5 A Funny Thing Happened on the Way to SOA: Insights from a Three-Year Experience with a Telecom Company

Paulo Rupino da Cunha (CISUC, Department of Informatics Engineering, University of Coimbra, Portugal), Paulo Melo (INESC Coimbra and University of Coimbra, Portugal) and Catarina Ferreira da Silva (CISUC, University of Coimbra, Portugal)

Abstract

We describe three challenges that a major telecom company faced on its way to SOA after the traditional first step of exposing legacy functionality and orchestrating it in higher level business processes. We show how the accepted practice of gradual migration can cause the materialization of a SOA-related “recent-legacy” that may constrain future architecture evolution; we discuss how newly gained agility in changing business processes can have complex consequences in managing the lower level XML involved in interactions with the leveraged “old-legacy”; and, finally, we address an emerging problem of finding the right services to build new business processes in a pool that steadily increases as the migration to SOA progresses. Our solutions are overviewed, and a set of lessons is compiled to raise awareness of potential pitfalls when entering more advanced stages of SOA evolution.

5.1 Introduction and Case Study Background

Competitive markets constantly pressure companies to become more agile, forcing them to leverage their legacy systems in novel ways: new business processes must be designed and quickly deployed to support new products or services; existing ones need to be reengineered or frequently tuned to keep up with best practices in the industry. The need to cope with these constant reconfigurations of existing and new functionality is one of the main drivers for the evolution towards service-oriented architectures (SOA) (Josuttis 2007). However, this migration is complex: IT economics dictate that the investments in existing systems must be preserved, thus discarding greenfield approaches. On the other hand, the risk of causing disruptions in customer service while modifying those systems must be minimized. These constraints usually lead to incremental evolution scenarios. We describe facets of one such project in a major telecommunications company with emphasis on the impact on business processes.

Advances in technology enabled the once independent fixed telephony, Internet access, and TV services to converge into a unified “triple-play” offering supported by IP (Internet Protocol) networks. This move, however, also meant that some of the once independent and heterogeneous systems that supported each of those original services would have to cooperate to enable the new business processes required for provisioning and overall management of the new unified product.

We have been working with the Operations Support Systems (OSS) Department of a telecom company in such a project. This group handles the systems used for service provisioning, including maintaining the network inventory, configuring resources, and monitoring operations. Several internally developed three-tier applications already existed, and the challenge is to migrate to a more flexible architecture that is capable of keeping up with the dynamics of the market in a world of convergence of telecom services.
The initial step was to build adapters that knew how to interact with each specific legacy application and expose required pieces of business logic, as web services, on an as-needed basis. The new business processes were then written as orchestrations of those web services using Business Process Execution Language (BPEL) to promote the required flexibility in changing them or adding new ones.

Having successfully completed this first stage of the paradigm shift from stovepipe applications to SOA, fresh challenges emerged. We will describe three additional steps in the journey of this company:

- the upgrade of SOA components used in the first stage, which revealed that performing incremental migrations introduces a new layer of “recent-legacy” that may constrain future evolution
- the development of a process design-time application to help cope with the complexity of communicating with the legacy systems using XML documents and associated validations and transformations
- the exploration of semantic technologies to assist business analysts and software engineers in dealing with an increasing number of services from diverse sources that are used to compose business processes

The remainder of this paper is organized as follows: we start by delving deeper into these three challenges to raise awareness of these potential pitfalls. We then present an overview of our solutions for each of those situations. In the last section, we talk about the lessons that we have learned so far, just before concluding with some remarks about future work.

### 5.2 Three Challenges on the Way to SOA

Each of the three challenges is described in a separate sub-section. First, we address the inadvertent introduction of a SOA-related legacy; second, we discuss how modifying high-level business processes may not be as simple as commonly advertised; and finally, we move to our present concern of making sure the business analyst does not become overwhelmed by hundreds of services when trying to design new business processes or reengineer existing ones.

