Bottom-up and top-down COBOL system migration to Web Services: An experience report

Juan Manuel Rodriguez Marco Crasso Cristian Mateos
Alejandro Zunino Marcelo Campo

November 23, 2011

Abstract

Moving from mainframe systems to Service-Oriented Architecture (SOA) using Web Services is an attractive but daunting task. The bottom-up or Direct Migration approach enables the effective modernization of legacy systems to Web Services. Conversely, bringing migration into fruition with the top-down or Indirect Migration approach is recognized as being comparatively harder but achieves better migration results. In practice, it is very uncommon to employ both approaches to the same large enterprise system, which leaves no room for comparison. This paper reports the outcomes of applying both migration approaches on a real COBOL system, presents the followed migration processes, their costs, and reports on the discoverability and reusability of target Web Services.

KEYWORDS: M.3.0.a Web Services Modeling, M.1.0.d Services Architectures

1 Introduction

Legacy systems are an undesired yet unavoidable reality for many enterprises. COBOL programs[1], which run in mainframes, and implement business logic of large financial organizations and governments are a good example. Migration of legacy systems becomes a necessity when they force organizations to incur high costs including payments for processing power, and encumber hiring programmers that master the involved technologies. Governments increasing tendency to adhere to open-source platforms also encourages migration.

Community's experience confirms that SOA and Web Services are a right target for migrating legacy systems, specially when loose coupling among providers' and clients' systems, and agility to respond to changes in requirements [1] are needed [2, 3, 4]. The reported experiences show two main migration approaches, namely *Direct* and *Indirect Migration* [2]. The former represents a bottom-up approach as it encourages engineers to upgrade the legacy COBOL programs to services. This involves implementing a thin service layer that describes programs' APIs using the Web Services Description Language (WSDL), and communicates clients' applications with COBOL using Simple Object Access Protocol (SOAP). Indeed, many tools can automatically wrap COBOL programs with COM+ objects, which are wrapped by .NET Web Services afterwards. Despite its simplicity, the bottom-up approach has not one Achilles' heel, but two. First, though a SOA-based system frontier is supposed to describe the services' offered APIs, this approach confines WSDL documents to be mere XML representations of the "COMMAREAs", or the interface exposed by COBOL programs, which may disregard established Web Services design concerns [5, 6, 7]. Besides, even when system migration is ultimately achieved, the mainframe cannot be turned off because business logic implementation still resides in COBOL programs.

Instead, Indirect Migration represents a top-down approach that requires to abstractly define target service APIs based on a high-level view of the legacy functionality, and then re-writing the business logic, or parts of it, for implementing defined interfaces. Despite being rather more expensive than the bottom-up migration approach, Indirect Migration has two advantages. First, such derived high-level view should lead organizations to improve the system boundaries [8] using well-known services design good practices [7].

[1] According to Gartner consulting, over 200 billion lines of COBOL code are running worldwide.

Second, porting business logic from COBOL to modern platforms like JEE or .NET also means abandoning the mainframe for new, more cost-effective computing environments.

The pros and cons of both migration approaches are something very well known, discussed and undoubted in the software industry community. Many case studies reporting either bottom-up or top-down experiences exist in the literature. However, none of them compares the outcomes of employing both approaches on the same real large enterprise system because supporting two migration attempts is not feasible for most enterprises. However, we report the migration of a large government COBOL system to SOA by using both approaches. This paper explains the historical reasons that originated both migration attempts, the specific steps that were performed to migrate the system, and then presents empirical evidences confirming that the top-down attempt cuts down mainframe dependency and outputs better designed WSDL documents at the expenses of higher costs. Therefore, we expect that project managers that have to deal with decisions like those described in this paper find it as an interesting and valuable contribution.

2 Legacy System Overview

The legacy system belongs to a large Argentinean government agency. This data-centric COBOL system maintains records related to individuals including personal information, relationships, work background, and received benefits. The system runs on an IBM mainframe, and accesses a DB2 database of around 0.8 PetaBytes. Two migration attempts were performed to modernize, and expose through a Web portal a portion of such system.

