BearingPoint. Management & Technology Consultants

Insight |

Positions on Service Oriented Architecture, Part 2

Service Oriented Architecture (SOA) has reached the peak of the hype surrounding it and is now reaching the level of maturity. Nevertheless, there is still a need to describe positions and approaches for the implementation of the SOA paradigm and – even more important – the methodology. This document, based on a number of blog entries I have written over the last several years, describes a number of these positions. Statements made here are based on real life experiences and my reflections on questions and discussions I have had.

Contents

Introduction

The publication of the first position paper in 2009 resulted in a number of interesting discussions with former colleagues and new contacts. I was surprised and encouraged by this result. For this year's paper, I have selected some topics that, while not necessarily related to my daily work, that I am convinced are very important. However, they are often overlooked. In this paper I describe my position on:

- Business Cases and SOA. This area is the result of observations and discussions I had with some clients who were trying to introduce the concept of Service Orientation into their environment, company or organization. Very often this ended in frustration because the discussion moved into an area with which the technologist or architect did not feel comfortable.

- Governance and some different facets of this topic. The governance topic is very wide and a white paper is too short to completely cover all the areas. But some of the aspects discussed here have been shown to be quite essential in daily work.

- SOA and Security. I was quite surprised when I wrote this at how much my interest in history and events from times long gone still have meaning in today's IT world. Even more interesting were the faces of my clients when I spoke to them about the topic.

- Central Data Management. This is a topic I've stumbled across three different times in the last year. I am not convinced that is going to be a bigger trend in the coming year, but I found it exciting and interesting.

As with every position paper, the above areas are places to start a discussion and to get in contact with you – the reader. As in last year's paper, some of the positions you may agree with, some of them you might want to discuss further. Please feel cordially invited to do so. I look forward to hearing from you.

Position 7: Make me a Business Case for SOA

This is a scenario most technical people dread. They leave a meeting or the office of the Project Manager with this task: “Make me a business case for the use of SOA in our project - and then we will decide whether we want to use the technology.” Typically, the technologist really doesn’t know where to begin building a business case.

The arguments for a technology are mostly technical and the benefits of a particular technology might only materialize after the project is completed. But when it comes to SOA the case is even more complex.

Service Oriented Architecture (SOA) is - contrary to common belief - not a “buzz word” phrase that was generated to have a particular sales pitch. It was also not designed to simply impress the market with a new trend. Like quite a number of developments in the integration and software development area, it represents a stage of thinking.

Service Oriented Architecture is an evolution, not an invention.

From the first days in EAI, architects of the first implementations of integrated environments reviewed their work, and identified where they could have improved their implementations. Like all new technologies, EAI implementations, in the beginning, lurched from trial to error and back again, in the implementation process. In retrospect, the implementers reviewed what they did and found that quite a few things could have been done better, would have resulted in more stability. And very importantly – there could have been less pain on the project management side of things.

So, they reviewed the different approaches, looked around in the IT world and collected what they found to be best practices in the implementation of their IT systems. Some discoveries included things like canonical models, service definitions and catalogues, among others. They gave these kinds of implementations a new name to clearly identify the improved methodology: Service Oriented Architecture or SOA.

Now, return to the poor technologist standing in front of the office thinking about the business case for SOA. He knows that although he has been tasked to *write* a business case for a methodology, actually he has been asked to write down “Why should we use this best practice?” Of course, a best practice should be used as the alternative to the use of something suboptimal.

So, where does he go from here?

First, the choice management wants to make is not about SOA - or another architectural solution. Management is interested in how to implement a solution in the most cost effective manner. Should they use a central application which contains all the functionality - or a distributed implementation which uses already existing capabilities in the environment?

The technologist needs to address this interest by explaining that the use of best practices like SOA reduces project risk and has an impact on the project baseline. He needs to **not** discuss too much of the deep down mechanisms of services themselves. His message needs to be: If the decision is made to integrate a system, the introduction of services and proven methodologies has been shown to reduce the risk in the implementation of integrated systems.

Second, introducing new technologies (and the related buzz words) is exciting for the technologist, but it is a horror for the typical manager. New technologies are notoriously unreliable, risky, and they always cost more than initially expected. They are only a win for the manager if they exceed expectations and the manager can present himself as a successful visionary. He wants to look good to the company’s board of directors and wants to be invited to conferences to talk about his success.

With this in mind, it is very important for the technologist to not “sell” SOA as a great technology solution. He should position it as a **requirement for success**. If you do not use best practices in the integration world - it will be much more difficult to be successful in the project.

Third, there is a good chance that the technologist got tasked with writing the business case because the manager is overwhelmed with new ideas and needs to find out about this SOA thing before making a decision.

One good approach is to introduce the different components of SOA one-by-one. Some examples of this might be the data model, the use of end-to-end processes and the re-use of functionality and processes. Technical tools and procedures - such as monitoring and alarming, and interface standards can also be used to explain why this particular approach is useful. This allows the manager to step away from the big acronym “SOA” to something that has a real impact on his business and his needs. Coincidentally, it also helps implement the tools the technologist needs to successfully complete his project.

Finally, the introduction of a methodology is often used as an excuse to spend a great deal of money on tooling. It is important to set expectations correctly and alleviate a manager’s concerns on this issue.

When introducing SOA it is not necessary to replace the integration platform with a new and fancy service bus (which is barely used as there are only a few services implemented). If the system needs a new service bus for legitimate reasons, i.e. the performance of the existing infrastructure is bad, the currently used platform is out of support, interfaces do not exist and I can buy them cheaper than building them – it might be justified. (There may also be a question as to whether or not an integration platform is even the right approach.) But, these investments are independent from the concept of SOA and they have their own justification.