#### 5.2.1 The Unexpected “Recent-Legacy”

The migration to SOA entails dealing with existing legacy systems. However, since that migration is a lengthy process, a second layer of “recent-legacy” may appear: one consisting of SOA tools used early in the project, but that need to be replaced due to technical reasons (such as inability to handle an increasing load) or commercial decisions by the vendors (such as sunsetting of a product). In this case, a rip-and-replace strategy may not be feasible; namely, when the tool is supporting business processes lasting for months or years and the migration of data is highly risky or impossible. In fact, in the telecommunications industry, some business processes have long life spans. Consider, for instance, the case of a contract between a telecommunications company and a property developer, by which the future buyers of the apartments to be built will have special conditions for cable TV and broadband Internet access for a specified period of time. The business process supporting that type of contract is initiated when the building construction is started, but it will terminate a few years later, when the new owners move in and eventually decide to take
advantage of the promotion. If, at some point in time, the telecom company decides to change how such contracts are handled, those already initiated must be honored by the previous rules. However, waiting for all the “running” processes to end before starting the new procedure is unfeasible—it would mean stopping making contracts until the new rules were in place (to avoid constantly pushing forward the end of the last contract) and then waiting for months or years for the last process to conclude. This would defeat the business agility that SOA makes possible. Much like this scenario of modifying long-running business processes, if the infrastructure supporting them needs to be changed, the same problem emerges of allowing those already running processes to complete.

These systems introduced in the first stage of an evolution to SOA, which must be kept running because of established long-running processes, are examples of what we call “recent-legacy.” Our case at the telecom company exhibited this architectural evolution challenge. The existing BPEL engine was becoming unreliable and unable to handle the increasing load, but some of the deployed business processes were long-running and could not be migrated—there were a lot of unknowns regarding how the engine stored its internal state, and some might have pending callbacks from external systems. A solution had to be found to enable a new engine and the old (the “recent-legacy”) to coexist until all processes in the latter concluded. This effectively constrained the path to SOA, forcing the adoption of an intermediate pattern.

5.2.2 Being Flexible Can Be Hard

The ease of modification of business processes is one of the great selling points and promises of SOA. In practice, at a high-level, this procedure often entails modifying invoked services or adding new parameters to be handled. At a lower level, however, these changes may ripple throughout all communication between the process and its building blocks—the services exposed from the stovepipes. In our case, an initial XML document was supplied to the business process and used to collect information as it was passed along the various services involved in provisioning a telecom product. Changing the data contained or expected in the initial XML meant that validations at every service entry (using XML Schema Definitions (XSD) (van der Vlist 2002)) and all required modifications throughout the process path (using Extensible Stylesheet Language Transformations (XSLT) (Clark 1999)) would have to be revised, and these many documents changed accordingly to account for that new piece of data. Manually changing these XSD and XSLT documents is an arduous and repetitive task, and is also prone to errors that are hard to detect until the process is in operation. If the flexibility in high-level business process change is to be attained, such tasks must be supported by technology that prevents (or at least flags) the errors and decreases the overall complexity of making changes.

5.2.3 Services Everywhere: How to Find the Suitable One?

Currently, we face a third challenge stemming from the rise in the number of available services, caused by the normal evolution towards SOA. Although registries or repositories are frequently pointed out as the solution to list which services are available and how to invoke them, when composing new processes, this syntax-oriented mechanism is of decreasing usefulness. In this situation, the analyst needs to be able to query the existing pool of services in terms of semantics to find perfect or close matches to desired business logic, regardless of the actual service names. So, more than just list the services and how to invoke them, we need information about their meaning in the business context. It is not feasible to sift through hundreds of services, reading
each description every time a new business process needs to be created. A mere name-oriented (syntactic) search is not reliable due to inconsistencies and biases towards engineering terms.

