Making SOA Work in a Healthcare Company

Jay Blanton
Health Net, Inc.
12033 Foundation Place
Rancho Cordova, CA 95670
1-916-935-1223
jay.blanton@healthnet.com

Steve Leski
Health Net, Inc.
21271 Burbank Blvd
Woodland Hills, CA 91367
1-818-676-6531
steve.leski@healthnet.com

Brian Nicks
Health Net, Inc.
21271 Burbank Blvd
Woodland Hills, CA 91367
1-818-676-6463
brian.nicks@healthnet.com

Traian Tirzaman
Health Net, Inc.
21271 Burbank Blvd
Woodland Hills, CA 91367
1-818-676-5739
traian.tirzaman@healthnet.com

Abstract

Making SOA work in a large and diverse healthcare company is not just about bridging the gap between business and IT. It is also about bridging the gap between the technologies of yesterday, today and tomorrow. As Health Net has grown by acquiring other entities, we have acquired a landscape of diverse assets written with many languages, hosted on many platforms. These range from Java on WebLogic to .Net to RPG on iSeries to CICS on zSeries to COBOL on OpenVMS. Integrating these systems goes beyond simple business services. Successful integration ultimately requires elevating IT teams to the vision of a SOA enterprise as defined by an enterprise reference architecture. Educating our IT project teams in the fundamentals of SOA design and development has involved special approaches and a commitment to mentoring and continuous education in the enterprise. This discussion covers some of the challenges, successes, and lessons learned that we have encountered in bringing SOA to Health Net.

Categories and Subject Descriptors
D.2.0 [**Software Engineering**]: General – *standards.*
D.2.4 [**Software Engineering**]: Software/Program Verification – *programming by contract.*
D.2.11 [**Software Engineering**]: Software/Architectures – *data abstraction, patterns.*
D.2.12 [**Software Engineering**]: Interoperability – *interface definition languages.*
D.2.13 [**Software Engineering**]: Reusable Software Interoperability – *domain engineering, reusable libraries, reuse models.*
K.6.1 [**Management of Computing and Information Systems**]: Project and People Management – *training.*
K.6.3 [**Management of Computing and Information Systems**]: Software Management – *software process, software selection.*

General Terms: Management, Documentation, Performance, Design, Standardization.

Keywords: SOA, Services, Healthcare, Health Net, ESB, gSOAP.

1. Introduction

Over the years, Health Net has grown by acquiring other companies, each with its own technology stack, each different than the other. In some cases, this posed no problem, as some companies could be operated as separate divisions without the need for in-depth systems integration. For those companies that could not be operated at arm's length, we had three choices:

- Import the data into Health Net core systems and dispose of the acquired company's systems. This was only possible where we had comparable core systems already deployed.
- Replatform the systems on Health Net core technologies. This ensured easy maintenance and enhancement of the system, but was prohibitively expensive.
- Integrate the systems, overcoming differences between the technologies, and trying to create a target hybrid platform that would provide a seamless user experience. Then, over time and as part of ongoing project work, merge the systems onto a single strategic technology base.

As this third integrate-and-merge approach has offered us the most value for cost, it has become a key strategy. Over the years, our EAI efforts grew. We finally reached a point several years ago where we were at a crossroads. We had invested over time in a variety of internally developed and proprietary vendor EAI solutions, and were performing mostly point to point integrations between solutions running on VMS, iSeries, zSeries, Java and .NET. The lack of interoperability of these EAI solutions, and their long-term strategic value were a concern. Should we continue with what we had or migrate to standards-based EAI technologies?

2. Early Steps

The industry was abuzz with stories about SOA, how it was a new integration approach based on well-defined, independent, interoperable and reusable services. SOA offered us a standards-based integration approach that we felt might address our need for a strategic approach to EAI, but we had concerns. Was SOA something new or just a re-skin of existing industry approaches? Why would SOA succeed where other EAI approaches failed or produced less-than-adequate results?

As we learned more, we found more to like about SOA.

- SOA had reached a point of technological maturity. There was an explosion of specifications being ratified by industry Standards Committees and vendors were aggressively supporting the emerging standards.