Conclusion

Using a concept or methodology to justify investments is very risky, because (at some point) the stakeholder wants to see a "return on investment" (ROI). If the technologist suggests investing in a technical tool, then he must concretely demonstrate the business benefit. If a tool can be used for many different reasons, then this is an additional plus marks. Always make sure that the introduction of a best practice like SOA is not seen as a smokescreen to introduce other technical investments.

To introduce a technology into a project, or into an environment, it is very important to build a vision with management. Find a path of small, reasonable, and understandable steps which can be explained to the business owners that shows the thinking behind your proposal. Show the business stories where these steps have proven practical, successful and have saved costs in one way or the other.

At the end - best practices sell themselves - whether they are given a particular name or not. A business case for SOA does not exist - but there is always a business case for doing things right.

Position 8: SOA Governance

Sometimes readers raise questions about topics. Topics that I thought I was thoroughly familiar with, but lead to interesting and surprising discoveries. As a result of the first paper, I was asked to write a position statement about the governance of service-based integrations. “Not a problem.” was my first thought - followed with the painful realization that I really had not spent enough time thinking about it before the request. Governance is something you **do** - and the knowledge **how** to do it comes out of the experiences and work you have done in the past.

I started in IT working on monolithic COBOL systems. I did the first object projects C++ based and learned CORBA - and ended up in the integration world ten years ago. The approach to governance followed from technology to technology - not changing too much in approach and execution. So, this might be a good time to think about governance from a service point of view.

Wikipedia describes the word “governance” as the “consistent management, cohesive policies, processes and decision-rights for a given area of responsibility”.

An integrated environment needs governance in two areas: the design and implementation of the system and the operation of services on the other. To completely describe the needs for governance we have to consider both sides. To accommodate that need, this topic is divided into two parts: governance in design, and implementation and governance in operations of service-oriented systems.

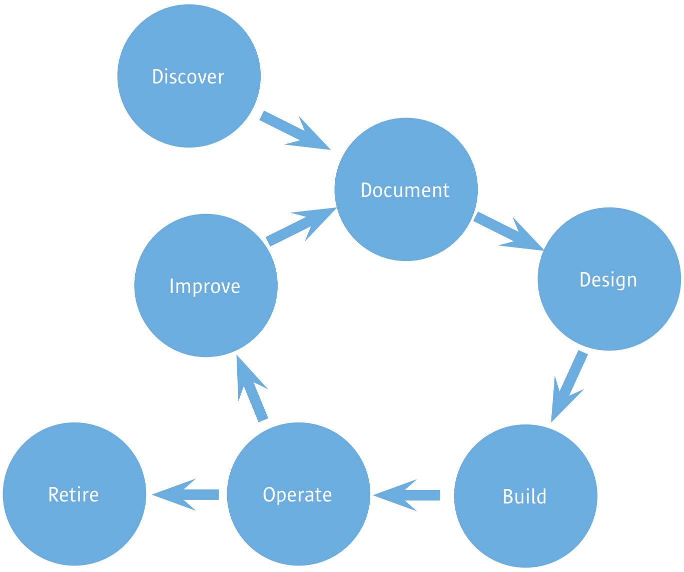


Figure 1: Service Lifecycle

Design Governance

It is possible to have an entire career based on the topic of “design governance”. (It should and will be in a paper on its own in the future.) But as part of this white paper, I can summarize a number of different aspects – those I believe to be of interest and importance.

If we compare the approach of service-oriented architectures with the EAI integration approach, we see that some of the advantages of the use of services over previous integration approaches are the introduction of technical standards and norms.

Many organizations introduced web services and WSDL as documentation and definition standards, replacing and streamlining the original jungle of adapters and integration techniques, with more lean use of rules. The use of a central WSDL standard supports the benefits of technical services and the reuse/synergy effect of the architecture. However, professionals often feel that SOA development processes do not really differ from the earlier EAI development processes and the differences lies only in the tools and in the application of lessons learned.

But, there is a fundamental difference between SOA approaches and the previous integration approaches. And that is - the point in time when the design process of services as an architecture concept starts.

Typically, services are found much earlier in the design process often during the definition of the business processes themselves. This is much earlier than in earlier integration projects - when the technical and integration team got involved after the processes were already defined and finalized. As a consequence, the governance of the service development starts earlier – during the discussion of the business processes and their (non IT-based) initial implementation.

The key topic at that point in the development process is the implementation of a service catalogue, a key difference from EAI integrations.

The service catalogue needs to be owned by a specific role in the organization. For our purpose, we will call them a “Service Manager”. This role can be handled by a single person, but is more appropriately located in a Centre of Excellence. The Service Manager is a central point of contact who governs the services implemented in the organization. The manager, however, is not the owner. Ownership of the services is with a functional or technical system or department lead.

The Service Manager is not limited to IT services, but supports all services in the organization. As a result, the Manager is responsible for the correct design and implementation of the service, coordinating the operational side as well as the design.

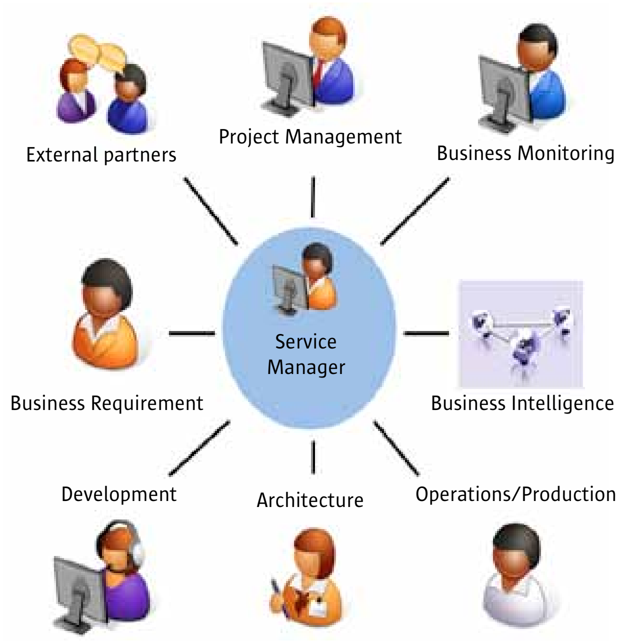


Figure 2: Service Managers position in an IT organization

Service Catalogue

All services discovered during the definition of the business process need to be catalogued. This allows the identification of not only the IT-based services that are under the control of the business owner, but also the manual, human services provided in the business process. More importantly, it is important to remember that only when the business services are inserted in the service catalogue do they become truly re-useable and understandable for the business owners.