Specifically, the system parts that were migrated using both approaches include 32 Customer Information Control System (CICS) transactions accessing 80 database tables. CICS is a transaction manager designed for rapid, high-volume processing, which allows organizing several programs as an atomic task. These programs comprise business logic, and database accesses (mostly input validations and queries). On average, each transaction groups near 19 files, comprises 8,178 lines of code, and performs 6 SQL queries. The 32 transactions represent 600 files having 261,688 lines of code. Additionally, for each transaction there exist documents specifying functionality, and diagrams illustrating the dependencies among various transactions.

3 First Attempt: Bottom-Up Approach

During 2009, the agency IT department faced the requirement of exposing system functionality as Web Services by using .NET, but conditioned by tight budget constrains and strict deadlines. Therefore, the IT department decided to follow the bottom-up approach, which resulted in each transaction wrapped by a *code-first* based Web Service with only one operation. Code-first is a Web Service interface construction method that promotes to develop service implementations first (.NET-wrapped COBOL in this case) and to derive WSDL interfaces later. For each migrated transaction, the IT department members performed 4 steps:

1. Automatically creating a COM+ object including a method with the inputs/outputs defined in the associated COMMAREA, which forwards object invocations to the transaction. This was done using a tool called COMTI Builder. Component Object Model Transaction Integrator (COMTI) is a technology that allows wrapping CICS transactions with COM+ objects.

2. Automatically wrapping the COM+ object with a C# class for invoking the COM+ object. This was done using Visual Studio built-in support to add project references.

3. Manually including specific annotations in the C# code to deploy it using the framework-level services of the .NET platform for generating the code-first Web Service.

4. Automatically testing the communication between the final Web Service, and its associated transaction, which was performed by means of the soapUI tool.

The organization IT department reported that it took 1 day to train a developer on these four steps and the tools employed. Professional and paid versions of the two first tools were used[2], whereas a free version of soapUI was employed.

Then, trained developers migrated one transaction per hour, mostly because all the steps except step 3 were tool-supported and automatic. Additionally, the accuracy and correctness of migrated services were not validated since they decorated the legacy system without modifying its implementation.

4 Second attempt: Top-down Approach

On February, 2010, the agency outsourced the migration of the same 32 transactions using the top-down approach. One reason to do this was to remove transactions' load from the mainframe, because the IT department estimated that the agency would had to hire more processing power by mid 2010. Another reason was to set a precedent in the feasibility of understanding and re-implementing the original programs, since most of them have being steadily modified for more than 30 years following the quickest available way, which is undoubtedly not always the best way to upgrade source code. Hence, the starting point was the same for both migration attempts. The steps followed during the top-down migration attempt are listed below:

1. Manually defining potential WSDL documents basing on the knowledge the agency had on the transactions' functionality.
2. Exhaustively revising the legacy source code.
3. Manually refining the WSDL documents in 1) by basing on opportunities to abstract and reuse parameter data-type definitions, group functionally related transactions into one cohesive service, improve textual comments and remove duplicated transactions, which were detected at 2). For data-type definitions, naming type elements using explanatory names and constraining their ranges.
4. Supplying the WSDL documents defined at 3) with implementations using .NET.
5. Testing the generated Web Services with the help of the agency IT department.

During the first step, three Web Services experts and two agency members designed WSDL documents to sketch desired system frontier together. This step comprised daily meetings between not only the specialists and the project managers in charge of the original COBOL programs, but also the people responsible for developing applications that would consume the migrated Web Services.

The second step involved revising the legacy code, and its documentation. Six software analysts exhaustively read each transaction source code and found out that:

- 17 transactions comprised a lot of business logic, and returned around 100 output parameters,
- all the transactions were coarse-grain because they offered a large view of the back-end data,
- 12 transactions implemented almost exactly the same functionality,
- many transactions internally called the same programs, which were acknowledged as potential service operations [9], or returned the same parameters. One program was called by 9 transactions, another by 8, another by 5, a fourth program was called by 4, and finally two programs were called by 3 transactions.

At step 3, the specialists employed previous findings to refine the service interfaces specified beforehand. They derived potential interfaces for the target services and preliminary entities in XSD (XML Schema Definition). Hence, final WSDL documents were iteratively built based on the desired business services, and the interfaces derived from the existing COBOL, which to some extent conditions the functionality that the resulting services can offer. Step 3 ended up with 45 operations organized in 7 WSDL documents.