- SOA offered a new paradigm for viewing our many systems that could be understood by both our business and IT teams. It offered new empowerment to our business teams by centralizing on a common vocabulary defining a service as a technology-agnostic embodiment of part of our business: a realization of a business requirement and a physical implementation of a solution to that requirement. We felt that this common vocabulary might help increase alignment between our business and IT teams.
- SOA offered reusability and interoperability at the enterprise level, across our many different technologies and business lines. We were encouraged by the SOA potential to wrap, expose and reuse our existing capabilities in a standard way across the enterprise.
- Orchestration of services would allow business agility, and faster time to market.

We felt that what we had learned had enough merit to warrant investing in a SOA platform for Health Net, and adopting SOA as a strategic company initiative.

At the time, research from leading technology analyst organizations indicated that the Enterprise Service Bus (ESB) was the leading entry point for implementing SOA. An ESB would provide a communication infrastructure, management of enterprise messages (validation, enhancement, transformation and routing), service orchestration, process management and security. It offered the advantages of SOA standards support, a relatively low cost and easy configuration.

Implementation of an ESB became the main focal point of our first SOA implementation. We first researched and identified the leading vendors in the ESB space. We took into account our existing vendor partners and leveraged those relationships where possible. We ultimately arrived at a finalist list of three top-tier ESB vendors and invited them to do a proof-of-concept implementation with us. We established requirements for the proof-of-concept that would allow us to prove out standard ESB capabilities as well as differentiating capabilities we felt would address specific challenges we had within our enterprise. We wanted to demonstrate not only connectivity between some of our disparate heterogeneous platforms but also service orchestration, one of our key requirements. We provided each vendor with detailed specifications for each platform, our connectivity requirements and the scorecard that we would use as a success criteria. We then conducted the proofs of concept, ranking each vendor product using the scorecards. Finally, at the end of the proofs of concept, we employed the services of an independent consulting company to review our scope, approach and findings. Some of the recommendations based on their analysis were:

- Our finalist list of ESB vendors was adequate. Our primary focus on connectivity was appropriate, given our heterogeneous systems environment.
- The timely selection of an ESB was crucial. Delaying the decision would be riskier than choosing any of the three top vendor products. Further proofs of concept would most likely not change the final outcome.
- The first implementation would likely take some time, and we would have a steep learning curve getting into SOA, so early action should be preferred over further, deeper analysis.
- Picking an ESB product was only the beginning of realizing SOA. Creating a SOA Center of Excellence to define the SOA roadmap and SOA best practices for the organization would be crucial.
- Rather than take a "build it and they will come" approach to developing SOA services, we should start using the ESB with our existing healthcare projects, identifying and building enterprise services in support of those projects.

Taking these recommendations into account, we bought our ESB finalist, a clear winner, and got underway.

3. Building the SOA Foundation

It was clear that the ESB implementation would be just a part of our SOA plan and that going forward, we needed an overall foundation to support SOA in the enterprise.

3.1 The Role of the Chief Technology Office

From the outset, we understood the need for a centralized enterprise approach to managing our SOA effort. We created a SOA team in the Chief Technology Office to help manage the effort, to help address the mountain of questions and issues that we knew would arise throughout the effort, and to act as a center of excellence for SOA. This enterprise-level support was important, as no project team would have the time, funding or breadth of experience to address all the pieces of a successful SOA program. This SOA team was charged with:

- Rollout of the enterprise service bus, along with an initial set of core enterprise services.
- Mentoring teams in estimation, design, development and testing of SOA solutions, and providing "Brown Bag" sessions for the IT department on SOA topics.
- Documenting best-practices, procedures and FAQ's into "cookbooks" and instructional videos that described how to expose and consume web services on the primary company technology platforms, and how to proxy them with the Enterprise Service Bus.
- Acting as a sort of "high-tech help-desk" to troubleshoot issues teams would have with the new SOA infrastructure.

Initially, this SOA team was barraged by requests for help and instruction, and they were stretched very thin. As common questions and help requests found their way into cookbooks and instructional videos, and as development teams began ramping up on SOA, the work became more manageable.

Today, our SOA team continues to fulfill these roles, and sees between 3 and 10 SOA support requests per day from different parts of the IT organization.

3.2 Designing the Reference Architecture

We felt that a SOA reference architecture, tailored to our environment and approach, would be key to a cohesive and organized approach to designing services. We created a comprehensive reference architecture platform that covered:

- Architectural principles, including data, security and service principles, as well as those principles that would be common to all of our service oriented architecture.
- A SOA reference architecture tailored to Health Net, including logical, implementation and deployment views.
- Data, integration and security architectures in support of SOA.
- A set of design guidelines, including recommended approaches not only for common services, but for presentation services, high availability services, transaction management, messaging and stateful versus stateless design.