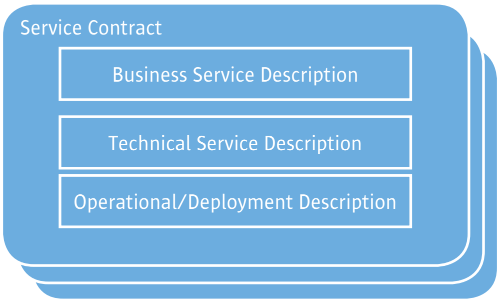

Service Contract
Business Service Description
Technical Service Description
Operational/Deployment Description

Figure 3: Service Contract Basis: Business, Technology, and Deployment

The service catalogue is not only a list of services that exist and can be used. It is not just a list of service contracts. The service catalogue is much more. It covers the service life cycle, it describes the discovery of the service during the review of the business process, the design of the service during the implementation phase and the change processes in the mature stages of the service.

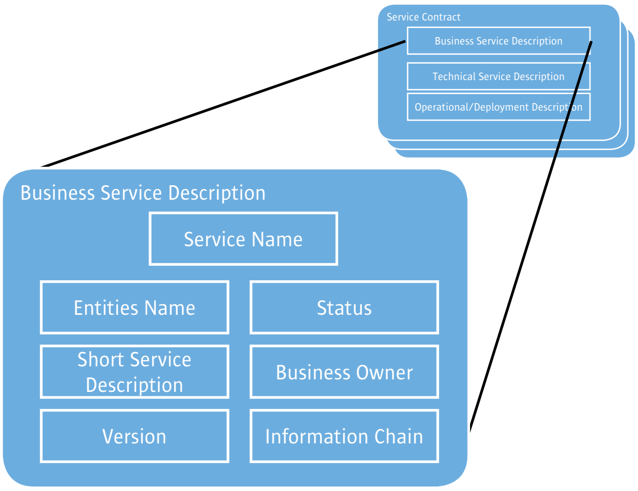

Service Contract
Business Service Description
Technical Service Description
Operational/Deployment Description
Business Service Description
Service Name
Entities Name
Status
Short Service Description
Business Owner
Version
Information Chain

Figure 4: Service Contract: Business Services Governance

The following information is essential for the service catalogue:

Business Service Governance

The service name. The service name should be a unique identifier of the service. There should also be a descriptive three-word description of the content of the service. One thing to avoid is just assigning a number to the service. This has a tendency to obscure or hide the real nature of the service – making re-use less likely.

Entity names. The entity names describe the entities the service handles. Every service implements a function on one or more business entities. To retrieve an overview on all services available to the entities, it is necessary to enter the business entities in the catalogue entry.

Short service description. A description that allows the reader to identify the content, and the consequences, of the service on a functional and technical level is important. This requires more than one sentence descriptions. However, descriptions which are essentially a description of the full design are also not useful as they stop the reader from seeing the important information on the nature of the service. As a rule, the service description should not take longer to read than one (1) minute. The reader should be able to retrieve all important information in a glance and within that timeframe.

Version. All services have a life cycle, and therefore, a history. Some services are alive with different versions at the same time (that is not ideal or wanted) - but sometimes can't be avoided. Therefore, the version of the service (which should be contained in the documentation) needs to be mentioned.

Status. Every service possesses a "state" in the lifecycle. For the consumer or user of the service, it is important to understand the current state of the service. Possible states could be:

- Discovered. While working on the business process, the process designer has discovered the need for the service and written the initial description and contract.

- Designed. The service has not only been described, but the process inside the service has been described and the functionality has been approved by the owner of the service.

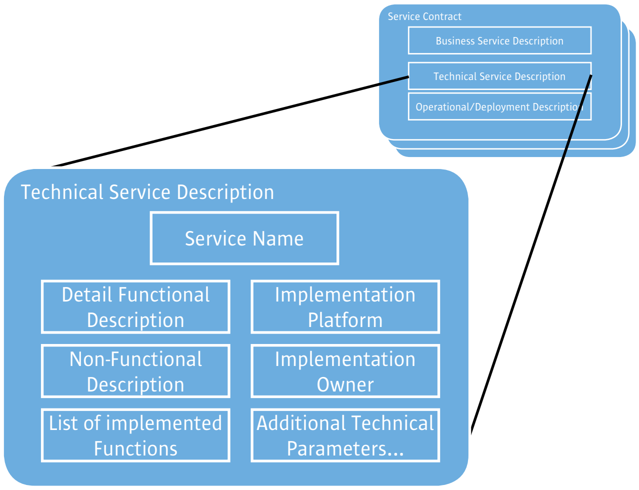
- Developed. Development of the service has been completed, but the service has not been tested.

- Tested. The service has been tested and is currently available, but no consumer is currently using it.

- Productive. The service is used by at least one consumer.

- Retired. The service is obsolete and should not be used any longer. This stage should be used to allow access to the service history.

Technical Governance



Service Contract
Business Service Description
Technical Service Description
Operational/Deployment Descriptio
Technical Service Description
Service Name
Detail Functional Description
Implementation Platform
Non-Functional Description
Implementation Owner
List of implemented Functions
Additional Technical Parameters...