5.3 Addressing the Challenges

Following the same format used in Section 5.2, we now present an overview of our solutions to each of the three challenges we previously described. First we show how we mitigated the effects of the “recent-legacy,” then we show how automation was used to lessen the low-level effects of changes in business processes, and finally we explain our current strategy of adding semantics to the emerging SOA. The first couple of solutions were implemented in two separate projects that totaled an effort of eight person-years, and included the development of two systems comprising around 50,000 lines of Java code. The system for addressing the third challenge is still under development. Five autonomous telecom “legacy” systems were involved in these projects.

5.3.1 Versioning—Outside the Box

To cope with the “recent legacy” problem, we created a self-contained component that implements the versioning usually required by long-running processes. Although some BPEL engines support this feature internally, we needed to manage this as an independent capability, such that we could seamlessly handle both the old and new BPEL engines. Additionally, callbacks from external services also needed to be routed to the appropriate engine in a transparent fashion. Using this approach, processes already deployed to the engine being retired can still be accessed until they terminate, but new versions are created in its new replacement.

Versioning must be transparent for all involved. Business analysts should not be concerned with how IT ensures co-existence of “old” and “new” versions of business processes. External users of the processes should not have to modify their systems every time a new version is deployed. As a result, deployments and invocations should only need the original process name, regardless of how many of its versions are already running and how many BPEL engines are in operation. The versioning system must thus ensure that new deployments do not overwrite existing (older) ones, and that requests to business processes are always forwarded to the correct version in the correct engine.

Our component consists of two modules: the Deployer and the Gateway. The Deployer intercepts deployments from the business process editor and modifies the data to add and track a version number. The Gateway intercepts all external communication with the business processes to ensure that it is always forwarded to the correct engine and version. In both cases, the modules act transparently to the involved parties. All the data about the BPEL processes under version control is kept in a database, to ensure persistence in case of planned or unplanned reboots.

As Figure 10 shows, the original BPEL process sent by the editor is intercepted by the Deployer, which determines the next version and modifies the name accordingly before letting the transaction proceed. The answer returned by the engine is then passed back to the editor, whose user is unaware of this intermediate processing. Also, from the point of view of the engine, all this occurs as if just another independent BPEL process had been deployed. The information about the mapping of the original file (in the editor) to the multiple file versions (inside the engines) resides in the versioning system only. To conclude, the Deployer notifies the Gateway to refresh its information about the processes currently deployed.
From the moment a business process is installed in a BPEL engine, it becomes available for instantiation and interaction with other systems. As in previous versions, the client only needs to know the original process name, and the Gateway module will take care of all necessary intermediation to ensure communication is done with the correct version, in the correct engine, as shown in Figure 11.

The Gateway works by intercepting invocations to the processes, finding out for which version any given process is intended and in which engine it resides, calling it, and then returning the answer to the original caller. The complexity resides in ensuring that messages that arrive in response to an interaction initiated by an older version of a process are delivered to that specific instance and version, rather than to the most current one. This is achieved by tracking and manage-
ing BPEL’s correlation IDs, which are frequently used, and sometimes required, in asynchronous calls.

5.3.2 A Little Data Orientation in a Control-Oriented World

The general problem of maintaining the validation (XSD) and transformation (XSLT) documents required to communicate with the web services exposed from the legacy applications can be a hard one (since it must include any kind of processing required to create the actual mappings), but can be made a little easier by observing the patterns of data usage on the processes involving them. Although BPEL is a control-oriented language, the processes in our case were actually data-oriented: the flow was designed to integrate data provided by the various operations support systems as a process advanced from one step to the next towards completion.

The scenario we found was that the process orchestrator sequentially invoked each stovepipe adapter web service, passing it the information it needed and receiving in turn information containing the changes it had made or eventual “new” information, as shown in Figure 12.

In such a scenario, only a subset of the capabilities of the XSDs and XSLTs is required. With this simplification, it becomes feasible to automatically generate or validate these documents every time the interactions with the legacy adapters are changed.