This comprehensive set of principles, architecture and guidelines provided us with an overall picture, a radar-view we could use to see where future services would fit.

3.3 SOA Governance

SOA Governance is defined differently by different organizations, and we knew that we couldn't be ambiguous in our approach. Likewise, we knew we couldn't be heavy-handed, creating such a mountain of regulation and structure around SOA that no team in our company would attempt it. What we started with was comprised of:

- An enterprise SOA standards committee to create, debate, vote on and ratify SOA standards for the enterprise. We invited members of each project team to become members in the committee, so that resulting standards would serve the enterprise. In addition to adopting and ratifying external standards such as industry WS standards, the committee created internal company standards to fill the void where industry standards left off.
- Roles and processes for conducting architectural oversight over SOA project design, in order to make sure that these designs followed our reference architecture.
- Implementation of a service registry to catalogue and describe the enterprise services that we would build and expose through our ESB.
- A mentoring process for coaching teams in the use of SOA best practices.

We have reviewed, revised and adapted this process over time. It is designed to be a living process, growing as necessary to accommodate the needs of the enterprise.

4. Finding a First Application

Finding candidates for a first application for SOA within Health Net was not difficult. We had many legacy platforms with functionality we were eager to expose as web services and choreograph into business processes.

Rather than choose arbitrarily, though, we felt that the first application should be:

- Simple to implement.
- Provide solid ROI.
- Involve several project development teams.
- Be minimally invasive to business functionality.
- Act as a vehicle for the deployment of an enterprise service bus.

All of these criteria were important to us. A complex application involving a great deal of business interaction would take the focus away from the deployment of the SOA infrastructure, and we needed the focus to be on the infrastructure, so we could concentrate on getting it right and tuning it and creating the process around it for the many business services to follow. Without an ROI, we wouldn't be able to justify or fund the project. We knew that by involving several project teams, we could effectively jump-start the SOA training process in the organization. The increased participation would increase organizational buy-in, too, and help drive adoption.

Our perfect first candidate for SOA turned out to be the replacement of our legacy proprietary enterprise messaging platform. It was aging, and the manufacturer no longer supported it. We faced either replatforming its services on the latest proprietary vendor messaging product, or replatforming them as web services on our Enterprise Service Bus. The support fees for the proprietary platform were high, so we had a compelling business case. Several teams were involved in building, enhancing, maintaining and using these proprietary services, and they were keenly interested in a future path that was sustainable for the business logic contained in these services. This gave us enthusiastic partners with which to embark on our first SOA journey.

We learned some interesting things along the way. Services on our legacy messaging platform were designed to spawn many simultaneous parallel message calls in order to complete a single transaction. This made for a very chatty message bus. There wasn't a clean way to duplicate this paradigm on the ESB, so teams were forced to either write wrappers for this parallel activity or create other workarounds.

Eventually, our ESB vendor released a version of the ESB that captured this parallel-call process in a new pattern implementation that offered us an avenue for easy replatforming of services that used this pattern.

With the help of the participating project teams and centralized management and direction by the Chief Technology Office, Health Net achieved its goal of deploying an enterprise service bus, moving off of its legacy messaging platform and creating a compelling and useful SOA infrastructure for future use by the company.

5. Case Study: SOA on OPENVMS

With a solid SOA infrastructure in place, it was now time to SOA-enable one of our core legacy systems. A fair portion of our business ran on HP OpenVMS clusters, but there wasn't a lot of commercial or open source software available, and this made the role of an OpenVMS architect challenging.

Most of our application portfolio is made up of custom-written COBOL, C and VAX BASIC programs. In order to take advantage of what we already had, we embarked on finding a software package that could leverage those assets into our SOA plans. When we began proofing SOA on OpenVMS, few technologies were available to integrate legacy software into a SOA. There was an extraordinarily expensive commercial product, and a free web services toolkit, called WSIT, from the hardware vendor. We decided to test WSIT.

5.1 First Attempt using WSIT

The Web Services Integration Toolkit (WSIT) is a freeware, OEM vendor supported SOAP development suite for OpenVMS. It is Java-based, and includes compilers and middleware to bridge Java to several legacy 3GLs including COBOL. Using Java, you can then build services into the desired application container. It had one shortcoming: it supported only the production of web services, not the consumption of web services. We piloted a proof of concept effort to get our first heartbeat out of an actual web service on the OpenVMS platform, and ultimately got an experimental service up and running. There was a lot of interest and support from HP in particular since we were one of their first customers to use WSIT.