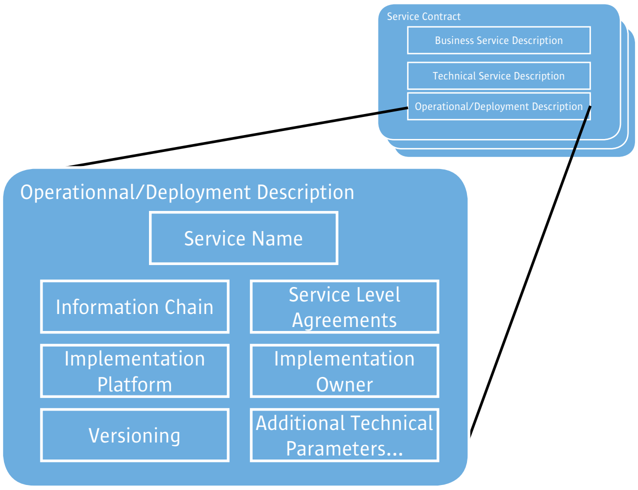
Figure 5: Service Contract: Technical Governance

Implementation platform. This information describes the platform that holds the services, which could also be a human or a department. It does not have to be an IT system and this is not solely technical information.

List of implemented service functions. A service must contain one or more service functions. These functions are named and described briefly in this part of the catalogue

Ownership (functional, technical, business). Every service needs to be owned. The owner is the sole authority over the service and is responsible for the service contract. The owner is also responsible for work on any issues related to the service. If the service ownership is not clear, it is important to discuss and agree on the existence of the service. For good governance, the rule should be that services without an owner need to be removed from the catalogue.

Operational Governance



Service Contract
Business Service Description
Technical Service Description
Operational/Deployment Description
Operationnal/Deployment Description
Service Name
Information Chain
Service Level Agreements
Implementation Platform
Implementation Owner
Versioning
Additional Technical Parameters...

Figure 6: Service Contract: Operational Governance

Information chain. If changes to the service's design or description need to be made, these are the individuals who should be notified. Changes may be planned or already implemented. This list usually consists of the owner and the consumer of the service, but might also contain the operational staff of the productive environment.

Service level agreements (SLAs). This is business and operational information, but it is information that is important for a potential consumer and the parties on the service contract.

Functional description. A full functional description of the service needs to be included.

Non-functional requirements (security, quality of service). These elements are very often forgotten or ignored. The load on the service is an important criterion for the consumer and also for the designer and developer. These technical criteria should be defined before the service is designed, so that the requirements become part of the service's characteristics.

This is not a mandatory or complete list. It is a starting point for the Service Manager (other organizations may call them a "service broker" or even "service policeman") to be able to build and maintain the catalogue.

Service catalogues need to be adjusted and connected with the user and author of the contracts in the catalogue. More content can be added, if it is deemed important for the environment. An example of this might be the test cases created for the service to verify its functionality. It is most important to gain a consensual understanding between the different involved parties on the content, and the responsibilities, of the different sections. As in all tools in the IT development realm, a service contract only works if it is adopted by all parties.

Standards and Rules

The second important pillar of service governance, in the design and implementation domain, is the introduction of standards and rules. These are required to allow the consumer to utilize the service with the use of its available documentation.

Most of the benefits of SOA are, unfortunately, lost in this part of the implementation. Most cases of problem implementations indicate that most (if not all) services in these business domains are hiding point-to-point connections.

Amazingly, many users of service technology use only the technical features of the services. They still implement the services as connections between two systems, having the business rules of a single connection, with a single consumer dictating the interface. A particular connection in the process defines the interfaces and is used as a reason to build the service. These definitions are so special and individualized that the use of the service by a different consumer still requires a significant level of customization of the interfaces. This approach does not lead to a loosely coupled environment, but a hard-wired connection with the use of web-services technology.

Standardization of Interface Technologies within the applications

Another area of governance related to the technical implementation has the origin already set from the time of EAI. Reviewing integration environments with limited rule sets (and this also is true for services), it is possible to find the same kind of connector (sometimes even to the same application with similar requirements) using totally different technologies. Sometimes, web-services are used for one connection and database adapters used in another. There may even be different standards and rules used for the same web service. The maintenance of this type of integration is costly, difficult and, needless to say, a lot of the benefits of the technologies are lost.

How do rules and standards help here?

Enforcing a well-defined and agreed rule set allows the designers and developers to build services in a way that supports the objectives of service orientation. These rules can start during the discovery of the services and can govern the features and nature of the service. Such a rule could describe (for example):

- When it is acceptable to call an exposed function of a service.

- When to use synchronous and when to use asynchronous services.

- What to do when a service fails.

These are fundamental rules that help the business process designer to declare the discovery of the services, and also to model them in a consistent manner.

During the design, a different set of rules are required. These rules contain more content-driven agreements. Some of these could contains rules such as

- Entities are always transferred in total, not in subsets.

- Services always return the result of the processing in the same manner.

Very technical rules, such as error handling and interface structures, can also be defined at this point, as well.

Interestingly, most rules defined in organizations using integration methodologies (such as SOA), cover the technical aspects adequately. This is usually because experienced developers understand rules and the need to develop and implement rules for their work. But these limited rule sets are, in many cases, insufficient for a proper governance of services in the development process.

Having rules and governance defined in **all** steps of the service lifecycle helps to decouple systems. Only in a decoupled environment is it possible to utilize all the benefits of a service-oriented architecture.

Conclusion

The topic of design governance can fill entire books. However, I would summarize good governance with the following words:

Good governance is based on good processes and good rule sets. The governance of the services starts with the discovery of the services, and it progresses until they are finally retired. Governance includes ensuring that a well-defined and described set of rules and standards, to describe the way services are built and deployed exists. Governance also includes the definition, use, and maintenance of a service catalogue, owned by a Service Manager. This catalogue describes and documents the available services and ensures that the consumer receives all functionality from the service he can expect - based on the catalogue.