To create a system that allowed such generation, we had to stop thinking of the processes as control flows of services and start viewing them as data flows.

![Figure 12 Process Structure Describing Adapters Orchestration](image-url)
We built an application that allowed us to describe how each data element was modified by different operations (services) as it “traveled” from the beginning to the end of the business process. We accounted for multiple possible origins and also for the fact that paths were not linear, since the same data element could be delivered to different legacy systems. Finally, since the same data element could be used in different ways in each stovepipe application, we had to allow it to be named (and placed in the XML tree) according to the invoked service or step and according to the constraints or default values required by that step.

Once the decision was made to focus on the data flow for each component, a system to help describe those components could be built. That system could in turn be used to generate the required XSD and XSLT documents. This design-time system has a Graphical User Interface (GUI) that allows the definition of the data path for each information component and the aggregation of simple data in structures on each application invocation.

However, even with the system in place, it was obvious that even for processes with the simple structure that was required, inputting the information describing the data processing in each step required significant domain knowledge. This included knowledge on which services are present and on the data used by the stovepipe applications exposed via adapters (which data and in which format). Rather than forcing common semantics, most often the adapters just exposed the internal data and process mechanisms used by the stovepipe application. We tried to lessen the burden by providing capabilities to import previously defined templates, and therefore help to reuse previously codified knowledge about the data.

5.3.3 Enhance Web Services Visibility with Semantic Technologies

To not overwhelm a business analyst with a huge number of available web services, which are usually developed and described using the WSDL, Web Services Description Language (Chinnici, et al. 2007) by different teams, a semantic search—one that goes beyond the mere syntax and into the business meaning—is in order. This can be achieved with appropriate annotations that link concepts in service descriptions to ontological concepts (Guarino 1998).

The solution we are developing explores this avenue by introducing a semantic registry of services, which are described using specific languages, such as WSDL and Semantic Markup for Web Services, OWL-S (Martin, Burstein, et al. 2004), as shown in Figure 13 OWL-S proposes a particular semantic framework within which to characterize the semantics of Web services, by specifying language constructors for service profile, service model—both to characterize the service—and service grounding to bind it, i.e., how to interact with the service. It adds some attributes to WSDL extensions in order to connect both languages. For example, OWL-S specifies parameters that map to WSDL message parts, such as operation, port type, binding, and service constructs. Furthermore, Martin et al. suggest that a WSDL operation can refer to an OWL-S atomic or composite process and a WSDL interface should refer to an instance of a profile class (Martin, Paolucci and Wagner 2007)
Specific SAWSL (Semantic Annotations for WSDL and XML Schema) attributes (Farrell and Lausen 2007), such as MODELREFERENCE, enable the annotation of service descriptions, providing links to explicit concepts that are defined in a suitable ontology. Since none exists at the moment for the telecom domain, we are building a draft from other accepted standards in this world, such as the Shared Information/Data (SID) model (Reilly and Wilmes 2008) from the TeleManagement Forum.

Some experimental semantic registries already exist under open source licenses (Kourtesis and Paraskakis 2008), as do matching engines that can compare ontological concept definitions referenced by SAWSL MODELREFERENCE attributes. OWL-S-specific engines can also detect similarities between services defined in this language.

Assembling these components together with some additional logic, we can provide the business analyst with a Google-like user interface to a semantic search that takes a business-level description of what he or she needs and returns the most probable matches available in the registry sorted by similarity. Additional details may include service functionality, invocation information, location, and owner contact. This possibility of querying the available services in terms of business concepts is very valuable to find the right building blocks when designing or reengineering processes.
5.4 Some Lessons, So Far

In our work with the telecom company, we have learned several lessons regarding the evolution to SOA. A few of them are not new, but are worth restating. Surprisingly, we also found out that following some accepted practices may lead to new problems.