After successfully implementing a proof of concept service, it was time to apply what we learned to building and deploying a service into production, for use in our business. For this, we employed a thin-thread approach; meaning that we chose only one simple business function to implement. This way, we would minimize complexity as we built the application through all the architectural layers, from presentation to persistence. Keeping the scope small virtually guaranteed success and would give us a quick win to build on. We used WSIT to build the artifacts and soon finished the application. It was deployed to production without a hitch, and has been working reliably in production for over a year now.

Unfortunately as we explored adding more services to the inventory, we discovered that WSIT wasn't performing very well. It was slow to execute, and we couldn't run many parallel instances without incurring high memory overhead. This hadn't impacted our business customers so far, since our service saw only low-volume activity. However, when we began load testing other service candidates, it became apparent that we needed to resolve this problem before we could move any further.

We did extensive troubleshooting with the vendor. We tried a multitude of application and system tuning approaches, and also tried seeing if an upgrade in operating system would help. None of these approaches worked. We were planning an eventual migration to faster hardware, but weren't sure that this would solve the performance problems we were seeing, so we started looking for another solution.

5.2 Second Attempt using gSOAP

During an onsite training class for WSIT we attended, a question was posed as to how a service could be consumed from COBOL, since WSIT doesn't support this. As luck would have it, the instructor had been working on an OpenVMS port of gSOAP, an open-source community solution that could be used to both produce and consume SOAP services.

gSOAP was the ideal fit. It didn't require any special middleware, as services could run on Apache server. The generated artifacts were native executables, written in COBOL, so there was no need for a JRE. We tested it, and found that performance was significantly better than the WSIT solution, by approximately an order of magnitude. So we decided to take our chances with this open-source port.

5.3 Refining the Solution

Once we settled on gSOAP, we then needed to integrate the tools and middleware into our existing development environment. For the most part, this was easy since gSOAP allowed web services to be built using the same 3GL languages we were already using. Our process for promoting applications from development to production only needed minor tweaks. The Apache server was easily configured and middleware was already in place.

Our greatest challenge on this platform was getting our existing developers up to speed quickly with web service development. gSOAP is C/C++ centric. Most of our platform developers were COBOL programmers, and they did not have experience with OO, XML, WSDL, SOAP and service design. But they knew the business and existing solutions logic very well, and we wanted to take advantage of that.

The OpenVMS calling standard allows for interoperability between all programming languages. So the code generated by gSOAP can interact with stubs and skeletons written in COBOL. To make things easier for our developers, we created an application framework that reduces exposure to the unfamiliar C-language constructs. Our framework consists of tools to simplify construction and debugging of services. It also contains type maps and definitions to specify the XML schema data types in a way that both the generated C and COBOL user code can consume. The most important feature is a set of conversion methods to take the C data types serialized from gSOAP and convert those to COBOL equivalents. COBOL has some unique data types that C does not have, such as "USAGE DISPLAY TRAILING", and it doesn't handle null terminated strings.

The framework also had to impose some standards on the service contract design due to limitations in COBOL. C is case sensitive, but COBOL is not. When creating a web service, gSOAP generates code in C or C++ that calls a user-written function to perform a service operation. OpenVMS COBOL has a limitation that requires function names to be in uppercase. As a result, the service operation name also has to be specified in uppercase, in order for it to execute. Also, COBOL passes parameters by reference only. So the XML schema data types have to map to pointers to the equivalent C data types before they can be passed to a service operation.

5.4 Supporting OpenVMS Developers

In addition to the cookbooks we wrote for the OpenVMS platform, we also created a small set of sample services to illustrate concepts described in the cookbook. This gave developers a way to get started quickly by doing a quick cut, paste and modify.

Since our audience for these instructional materials was comprised of COBOL programmers, we had to assume that they had little or no exposure to SOA concepts. We found the core concept of XML particularly important to teach, as most of our developers on this platform were unfamiliar with it.

gSOAP only supports contract-first development, a significant plus as this is the preferred approach. But in the beginning, we found that several teams were designing service contracts that were nothing more than a mirror of the legacy interfaces they were supposed to abstract. This was due in large part to the learning curve required to effectively create enterprise-level SOA designs. It was easy for programmers to view SOA as just another point-to-point communications technology. Thinking in terms of the big picture, in terms of enterprise reuse and interoperability, was a new concept, requiring education, support and oversight.