[2]IBM migration tools were not used as the agency had licenses for similar tools from other vendors, whose use is enforced through an organization-wide policy.

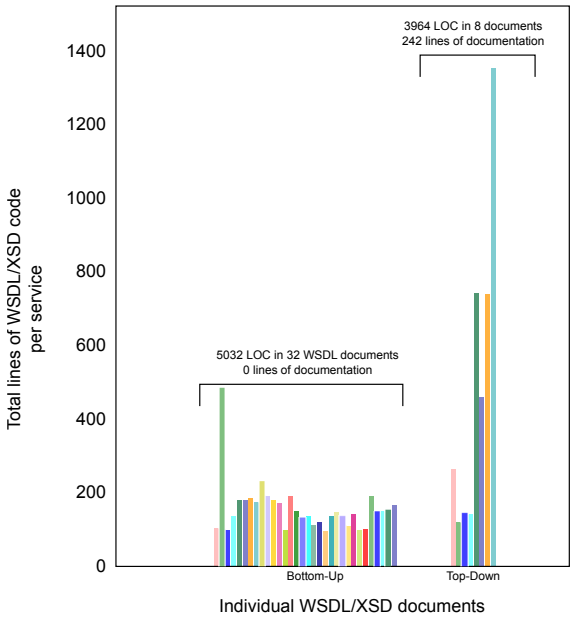

Figure 1: WSDL/XSD code lines of the different service interfaces.

They have comments describing offered operations, but not business object definitions, which were placed in a separate XSD file so they can be reused from different service descriptions.

At step 4 two more people were incorporated into the project for implementing the services using the .NET ASP 2.0 Web Service Application template as the agency required. Each service implementation consisted on a WSDL document, and a handful of C# classes implementing the business logic. Accordingly, 268 C# classes comprising 44,154 lines of code were written. Data accesses were handled through MyBatis[3], a data mapper which required developers to establish relational to object-oriented mappings via XML configuration files.

Finally, at step 5, the agency IT department employees planned and conducted functional tests to ensure that the Web Services were functionally equivalent to the migrated transactions. Basically, they executed the same test cases twice: using a terminal for invoking a mainframe transaction, and using the Web Service operation that resulted from migrating that transaction. After that, they compared both results and cooperated with the 8 external developers when something had to be fixed.

This migration attempt demanded 13 months. Step 1 demanded one month, step 2 needed three months, step 3 took only one month, step 4 (implementation) required six months, and finally, step 5 two months.

5 Comparing resulting WSDL documents

We studied the WSDL documents that resulted from both migration attempts, to assess the qualitative differences of simply decorating the legacy transactions over re-engineering them by taking into account specific services API design concerns. We employed classical metrics (total lines of code and number of resulting files) and a well-established catalog of common WSDL bad practices –i.e. anti-patterns– that jeopardize service understandability, legibility and discoverability [5]. Although both migrations satisfied the agency non-functional requirements –e.g. performance– , these are not discussed here for brevity. Indeed, preliminary results show a slight performance advantage when using the re-engineered system rather than using the wrapped one, but these systems run on different platforms making hard to assess a fair performance study. Finally, comparisons are based on statistics of the code of the deployed services.

The first evident difference is the number of WSDL documents in each specification: 32 with the bottom-up approach, and 7 -plus 1 separated XSD file- with the top-down one. Despite exposing the

[3] `http://www.mybatis.org/dotnet.html`

Table 1: Manifestation of Web Service discoverability anti-patterns.

Row	Anti-pattern name	Problem description	Bottom-up	Top-down
1st	Inappropriate or lacking comments	Occurs when some operations within a WSDL have no comments or the comments do not effectively describe their associated element's (message, operation) purpose.	YES	NO
2nd	Ambiguous names	Occurs when some WSDL operations' or messages' names do not accurately represent the elements' semantics.	YES	NO
3rd	Redundant port-types	Occurs when a port-type is repeated within the WSDL document, usually in the form of one port-type instance per binding type (e.g. HTTP, HTTPS and SOAP).	PARTIAL 50%	NO
4th	Enclosed data model	Occurs when the XSD data model is defined within the WSDL document instead of being defined in a separate file, which difficult data-type reuse across several Web Services. The exception for this rule is when it is known before-hand that data-types are not going to be reused.	YES	NO
5th	Undercover fault information within standard messages	Occurs when error information is returned using output messages rather than fault messages.	YES	NO
6th	Redundant data models	Occurs when a data-type is defined more than once in the same WSDL document.	YES	NO