Operational Governance

The idea of governance of services does not stop with the implementation of services in a production environment. When it comes to operations, a different kind of oversight is required for the services. The following section summarizes a number of tasks important for the health of the services, and therefore, the happiness of the consumers.

Moving services from the design to the operational use means making the service discoverable to the outside world. As part of service and project governance, the deployment procedures the service needs should be made formally available. This can be done by using a service registry. The service registry can be used in several ways. Service management, versioning, and monitoring are but a few. When the service catalogue is incorporated in the implementation, it can provide designers with important information on the services. Some of that information could be - the use (acceptance) of the services implemented, the volume of data being run through the services, and up- and down-times.

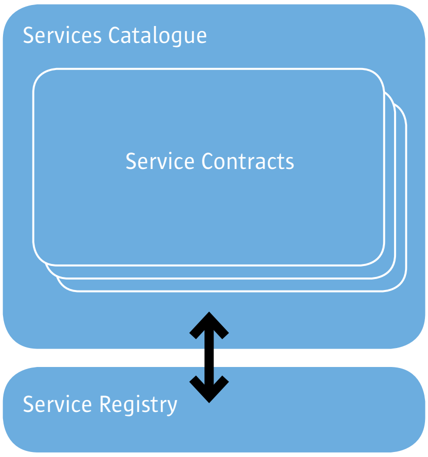


Figure 7: Service Catalogue and Registry Link

Monitoring of services

Many services live a quiet life. Once they are in production they are not seen or mentioned again - except for two scenarios. In scenario one, the service stops performing. In scenario two, the service needs to be adopted. While the second case is not something anybody is worried about, the sudden absence of the service is definitely something to be concerned about. Especially if the service level agreement in the service contract requires the service to be active within certain parameters. I will concentrate on one particular issue of the monitoring of services - the heartbeat. There are other aspects, but I have found this to be extremely useful and missing in many other sources in the integration world.

As part of an operational integrated environment, it is important to have the heartbeat of the service monitored. This monitoring not only verifies that the service is available but that it should not exceed critical limits or response times, memory usage, or a number of spill-off processes. A single 'pinging' of the service is not enough; the service needs to send a "health message" to the monitor.

This monitor messaging is not implemented very often – for two reasons. First, the architect/designer fears the network traffic that might be incurred during the heartbeat process. Second, because this infrastructure measure does not provide immediate functionality (and therefore costs money) – it cannot necessarily covered by the business case. On the other hand, it is helpful to consider the following: Our modern networks are much bigger today than they were even three years ago. Most networks internally are 100 times faster nowadays than just a few years ago. The additional load of a heartbeat/health message is limited - a few bytes if being carried on a big, glass fibre network. If the heartbeat is sent every couple of minutes, it does not affect the network load at all, but can save a sleepless night when the service is suddenly down. The heartbeat should be implemented in the service framework as a feature and routine that the local developer should provide to every service and interface in the environment. The costs are then covered by the roll-out of the integration, and split between the different business cases, at the end. They can then end up paying for themselves.

Do not operate – manage service

As services have a lifecycle, just "operating" them is not enough. As the ITIL software lifecycle describes, the life of the service does not end with moving it to production. The owner has the additional responsibility to constantly improve and refine the service.

Improvements can be two-fold:

- Technical improvements. Reducing error situations, raising the availability and performance, and adding supportive functionality to the service (like human interfaces for pure IT services)
- Functional improvements. Adding functionality or closing gaps in the existing interfaces.

Most critical is the need to keep the service "on the radar". The owner needs to know the service and needs to be informed on its use and overall performance.

Services which are not used need to be retired.

Over time a service can become a commodity – like a report on a legacy system. Nobody knows why it is around, nobody really needs it, but as long as nobody questions it – it will remain in the environment. Services can end up in the predicament. They may not be utilized or changed, but they are still using resources - and even maintenance, if the underlying system is changed. Good governance and management means: retire those unused services to reduce the costs in the environment.

What-if Scenarios

One area of interest are “what-if” scenarios. This area could be much expanded upon, but this document will only look into two.

- What happens if the service is not available? This scenario can mean real disaster - transactions and business processes get “stuck”; the critical path for the process is blocked, and the operation comes to a standstill. This can happen to the most important business processes in the organization, causing real damage to the financial situation. The what-if approach here outlines the development of contingency plans, including the development of alternative business processes. Most important, however, is the development of an alarm process in which the impacted instances (departments and persons inside the impacted business and IT) are informed and advised.

- What happens if the transaction in the process fails? This is not only a design issue - even if the responsibility for the design of the rainy-day scenario is always in the design group. Most business processes are so complex that not all process flows can be foreseen, and not all exceptions are frequent enough to implement an automated error handling. So, if the process fails on a small number of transactions, it becomes important that the operational support group knows how to handle the situation.

Good governance on the operational side means being prepared for these and other scenarios. To not only handle them - but to prepare for them. Building guidelines, alarm processes, and most important, being informed and trained in the handling of the services, is an important part of the operational governance of a service-oriented environment. Any Centre of Excellence or SOA architecture group should spend time to prepare a complete list of what-if scenarios, with contingency plans outlined, for all parties using the services.

There’s always room for improvement…

This sounds like a no-brainer. But most services are accepted as they are and not looked at closely until they fail. To proactively handle the services, it is important that the improvement of services is an on-going target of the Service Manager. He needs to be supported by the team responsible for the operations of the environment. This includes the ability of the Service Manager to introduce improvements without project or business case. Budget needs to be allocated, if the services are to be kept on the current, functioning and effective.

Conclusion

Governance does not stop with the implementation of the services. They need to be governed during their entire lifecycle - especially when being active and life. This governance requires budget and dedicated work force. Processes to manage failure and disasters need to be considered, designed, built and the system monitored.