Over time, equipped with the gSOAP solution and enterprise mentoring and support, we saw our OpenVMS team members grow proficient with SOA, and this opened new potential for our company to leverage the substantial business functionality we had running on that platform.

5.5 Next Steps

As we still have a lone WSIT-based service in production, we plan to displace it with an equivalent, albeit speedier, gSOAP service in the near future. This replacement will have no impact on the service's clients whatsoever, as the Enterprise Service Bus provides a valuable layer of indirection for the back-end business services. In this hassle-free service swap, we will see the fulfillment of another part of the promise of a well-designed service oriented architecture.

6. SOA Challenges

6.1 Teaching SOA to the Enterprise

Learning how to teach SOA to an IT department with a wide array of heterogeneous skills and experience was a real challenge. The job was easiest with our Java teams. Most of our Java programmers had been doing web development for some time, and it was easy for them to understand the relationship between a Java method and a WSDL operation or between a Java Object and a schema complexType. They were already versed in basics such as XML, so there was little knowledge gap to bridge.

It was much harder to teach our RPG or COBOL programmers SOA concepts because the sheer number of new and unfamiliar terms, concepts and tools made the learning curve very steep for them. We found that the fundamental key to bridging the knowledge gap lay in aligning vocabularies between their world and SOA. With a common understanding of terms and taxonomies, we would have a common language to act as a vehicle for all of the concepts they needed to learn.

We could have approached the SOA education of our programmers with a "we lecture, you take notes" approach, but we had learned a lot about mentoring teams in technology subjects by then, and we had learned that education as a one-way street rarely works well. Our enterprise SOA mentors, who were mostly JEE and SOA experts, found that they needed to learn and become conversant with iSeries and OpenVMS concepts and terminology in order to teach SOA by analogy, in the language of their students. This interactive approach to teaching was crucial in early adoption and ramp-up of our development teams.

Important too were the live demos we conducted during enterprise Brown Bag sessions, where we demonstrated web services on different platforms. There was a real impact on development teams when they saw cross-platform message interchange, devoid of any platform binding.

The one constant between all of our teams, regardless of technology, was their open enthusiasm for learning about SOA, and their eagerness to employ what they were learning in upcoming projects. This, combined with their experience and intellect, allowed them to learn and begin employing SOA rapidly.

6.2 Design Challenges

As services began to grow around the ESB, the enterprise began to seriously adopt SOA in their project designs. The low hanging fruit was comprised of proprietary services that wrapped point-to-point integrations, some based on end-of-life products. We migrated them from their point-to-point communication to web services hosted on the Enterprise Service Bus. As teams began to convert current point-to-point applications into service-enabled endpoints, we went through many service design challenges. Our project teams tended to design services "code-first" rather than "contract-first". Thomas Erl describes the service contracts produced by these code-first services as having high "implementation coupling" [1].

The result of this coupling is that even though we had simplified the integration of some of our more complex systems, we had created contracts that represented the underlying implementation details of the logic used within the source systems. This caused service clients to develop their code against an implementation-based contract, which was impossible to orchestrate into meaningful business services. For example, a CICS application might have a function called EMP77047 with fields IEX789, IEX777 and IED456. If a service contract was generated from the source code that represented the underlying program, the contract created would mean nothing to the enterprise – it would only hold meaning for the original author of that service contract. This antipattern was exacerbated by some of the web service code-to-service generation tools, which offered an exceptionally easy way to convert programs to web services, but preserved all function and variable naming in the process. As these tools would only perform code-first contract generation, teams' hands were tied when it came to adopting contract-first best-practices.

This is where the benefit of SOA design patterns and the Enterprise Service Bus came into play. The Enterprise Service Bus can proxy an underlying service that has a poor contract with an enterprise view of the operation and entity models defined within the WSDL of the underlying system. Thomas Erl refers to this as the "Legacy Wrapper" pattern [1]. Application of this pattern effectively remediated the coupling we were seeing in some of the services.