same functionality, the bottom-up specification had 4 times many files as the top-down specification had (Figure 1). This stems from the fact that the bottom-up specification was blindly generated, whereas transactions were deliberately grouped by their functional similarity in the same Web Service for the top-down migration, which is highly desirable since it allows service consumers to seek semantically-related operations within a single WSDL document [5].

Another difference is the number of lines of WSDL+XSD code per document, which on average is 157 and 495 respectively. The bottom-up specification contained smaller WSDL documents, which are usually preferred by developers [5], but at the cost of scattering the functionality across several Web Services. Thus, finding a needed operation might require to inspect more WSDL documents. While, the top-down specification included less, albeit larger, WSDL documents, but arguably offered a better alternative to the contract length vs functionality scattering trade-off.

The documentation present in resulting specifications is also a difference. The bottom-up WSDL document has no comments, while the top-down WSDL documents have 242 lines of comments, i.e. near 5 lines per operation on average.

Six of the anti-patterns described in [5] were present in the studied WSDL documents, as summarized in Table 1. For fairness reasons, the anti-patterns detection was performed by two WSDL specialists not involved in the migration processes with a limited knowledge on system functionality. Besides, since detection of some anti-patterns is somewhat subjective, a peer-review methodology was applied to prevent biases.

The first two rows describe anti-patterns that impact on comments and names [5]. It is reasonable to expect these anti-patterns to affect the WSDL documents of the bottom-up specification, since all information included in them is derived from COBOL code and it does not offer a standard way to indicate the scope and purpose of comments in the code, and code identifiers have length restrictions of up to 4 characters in some COBOL flavors.

The third row describes an anti-pattern that ties abstract service interfaces to concrete implementations, hindering black-box reuse [5]. We have checked that this anti-pattern was caused by the tools employed for generating WSDL documents during the first migration attempt. Likewise, the fourth row describes an anti-pattern that is generated by many code-first tools, which force data models to be included within the generated WSDLs, and cannot be avoided with the bottom-up.

The anti-pattern described in the fifth row deals with errors being transferred as part of the output messages, which under bottom-up resulted from the original transactions that used the same COMMAREA for returning both output and error information, and from the employed method that converted any COMMAREA parameter into a service input/output parameter. In contrast, top-down WSDL documents had

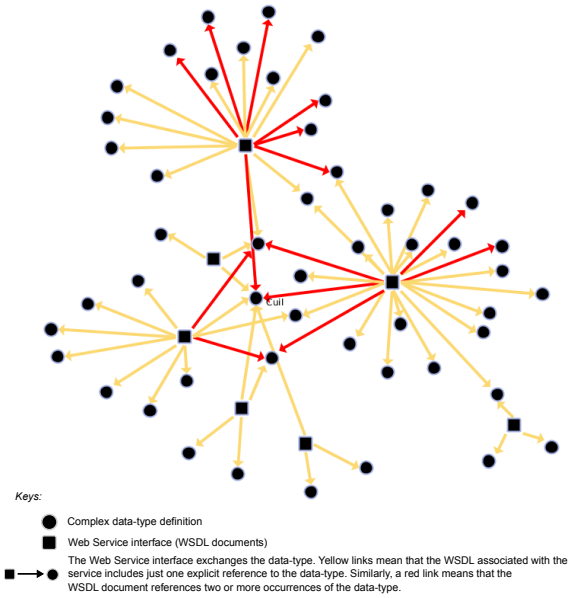

Figure 2: Data model reuse of top-down services. For the sake of readability, transitive data-type reuse is not illustrated.

a proper designed error handling mechanism based on standard SOAP faults. Finally, bottom-up WSDL documents had no error handling because ASP tools, in their out-of-the-box versions, do not support fault messages.

The last anti-pattern relates to bad data model designs. Firstly, bottom-up produced 182 different data-types, of which 73% were defined only once, and the rest were duplicated across different WSDL documents. This problem stems from a combination of two facts: the poor quality of data model design in the transactions, and the tools employed for migrating them to XSD.