Finally, unused services should be retired. Only active and used services are an asset to an organization. Only by monitoring and managing the services it is possible to identify potential candidates for this last step in their lifecycle and - maybe by promoting them – finding a way to keep them alive.

Position 9: SOA and Security – The second Great Wall of China

Between the years 220 to 206 B.C., the first Emperor of China, Qin Shi Huang, decided to protect his lands by providing a wall. He linked existing walls and protections in the country to create one massive, protective wall. This wall would become world famous and (after being expanded by the Ming Dynasty) would become the "Great Wall of China". The idea was genius. If the intruders run up against a heavy big wall, they will not threaten the country. The idea did work, until the Manchu attacked. While the army was protecting the wall at the Shanhaiguan pass, the Chinese general Wu Sangui (who was opposing the royal house), opened the gates to the enemy. Once the gate was broken, there was no stopping the flood of enemy troops and the capital fell in just a few weeks. Relying on the fortification of the wall, the emperors did not provide enough defence capabilities in the rest of the country to stop the Manchu army.

Figure 8: Great Wall of China (© Matthieu Million - Fotolia.com)

Why do I use this example of ancient history when writing about Service Oriented Architecture? Because again and again, we see integrations which build great walls to secure the "land" by providing firewalls, de-militarized zones and other security measures - but leave the applications (or in Qin Shi Huang's case - the cities) wide open. As history has shown us, once the intruder is in the realm, or in our case - the production environment, all the gates are wide open. Let's have a look at today's wall - and how to close the gates.

The first open gate is - no gate at all. As web-services are developed in rapid pace and on short notice, the implementation of a log-in mechanism and transaction security is seen as a secondary concern. As a consequence, even these basic security mechanisms are not implemented. A web-service in the production environment is deemed secured because the generation of a service call is seen as too complex to be done manually. Additionally, an external intruder needs to know the service catalogue, the internal data structures, the integration steps, etc. How could an external intruder possibly get all this information?

Remember the Chinese empire. An internal informant provided all the information that was needed to penetrate the system. As soon as the information is available, all that is needed is an XML editor and submission tool and access to the production environment. Some of these tools are standard packages used for the development of the applications, such as the free tool SoapUI. This is one of the most important tools for developers of web-services using the Microsoft SOAP standard.

Gaining access to the environment does not even have to pass through already secure environments. Security experts have published papers for years demonstrating that the main threat to an organization's IT infrastructure comes from within the organization. A disgruntled developer, a playful operator and the gates can swing wide open.

The second open gate is often the result of carelessness. Even if the gates are built and stable, closing and protecting them is expensive and time consuming. In many cases, those gates remain open; the user identification is uses a "standard" user which might even be identical for all applications. This scenario is even more dangerous than not having any security at all because it provides a false sense of security. In some cases, the identification of the user is wired into the messages and the intruder only needs to copy it. In an even worse scenario, that user information might even work on the human customer interfaces, opening the application even further. A special variant of problem is the developer "backdoor". In this case, code used in debugging and coding interfaces are developed and used during the development and testing process - but not removed before the application is moved into production. This code provides an open door into the application similar to an open backdoor into a town. The development of web and web-service enabled applications does not normally include a security scan - so those doors are forgotten, overseen, and sometimes, even left in on purpose. (The rationale for this is they are useful in case something goes wrong and the application needs to be supported by an expert. But what happens if the expert becomes a spy?)

The internal attacker does not even have to be hostile or a human. In an ungoverned production environment, the migration of services (consumers and providers) can result in requests that are unplanned, uncontrolled, and unwanted. These "rogue services" can wreak havoc in the production environment. Of course, this can be prevented by having strict rules of deployment. However, a more secure option is the closure of access to the system on the provider level. If a rogue service is installed and is running against a closed door it might cause some incompatibilities with another owner of the consumer, but any additional services remain close and secure.

So, now the gates are guarded and the backdoor is closed. The city is safe! But, is it? Imagine a guest coming to the town to be let in - and the guards simply allow him entrance. Additionally, as soon as the guest/user is in, he is allowed to use all facilities, all tools, see all archives and rewrite all laws. What is the equivalent in our IT world? Very often the web-service client has a high level of permissions. This usually happens for two reasons:

- To actually reduce transaction errors. If the integration has to handle additional technical and functional issues, as well as security issues, the actual integration becomes much more difficult and needs much more effort. The easiest (and laziest) thing to do is give the web service privileges and avoid these issues. I'll leave it to you as to whether this is a good idea.

- Generating user accounts and permissions consistently over many different systems is difficult and expensive. Using a component that provides a single sign on (also called SSO) means that additional funding for development and maintenance needs to be provided. It is much easier (and seemingly cheaper) to generate a single user which can access all applications and is trusted as soon as authenticated. In real life, would you give anybody unlimited access to all your information after just one authentication? Or would you feel happy if your bank allowed someone to execute transactions for you - just because your account representative has identified him?

This list does not discuss security issues caused by the actual coding and development which is another area of concern. It does not include the security issues in the different pieces of software which are used to expose the web-services environment to the world. The issues described so far are easy for someone with an IT background to exploit. If the coding issues are a small hole in the wall - these holes described here are big enough to let an aircraft career pass through.

The consequences are obviously serious. So, what can you do? There are a few steps which everybody should take who wants (or needs) to expose any application with web-services to the network or Internet:

- Plan your walls. Enable every means of security the protocol allows. This implies that user accounts and passwords must not be transferred as non-encrypted (readable) text. A better option is to encrypt the authorization information with an algorithm that provides changed information every request. An additional "gate" could be to use date and time as part of the encryption mechanism.

- Plan your authentication and authorization. When building the business process which underlies your integration, also define the users and their permissions in the applications that participate in the process. Make sure that the user really is allowed to do what he wants to do.

- Make sure that you can identify the real user behind the request in every application participating in the process. Just storing the web-service as a user allows information to be changed, without a proper audit trail, from users that cannot be identified or that are almost anonymous.