6.3 Supporting SOA Development

As our SOA infrastructure began to grow, there came a pressing need for tools, processes and standards to support the new enterprise development process. In the past, teams had often built code in local project-oriented silos, and promoted system enhancements into production with limited external dependencies. There were controls in place to make sure integration and regression testing took into account affected systems, and further controls in place to ensure that code met standards before it was allowed to run on production systems, but these controls were different for each platform, and it soon became clear that for these new enterprise-spanning services, we would need enterprise-spanning tools, processes and standards.

One of the teams that contributed to the initial deployment of our ESB had a CVS server and a build server running Luntbuild, an open source continuous integration tool. These servers were used to store, stage and build new services into our pre-production and production environments, and at the outset, they worked very well for that purpose.

As more and more teams started building services proxied through the bus, the need for support went through the roof. An administrator was required who could grant accounts, manage access, help teams set up build artifacts and monitor projects through to production. Much of this was performed by the team who owned the servers, and the overwhelming enterprise demand for service on these platforms quickly exceeded their ability to accommodate that demand.

The servers too, were suffering. The Luntbuild server was so overtaxed that a full deploy of the company's SOA infrastructure was taking it 12 hours to accomplish. Our production downtime windows were much shorter than this, so it took some real creativity to shoehorn deployments into each downtime window.

The downtime windows were themselves a problem. These windows were specified by the team that owned the deployment infrastructure, and while they met the needs of that team, they did not meet the needs of the enterprise. Negotiating windows for enterprise deployments that met the needs of all teams became a rigorous and diplomatic balancing act.

We learned from this that enterprise repositories, continuous-integration infrastructure and processes were needed for enterprise software assets. That seems straightforward in print, but it was a learning experience for a company that had grown around different business and technology silos. We learned too that our software build infrastructure needed to be designed and built for performance, so that as our enterprise-level assets grew in number, we could still meet system downtime SLA's.

6.4 Monitoring for Success

As more of our business functionality began running through the ESB, we learned that it was important to make sure that teams understood the cross-cutting concerns the bus could help them with. Since development teams were so used to monolithic systems development where there was no separation of concern, it was easy for them to assume that the only way to achieve security, monitoring, SLA alerts and the like was to build this functionality directly into the business services.

As teams ran into their first performance-related issues with SOA applications, and the need to perform Business Activity Monitoring and alert operational support teams of SLA violations became critical, we saw that much of the cross-cutting functionality in the bus was not being utilized. Thankfully, this didn't require a lot of rework to address, but we learned important lessons from this experience:

- The specification of service SLAs needs to occur during the requirements process, in accordance with business needs and the business process being modeled.
- Teams needed better instruction in how to effectively instrument their applications using the ESB.
- We needed a process to provide closer oversight during the design process, to ensure that SLA requirements were addressed, and during the completion of projects, as services were promoted into our production systems, to ensure that the appropriate SLA's were applied and enforced in the ESB.

6.5 Process in Support of SOA

Our design documents were conceived largely around our legacy systems, so there was more focus on data, data interchange and data design than business process modeling and contract design. These design documents were reviewed and approved by the Chief Technology Office prior to the development and deployment of applications, so if the documents did not address service design, this could lead to situations where improperly modeled services might go all the way through to production. We addressed this by rewriting our design document templates, so that service design would be fully documented.

The human element was important. We sought and secured more touch points within the process where the Chief Technology Office could become engaged in projects, to give architectural guidance, review service designs and correct problems early before they propagated into our production systems. As projects came out of the requirements phase, mentors in our CTO department would help direct teams in the design of services based on elements of the reference architecture, using the processes defined in our SOA cookbooks. This helped to ensure that services that were being designed were being implemented for the enterprise, rather than as point-to-point services, and it helped teach the reference architecture to our project teams, giving them an enterprise view of our business and showing them where their efforts intersected our business. This elevation of world-view from a single technology or business area to the full breadth of enterprise-level activity was an epiphany for many, and it helped align teams, technology platforms, projects and individual developers with enterprise goals.

It was important that these changes to our SDLC and our process for putting new services on our ESB not be ad-hoc. We wanted to capture and refine our processes so that we could bring the principles of continuous improvement to our SOA approach. This was an achievable goal. Organizationally, we had reached a maturity level where we could define, model, reengineer and ultimately improve our business processes. This allowed us to address our SDLC and work-processes from the abstract to the concrete. We used IBM Rational Method Composer at an enterprise level to define the processes by which we performed work at every level of our IT organization. This allowed us to define the exact processes for establishing services on the Enterprise Service Bus, identify all of the required artifacts that were to be produced, and then connect these SOA processes into the overall SDLC processes. By doing this, we had a process picture we could refine and improve over time.

