned "II-parameters" have improved during the first project stages – but not as much as expected. Due to unclear governance and missed project goals to reduce complexity on given systems connectivity, most of the legacy interfaces are still productive.

Contribution of SOA principles to given architectures

In the following, we sketch how by combination of aforementioned principles, a good DaaS implementation can be achieved on nearly any given legacy infrastructure.

Reduction of complexity

Since this is a significant driver, any architecture transformation should aim to reduce system interdependency and thus as benefit after decommissioning legacy systems and interfaces, maintenance cost reduction. After this gained reduction of complexity the most effort should be made to keep these architecture tight and reduced at all times. At all times, systems should be evaluated based on the principle of separated concerns. This will keep architectures clean from duplicated sources, thus reducing maintenance.

Clean data contracts

When implementing a new SOA/DaaS infrastructures the main part on the "hands on" side is about keeping system interdependencies, by means of data contracts, clean. All legacy and the new data structures should be analyzed thoroughly and not jumping to seemingly easy workarounds. This analytical effort should also be made centrally available for all other implementers by means of an EAM system. The EAM should be implemented from ground up at the very beginning of every architectural or business driven change. This system inherits all changes and versions over time and is the central repository and data register for both business analysts and developers in equal measure.

Data tailoring

The data itself can be tailored by business segments and be maintained by the respective business units. Contracts to other systems and the governance of the respective data structures should be also maintained by those units. This also includes not bloating services with different versions, but rather consolidating each and every service as soon as possible. Sticking to these clean and straight architecture baselines, no workarounds should be allowed.

The data structures themselves should be tailored at a sufficient granularity respecting business or technically driven (Meta data) entities. Those entities should be reusable and common throughout the whole company.

Culture changes and communication

On existing legacy structures a migration and implementation to SOA/DaaS principles can at

best be achieved by influencing the principles how new projects are done. The most successful change can be made when the project members are living the principles we depicted here. Despite new technical or architectural patterns, to communicate and transport those principles is crucial.

Previously established principles should be put to the test evaluating their usability and leverage potential in FSIs. Often externally established principles need to be transferred to the customer, as does the implementation of new ideas and concepts. As these ideas often meet with resistance a leveraged approach is to set up a community of stakeholders that champion your causes, ideas and concepts.

Conclusion

As stated, setting up proper business metrics should be starting point of any transformation planning. Metrics to challenge initial plans, and to strengthen the basis for project stage budget provision and controlling.

We have illustrated the reasons for transforming topologies and described the principles one should reflect when designing target architectures. We have followed this up with transformation examples, to reflect the stated principles and the benefit potential of SOA against recent transformations in practice.

Since "one size fits all" detail architecture blueprints do not exist and never will, a good SOA transformation is much about implementing good architecture principles. Principles to be aware of, to evaluate, to adapt to given FSI architecture – but sometimes to ignore, if not applicable. Stay pragmatic, not dogmatic.

Never abandon business principles, like putting adequate effort in an initial business case evaluation. To assure that transformation benefits get reaped, one should extent an often just work package-focused earned value controlling. How much benefits achieved with overall actual transformation cost?

Legacy business processes and an enterprises' culture need to be reflected early, since it is often strong stakeholder resistance that put FSI transformations success more in danger than all aforementioned business or technical aspects.

References

- Gamma, E., R. Helm, R. Johnson, and J. Vlissides 2001, Entwurfsmuster – Elemente wiederverwendbarer objektorientierter Software, Munich: Addison-Wesley
- Leukert, P., B. Alliet, A. Vollmer, and M. Reeves, 2011 "IT Complexity metrics – How do you measure up?," The Capco Institute Journal of Financial Transformation, 34: 11-15
- Mocker, M., 2009 "What Is Complex About 273 Applications? Untangling Application Architecture Complexity in a Case of European Investment Banking," Proceedings from the 42nd Hawaii International Conference on Systems Science, IEEE Computer Society, 1-14
- Parnas, D. L., 1972, "On the Criteria To Be Used in Decomposing Systems into Modules," Communications of the ACM, 15(12): 1053-1058
- Paoli, H., C. Holtmann, S. Stathel, O. Zeitnitz, and M. Jakobi, 2008, SOA in the Financial Industry – Technology Impact in Companies Practice, in Seese, D., C Weinhardt, and F. Schlottmann (eds), Handbook on Information Technology in Finance, Berlin and Heidelberg: Springer, 9-28
- Soeffky, Manfred, 1998, "Data Warehouse: Prozess- und Systemmanagement," IT Research, Höhenkirchen
- Starke, G., 2009, Effektive Softwarearchitekturen – Ein praktischer Leitfaden, Munich: Carl Hanser Verlag
- Timm, I. J. and T. Scholz, 2008 "Towards Reliable SOA – An Architecture for Quality Management of Web Services," in Meersman, R. T. Zahir, and P. Herrero (eds), On the Move to Meaningful Internet Systems: OTM 2008 Workshops, Berlin and Heidelberg: Springer, 170-179
- Wild, M. and S. Herges, 2000; "Total Cost of Ownership (TCO) – Ein Überblick," Working Paper, Mainz University

Individual Selected Article from Journal 36

Layout, production and coordination: Cypres – Daniel Brandt, Kris Van de Vijver and Pieter Vereertbrugghen
Graphic design: Buro Proper – Bob Goor
Photographs: Buro Proper - Bob Goor

Journal
The Capco Institute Journal of Financial Transformation

CAPCO

Amsterdam
Antwerp
Bangalore
Bratislava
Chicago
Düsseldorf
Frankfurt
Geneva
Johannesburg
London
New York
Orlando
Paris
San Francisco
Toronto
Washington, D.C.
Zurich

CAPCO.COM