- Finally, execute frequent security checks. Ensure that the development and deployment processes are designed, developed, and tested taking security into account. Know what the web services are doing in your system and make sure that you can identify the actual user behind the service request.

Conclusion

Even as security in integrated environments is recognized as being important, the use of existing security concepts with the traditional firewall as main defence does not provide the security required in today's hostile world. Services need to protect themselves. The best approach to security is to treat the services in a production environment as if they are exposed to the outside world **without** a protective firewall. Expecting interference from the hostile outside and also from internal sources that can normally breach security much easier than an external attacker.

A great wall is a great tool to secure your world. But if there is anything we can learn from Lenin in the world of information technology, then it is his alleged proverb that "trust is good, but control is better."

Position 10: Central Data Management - or just a persistent canonical data structure

Few integration projects fail due to technical issues. When analysing the reasons for a failed project, it is mostly the discussion and communication (or lack thereof) of the different involved parties that causes the project to be terminated prematurely.

One of the most fiercely fought battles surrounds the ownership of core data. This discussion is less about the responsibility for important, but very static reference information (i.e. postal codes, calendar information) - but more about information that is rapidly changing and is also a core business asset. Client information and contracts would come under this heading. The conflict is often not about the right place to store the information, however. It is often about the power that is contained within the data - following the rule that "ownership brings power".

If I hold ownership of the customer data, I own the customer - other departments have to come to me if they want their share of the information. I have participated in a number of discussions and design meetings which did not try to solve a technical problem - but which were more focussed on company management issues, i.e. the right way to handle information and who owns the business processes. If the design meetings are running in this direction you are only a second away from the worst possible catastrophe of the integration project - political design paralysis and imminent project termination. The end result can be very bad news. Before the project is terminated a lot of time and money is spent and a lot of people are working for the bin. Unfortunately, that is not the worst outcome. The worst is that as a result of the termination a general look for a guilty party will always point to the integration team (as the weakest link in the organization). The underlying cause for the problem - the failed discussion on data and functional ownership - will not be addressed.

I am not the only architect who has seen this situation. Every integration architect with a number of years of experience can discuss these war stories. You will not hear them during conferences and product meetings with the vendors, but in the evenings when they meet their friends and former co-workers – the tales begin.

Coming home from these meetings and hearing these discussions, every architect starts thinking about trying to find a *technical* solution for an *organizational* problem. Oddly enough - in this case - there might be one.

If the best place for the data is not in the applications, why not keep it in a place that is neutral to all parties - in the integration infrastructure. As the work is done, a canonical representation of the data structure should exist, and with it a representation of the information independent from the applications. It might even have been defined down to the level of the typed attributes (which is a very good practice to employ to avoid integration problems), so why not keep the data in the canonical form in the integration layer?

Technically, the solution sounds pretty straight forward. For the core entities of the organization, operation or firm, you develop a number of core services. These services can be used by the different business applications to coordinate (create, update, and retrieve) their internal representation of the business entities with each other. And as these entities change, so the central storage of the data is updated.

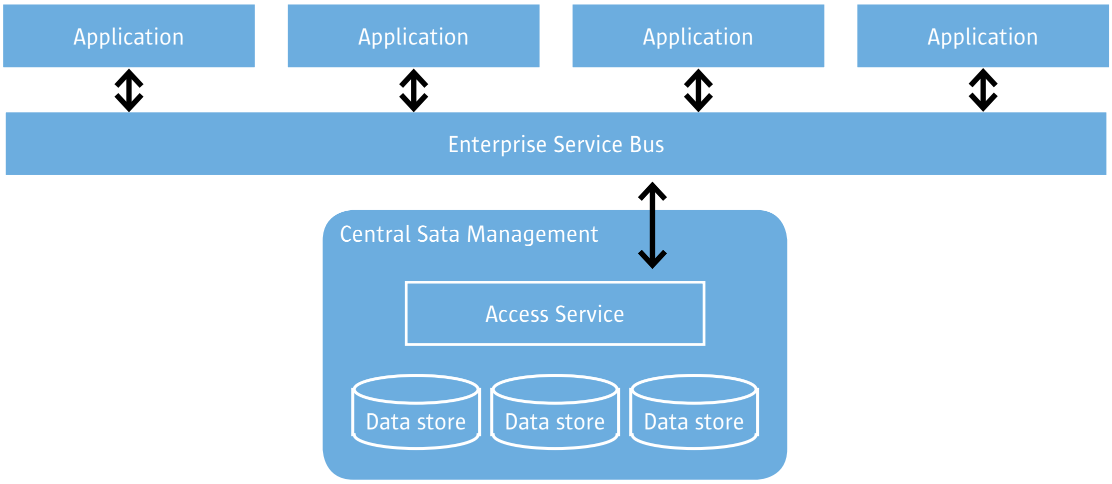


Figure 9: Central Data Management: Simplified Architecture

There are many examples for approaching data storage and use this way. Many organizations use central repositories for their master data, i.e. reference information, and ZIP codes. This storage, important as a technical answer to a technical issue, illustrates how to coordinate base information on a technical level. The most common example for a central storage of information is the well-known mechanism of handling of user accounts in an LDAP server or Active Directory structure.

Technically the storage of the data in the infrastructure is not a problem. The necessary frameworks are either provided by the vendors of the integration frameworks (like TIBCO) or can be built (on a basic level) by the integration team itself. Some service buses do not offer the level of guaranteed delivery the integration architect or business requires. Because of this, the integration team may already have built a level of storage in the bus to ensure that the data is correctly delivered. These solutions can be updated and expanded to also accommodate the missing services needed for the integration approach.

So, the need for a solution might be there, the technology is there, for sure. So, where are the pitfalls of the solution? Why is it not being used everywhere it is needed? The first reason is - as usual - cost. By building the central data repository, another replica of data is generated. This data is already in the environment. This additional data storage incurs costs in the generation of the solution, but also operational costs when running and modifying the structures. These costs can be quantified however. Unfortunately, the savings are more hidden. But, let's discuss them.