6.6 Challenges Varied by Platform

The easiest part of our SOA initiative was defining and communicating our SOA vision for the enterprise. The hard part was making it actually physically happen.

Many vendors claimed to participate in the SOA initiative and embrace SOA standards, but when it came to actually developing and deploying SOA artifacts, we saw many products fall flat. We had to learn to tell the sound products from the marketing as we navigated the SOA landscape. It became very evident along the way that there was no substitute for test-driving products using use cases pertinent to our business.

Even though we were well versed in Java design and development as a company, and even though Java lends itself well to creating web services, the very simplicity of the process made it easy to implement antipatterns. In the common case, where we were exposing existing code as web services, a proper contract-first approach entailed refactoring code in support of the contract, adding substantial cost to the web service implementation. Developers found it easier to take tools like Java2WSDL, bundled with Apache Axis, and generate a web service directly from the Java code, without regard to how the object model should relate to the schema representation (such as whether a field is required or nillable). This code-first approach to web service development was an easy trap to fall into.

While support for WS standards was excellent on the Java platform, support for these standards was spotty on our legacy platforms. This led to difficulty in implementing some standards across the enterprise. While implementing SAML security as part of the WS-Security standard, we found that the libraries we were using to expose services on OpenVMS did not support SAML. On our iSeries systems, we had a different problem - they used RACF tokens instead of SAML tokens, driving the need for token mediation between platforms.

Different, too, were the pre-production environments available for our various technology platforms. We had five well-defined environments with discrete IP addresses for our ESB and our Java artifacts: Development, QA, User Acceptance Testing, Staging and Production. These were the only platforms with as many discrete environments, however. Some of our systems had as few as three environments, and so it was difficult to coordinate environmental moves for distributed enterprise SOA projects, because there was not a one to one mapping between the promotion stage and the environments. What made this exceptionally difficult is that some legacy environments had only one web server servicing all pre-production environments, which each resided in different directories on a single server. Each service in each environment needed its own WSDL, and the opportunities for error, particularly during automated service generation, were very high.

7. Lessons Learned

As no project is successful unless you learn something from it, we captured "lessons-learned" after our first SOA initiative to assess what went right, what went wrong, and what next steps made sense based on what we had learned.

One of the big things we'd learned is that our reference architecture was a fine high-level model, capturing our long-term vision of the architecture as a whole, but there was little connective tissue defined to take that model down to the more physical and practical cookbook and service level. Our developers didn't see why we or they should care about the WSDLs they generated, or why the schema structure should represent domain models. We had left them with a vision but without the information needed to translate that vision into action. Thankfully, the ongoing mentoring we were providing did communicate much of that information, but it was clear that we needed to extend our reference architecture down to the trenches in order to properly align our SOA development with our SOA vision.

Moreover, by creating that lower-level model in support of the reference architecture, we could create service design standards that supported the reference architecture. This would assist the governance process by giving service architects the ability to guide teams in the correct design of services that supported the reference architecture.

We learned, too, that it was unreasonable to expect the end state reference architecture to be represented in the first phase of the development on the bus. It made sense for us to adopt a proven SOA maturity model or create our own maturity model. Like the Capability Maturity Model Integration, a SOA maturity model would help to establish a phased approach to the progression of SOA within our enterprise. As our reference architecture defined our vision for a SOA enterprise, a SOA maturity model could help define how we would get there, and how we would know that we were successful at each phase of maturity. This roadmap of small achievable goals would help us better understand the goals of our SOA and the path to achieving those goals.

Finally, we learned that SOA is a journey, not a destination. SOA cannot be achieved at one specific point in time. It is an evolution of development, design, and process. Our reference architecture, governance model, development architecture, and other driving forces behind our SOA initiative were part of an iterative process of definition and clarity, involving the incorporation of lessons learned from the past to better shape future phases. By refining our craft based on what we were learning, we could streamline the application of best practices, simplify and reduce the cost of our development process, and begin to see the orchestration and reuse of business processes that are the hallmark of a successful enterprise SOA.

8. Reference

[1] Erl, T. 2009. SOA Design Patterns. 1st. Prentice Hall PTR, Upper Saddle River, NJ, 441.