With the top-down approach, 104 data-types represented business objects, including 39 simple types (mostly enumerations) and 65 complex types, in addition 131 data-types were needed for making WSDL documents compliant with the document/wrapped standard structure –this means, a message formed by one element–, which is recommended by the Web Service Interoperability standards (WS-I)[4]. For example, the Cuil data-type, which represents a business object that is analogous to the Social Security Number in the United States, is defined in the shared XSD schema as a complexType, but also wrapped by an element in the WSDL documents. Figure 2 uses the Guess tool [10] to show the relationships between the top-down Web Services and the 104 data-types designed for representing business objects. As observed, the abstractions that best represent the core business of the agency –e.g. the Cuil entity– resulted in XSD entities with higher fan-in.

6 Conclusions: approaches costs vs. benefits

We observed that with the employed top-down approach mainframe independence was achieved, but also better WSDL documents in terms of desirable design concerns such as understandability, discoverability and reusability, were generated. This nevertheless involved rewriting and porting the business logic of the 32 transactions, which demanded the resources specified in the third column of Table 2.

In contrast, as shown in the second column, the bottom-up approach took 5 days and 1 trained developer to migrate the same transactions. We have considered that the cost was u$s 3,000 for the bottom-up attempt because this was the average monthly salary for a senior .NET Developer paid by the agency, and it had the respective software licenses for the employed tools beforehand. Clearly, this could be not the

[4]Basic Profile Version 1.1: `http://www.ws-i.org/Profiles/BasicProfile-1.1.html`

Table 2: Comparison of resources required by the two migration approaches.

Required Resources	**Bottom-up**	**Top-Down**
.NET Developers	1	8
Web Services technologies experts	0	3
Time	5 days	13 months
Commercial software tools employed	COMTI Builder and Visual Studio	Visual Studio
Money	u$s 3,000 (*)	u$s 320,000

(*) This was the average monthly salary in Argentine for a senior .NET Developer at the time of the bottom-up migration attempt.

Table 3: Comparison of the employed migration approaches.

Requirement	Migration Approach	
	Bottom-up	Top-down
Minimum essential manpower hiring, regardless of the actual regular staff	✓	×
Minimum organizational changes	✓	×
Speed to produce the service-oriented version of the legacy system	✓	×
Port the legacy code to a modern computer programming paradigm/language, enhancing system understandability, stabilizability and maintainability	×	✓
Either lower-cost or modern hardware platforms for hosting the migrated system	×	✓
Updated documentation of current system functionality, dataflows and workflows	×	✓
A clear SOA frontier of the migrated system, increasing third-party applications development	×	✓
Minimum requirements for testing, validating migrated functionality accuracy and correctness	✓	×
Minimum operational risk when deploying the migrated system	✓	×

case for many enterprises and then a project manager should consider tool licensing (according to `http://www.microsoft.com/biztalk/en/us/pricing-licensing.aspx` such a license costs u$s 44,228) and man-power hiring. For performing the top-down attempt, external developers and experts were hired, whose salaries have been included in its cost.

The bottom-up approach was virtually performed without costs, however the legacy code was not removed, meaning that in the future there will be two software layers to deal with. Instead, though the second attempt was more expensive, it may be indirectly helping to reduce other costs, namely transactions maintenance and payments for processing power, since by not removing the transactions' load from the mainframe (only 6 of them represented the 56% of total mainframe load) the agency would had to hire more processing power at a cost of u$s 2 millions a year. In other words, the agency saved the money invested in the second migration.

By basing on this reported experience software engineers and particularly project managers should ponder that bottom-up is inherently better than their counterparts regarding time-to-market, due this approach bases on decorating the legacy system by means of tool-supported methods. However, a top-down approach may achieve better SOA designs and eventually allows an organization to modernize legacy platforms. Table 3 provides a checklist for those practitioners wondering which migration approach should be followed according to certain requirements, which were extrapolated from this experience.

Recently, *meet-in-the-middle* methods like the one supported by the NACA project[5] have been explored. These methods attempt to bring out and combine the best of top-down and bottom-up approaches. Therefore, we believe that in the near future this topic will undoubtedly require more research.