The first level of savings is in the increased level of data consistency in the IT environment. This is an indirect saving to the operations manager as he can reduce the number of staff maintaining the existing data with fewer service calls to his department.

A second level of savings is in the application- independent storage of corporate information. This becomes important if a decision is made to either introduce new functionality and solutions, or replace existing solutions. In the first case, the centrally stored information is a good foundation for the population of the new system. Some enrichment processes will be required, but can be planned into the environment. For the replacement of existing solutions, the central data storage is a perfect master copy of the core data and for data cleansing.

A second reason for the unpopularity of this solution is missing sponsorship. Remember this dispute is about the ownership of the information, but this is also a problem in the solution space. The owner of the solution in the corporate world has to become a sponsor for a part of functionality that is outside of their domain. This obstacle can only be removed if the lead of the integration team has enough political "clout" to build the solution outside of direct project planning. This solution is very successful in organizations where the integration of applications is seen as core functionality and equipped with project independent budget. As part of a single project, this solution might have a less successful reception. I can usually determine the abilities of an integration team to implement this kind of solution by looking at the organization and the sponsoring of the canonical data model. I have found that organizations that handle their canonical model as a permanent central function of their integration are much better suited to build this solution.

The final obstacle to the implementation of this solution is the timeline of many integration projects. If the concept of a central data store becomes part of the critical path of the project, it is very likely that it will be rushed and built as a pure data cache. It then loses many of the features that can make it a benefit for all involved parties. Some of those benefits are:

- Knowledge of the structure of information is available during the integration design process. The connectivity and extension of the central storage still remain as challenges for the work in the project, of course.

- Additional synergies. One example of this might be the use of the central data storage for business intelligence processes. These can be introduced as part of independent projects as only those types of projects sponsor the maintenance and modification of the register.

To summarize, the central data repository can be a most useful tool in an organization that has very strong and independent systems which need to be integrated. It removes some of the main obstacles for the successful completion of integration projects - the discussion of data ownership by introducing a central and independent place to handle the data. It is technically feasible and can be readily introduced into service architecture. However, it does incur permanent costs which need to be justified by benefits that exceed its use as a neutral broker.

A final word of wisdom to all the architects who read this: Do not rely on a *technical* solution for a *management* problem. The solution described here might work in many cases, but more often than not you will need to talk to the sponsoring manager to address the underlying problems.

Conclusion

Central Data Management can be a useful tool to ensure that important data is kept synchronous with the replicas in the applications. It can also be used as a central data layer, providing input to the applications during the execution of business processes. This is a different approach than giving ownership to data to single applications. However, because of the independence of the data from the applications it can be kept in a canonical format. The use of a central data repository does end the need for the definition of ownership on the data within the processes. But, it is not a solution for conflicts between application owners – the resolution of those has to be part of the design process for services in the environment.

How can BearingPoint help you?

BearingPoint is a leading provider for services and solutions for integration and service oriented architecture. Supporting clients Europe-wide, we bring a wide range of experiences which can help you to be successful in your approach. We can help you by:

- Helping you to review your current situation in your environment.
- Supporting you in the introduction of an end-to-end service development process, starting from the business side of your project to the final roll-out.
- Selecting the right integration environment for you. BearingPoint praises itself for being an impartial advisor on the market.
- With experienced design, development, and testing staff BearingPoint is not only able to fill your needs with expertise, but also to help you to build your service expertise and development stream.
- BearingPoint has extensive knowledge in the modelling of enterprise data. The combination of business expertise and technical background helps you in building the data model that reflects your business.
- BearingPoint has experienced architects with an average of 12 or more years of business background. We can help you to work on the most challenging situations.

About the Authors

Andreas Grimm is a Senior Technology Architect at BearingPoint and a leader in the integration practice in the Netherlands. With over 20 years of experience in the Telecommunication, Utilities, and Financial Industry (of which 12 years are in the integration area), his main focus is in the design and implementation of integration projects. Mr Grimm holds a German degree in Computer Science (Dipl.-Inform.) and a British B.Sc.(Hons).

Chana O'Leary is a Technology Architect at BearingPoint in the Netherlands. She has over 15 years of experience in all phases of enterprise and software architecture development covering the areas of utilities, healthcare, insurance, aviation and private sector development. She is an experienced project manager, trainer and team mentor in the areas of advanced development technologies. Ms. O'Leary holds a Bachelor's degree in Professional Studies and a Graduate Certificate in User Interface Design and Usability Engineering.

About BearingPoint

BearingPoint. Management & Technology Consultants.

We deliver Business Consulting. We are an independent firm with European roots and a global reach.

In today's world, we think that Expertise is not enough. Driven by a strong entrepreneurial mindset and desire to create long term partnerships, our 3200 Consultants are committed to creating greater client value, from strategy through to implementation, delivering tangible results.

As our clients' trusted advisor for many years (60% of Eurostoxx 50' and major public organizations), we define where to go and how to get there...

To get there. Together.

For more information: www.bearingpoint.com

Argentina* | Australia | Austria | Belgium | Brazil* | Canada* | Chile* | China* | Denmark | Finland | France | Germany | Ireland | Italy* | Japan* | Korea* | Malaysia* | Morocco | Netherlands | New Zealand | Norway | Poland* | Portugal* | Romania | Russia | Singapore* | Spain* | Sweden | Switzerland | Taiwan* | Thailand* | United Kingdom | United States* & Emerging Markets in Africa

*Markets served through our global network in Asia, North and South America: ABeam Consulting, West Monroe Partners, Business Integration Partners & Ipopopema Business Consulting

Credits photo : fotolia.

BearingPoint®

Management
&Technology
Consultants

www.bearingpoint.com