Moving to SOA frequently involves understanding and adapting existing “stovepipe” applications to expose some services. Their complex business logic would be hard to duplicate and difficult to debug if implemented “from scratch.” To do so would also require infeasible time spans, large investments, and great risk.

Risk is also present in “service-enabling” these current mission-critical systems, so the transition to a future SOA should be gradual. Mistakes in this step may potentially affect a large number of customers. Unexpectedly, this incremental evolution may create challenges of its own. In fact, in our case, a BPEL engine had to be replaced due to scalability and reliability problems, but running processes could not be migrated to the new product due to their nature, as they were long-running orchestrations of services provided by external systems. A SOA-related “recent-legacy” had been introduced by the gradual migration, and was raising different challenges than the ones presented by the traditional “stovepipes.” Eventually, a “gateway” had to be developed to allow old and new BPEL engines to work in parallel for a few years, until all processes running in the former complete their execution.

Although not exclusively present in SOA migrations, a second situation, involving the sunsetting of a different product by the vendor, indicated the need for tighter control over the evolution of key systems in the company’s portfolio—including those required by the SOA technologies used. Strong-copyleft licenses, like the GNU General Public License (Stallman 2007) fill this requirement better than less-rigid ones. For example, the overall solution proposed for the company included a GPL-licensed BPEL engine and an editor that was simply free-of-charge under vendor terms. After a couple of years, the GPL-licensed product is still available, but the freeware component has become a commercial product. In spite of its predictability, however, use of GPL code requires a strong discipline. Care must be taken to isolate its components in a way that the fast evolution of the open-source world doesn’t keep forcing the company into constant upgrades. Such isolation also helps in delimiting which parts of the company’s code base must be shared with the community, according to the license terms.

The process of “service-enabling” the legacy also provided some learning points. Usually, the easiest way to expose functionality of each existing application is to create a “fat” adapter that allows access to its full capabilities. However, while such solutions may allow for faster integration of the application in new “SOA enabled” processes, it also makes those processes harder to build, since those adapters tend to do more than just expose core functionality: they often also expose the original application’s internal structure and logic. This can happen, for example, by requiring the invocation of capabilities to be done in a certain order, or by forcing it to include additional information, or to have its data presented in a particular way. Moreover, when integrating different stovepipes whose services have been exposed in this way, one usually discovers that the logic and requirements of different applications are extremely diverse, preventing common solutions. This increases both the cost of creating and maintaining the adapters, and also the sharing of information among the different applications.
Finally, using the services originating from several vertical legacy systems in transversal, end-to-end, long-running processes, may raise a new problem: the need for rollbacks or cancelations. In a typical telecom provisioning process, for instance, several network resources have to be reserved in order: physical circuits, logical configurations, field teams. These are usually handled by different legacy applications. If one of these resources is unavailable, then the ones already acquired must be relinquished. This can be hard to implement, since it is not usually supported by the original legacy systems.

5.5 Future Work

The introduction of the semantic registry/repository poses new socio-technical challenges. There are normal governance considerations, such as the attribution of decision rights to add, change, and retire services, but also specific issues regarding how the services are semantically described and by whom. The reference ontology has to be agreed on and maintained. Decisions have to be made on whether developers become responsible for annotating their services, or if a dedicated team should be constituted. Resistance to change may arise, so reward mechanisms should be discussed in tandem with the technological solutions to ensure effectiveness.

On a different level, although our present experiments focus on the use of the semantic registry/repository at process design-time, its use for runtime dynamic discovery and binding is being considered in the longer term. In that case, selecting from multiple matches returned by a service search is not trivial. A rules mechanism must be able to account for both technical and business concerns, such as simultaneous interoperation of interdependent services, quality of service, and cost. Processing performance of this kind of computation will also be an issue.

5.6 Acknowledgements

The authors are indebted to the reviewers for their comprehensive comments and suggestions.
References