[5] `http://code.google.com/p/naca/`

Acknowledgments

We acknowledge the financial support provided by ANPCyT through grant PAE-PICT 2007-02311.

References

[1] N. Gold, A. Mohan, C. Knight, and M. Munro, "Understanding Service-oriented software," *Software*, vol. 21, no. 2, pp. 71–77, 2004.

[2] S.-H. Li, S.-M. Huang, D. C. Yen, and C.-C. Chang, "Migrating legacy information systems to Web Services architecture," *Journal of Database Management*, vol. 18, no. 4, pp. 1–25, 2007.

[3] A. De Lucia, R. Francese, G. Scanniello, and G. Tortora, "Developing legacy system migration methods and tools for technology transfer," *Software Practice Experience*, vol. 38, no. 13, pp. 1333–1364, 2008.

[4] M. Colosimo, A. D. Lucia, G. Scanniello, and G. Tortora, "Evaluating legacy system migration technologies through empirical studies," *Information and Software Technology*, vol. 51, no. 2, pp. 433–447, 2009.

[5] M. Crasso, J. M. Rodriguez, A. Zunino, and M. Campo, "Revising WSDL documents: Why and how," *Internet Computing*, vol. 14, no. 5, pp. 30–38, 2010.

[6] J. M. Rodriguez, M. Crasso, A. Zunino, and M. Campo, "Improving Web Service descriptions for effective service discovery," *Science of Computer Programming*, vol. 75, no. 11, pp. 1001–1021, 2010.

[7] M. Papazoglou and W.-J. van den Heuvel, "Service-oriented design and development methodology," *International Journal of Web Engineering and Technology*, vol. 2, no. 4, pp. 412–442, 2006.

[8] G. Lewis, E. Morris, and D. Smith, "Analyzing the reuse potential of migrating legacy components to a Service-Oriented Architecture," in *Software Maintenance and Reengineering*, pp. 15–23, IEEE Computer Society, 2006.

[9] S. Alahmari, E. Zaluska, and D. D. Roure, "A service identification framework for legacy system migration into SOA," in *International Conference on Services Computing*, vol. 0, pp. 614–617, IEEE Computer Society, 2010.

[10] E. Adar, "GUESS: A language and interface for graph exploration," in *Conference on Human Factors in Computing Systems*, pp. 791–800, ACM Press, 2006.

Juan Manuel Rodriguez (`http://www.exa.unicen.edu.ar/~jmrodri`) is working on his Ph.D. Thesis about quality of Web Services APIs at the UNICEN, being founded by the Argentinian National Council for Scientific and Technical Research (CONICET) and working under the supervision of Alejandro Zunino and Marcelo Campo. He holds a Systems Engineer degree from the UNICEN. He is a member of ISISTAN Research Institute.

Marco Crasso (`http://www.exa.unicen.edu.ar/~mcrasso`) received a Ph.D. degree in Computer Science from the UNICEN in 2010. He is a member of the ISISTAN and the CONICET. His research interests include Web Service discovery and programming models for SOA.

Cristian Mateos (`http://www.exa.unicen.edu.ar/~cmateos`) received a Ph.D. degree in Computer Science from the UNICEN, in 2008, and his M.Sc. in Systems Engineering in 2005. He is a full time Teacher Assistant at the UNICEN and member of the ISISTAN and the CONICET. He is interested in parallel/distributed programming, Grid middlewares and Service-oriented Computing.

Alejandro Zunino (http://www.exa.unicen.edu.ar/~azunino) received a Ph.D. degree in Computer Science from the UNICEN, in 2003, and his M.Sc. in Systems Engineering in 2000. He is a full Adjunct Professor at UNICEN and member of the ISISTAN and the CONICET. His research areas are Grid computing, Service-oriented computing, Semantic Web Services and mobile agents.

Marcelo Campo (http://www.exa.unicen.edu.ar/~mcampo) received a Ph.D. degree in Computer Science from the Universidade Federal do Rio Grande do Sul, Brazil, in 1997. He is a full Associate Professor at the UNICEN, Head of the ISISTAN, and member of the CONICET. His research interests include intelligent aided software engineering, software architecture and frameworks.