

CASE STUDY:

SOA FOR CRM-INTEGRATION AT T-MOBILE

Günther Budzick

IT CRM

T-Mobile, guenther.budzick@t-mobile.de

Thomas Dupré

IT CRM

T-Mobile, thomas.dupre@t-mobile.de

Abstract

The mobile phone market is rapidly changing and highly market driven. You have to realize business processes spread over different departments, different brands, different national countries, and B2B partners. For all these scenarios SOA was and is the key technology enabler, which is used to get a significant market advantage. Currently T-Mobile has a SOA landscape with up to 10 million service calls per day. We are switching from a first generation infrastructure approach using different technologies in different nations to a consolidated second generation infrastructure. From a business point of view this infrastructure is used to harmonize business processes across different front-end channels, This paper will describe the key aspects of this approach and the major lessons learned.

Keywords

SOA, distributes business processes, CRM, BPEL, T-Mobile, Integration, order management, ESB, Web Services.

History and Motivation

Around 2000, after a phase of fast growth, it became more and more clear that T-Mobile needed a general approach to be able to realize business processes distributed over several internal and external systems and business units. For example, if a customer wants to prolongate a mobile phone contract, we have to update data in different back-end systems (such as a CRM and a billing system) and perform the shipment of a new mobile phone for the prolongation. This shipment is handled by an internal logistics system as well as by major external shipping companies such as DHL or FedEx.

The initial approach was to realize individual SOA landscapes in different national companies of T-Mobile while using the same general EI infrastructure

(in this case Tibco Rendezvous). However, it turned out that specifying a common infrastructure did not by itself provide sufficient international interoperability, because this infrastructure was used in different ways both from a technical and organizational point of view. For this reason, in 2006 a project for a "SOA Backplane Program" was started to harmonize the ESB infrastructure of the whole company for better maintenance, flexibility, and time to market.

In parallel with the technical consolidation in the past years there were several domain specific approaches to harmonize and consolidate business behavior. In principle, the goal was to have an architecture where multiple front-ends can start a business process, which is then processed using the SOA approach to call orchestrated basic services provided by different back-ends.

A fundamental part of this business process consolidation refers to the fulfillment of orders. Different front-ends (such as a web portal, a call center, a B2B gateway for resellers, or a mobile phone) can use this system to execute or fulfill a customer order such as an address change or a contract prolongation. Technically, the fulfillment is performed by a BPEL engine, which has an additional order management layer.

Currently, the situation is as follows: We have about 500 business services (each version counts) running in production with up to 10 million service calls per day. For example, whenever a customer calls into the call center his/her profile data is loaded via back-end services into the call center application. In addition, both external and internal front-ends use a central offering system to call services that provide specific offers for the specific customer. Most of these calls are currently handled by the first generation infrastructure, while we are moving step-by-step to the new harmonized infrastructure. This move is

triggered by a step-wise enabling of harmonized order fulfillment via the new orchestrated process services. Currently, about 10% of the overall offered services are using the new infrastructure.

Of course, there are many issues we could talk about in this case study, so we decided to focus on the topics that will demonstrate both fundamental technical and organizational aspects regarding the realization of our SOA landscape and its excellence.

Organizational Changes

A first step of the SOA approach was to reshape the organizational structure of the company according to future needs.

As Melvin Conway formulated in 1968 (see [Conway68]):

Organizations, which design systems, are constrained to produce designs, which are copies of the communication structures of these organizations.

That means, system structures typically match with organizational structures. And in fact T-Mobile formerly was organized in such a way that each team was responsible for a specific scope of functions (which was important to get things done during the fast initial growth of mobile phone companies). As a result each team usually provided an isolated system including back-end and front-end layer to solve the task. Over time, when new requirements were integrated into these systems (giving each task to a particular team) the systems grew and became what we call silos or monolithic systems (see Fig. 1).

As the growth of the market decreases and loses its dynamics we feel the need to increasingly invest in the retention of customers. This involves discovering their personal needs and wishes, providing them with new offers, giving away incentives etc. Thus more and more systems apart from just CRM and Billing come into play and requirements tend to have a broader impact on this system landscape as a whole. As a result, these days new requirements usually have an impact on multiple existing systems and domains. Consequently, it is more and more difficult to find a “natural” major system because each system is playing only a minor part in the whole implementation process. In addition, to improve the quality of front-ends in a way that end users get one consolidated view we have to integrate all the different front-ends that departments had implemented for their specific solutions. Especially internal users such as call center agents had to use too many different clients to perform their tasks.

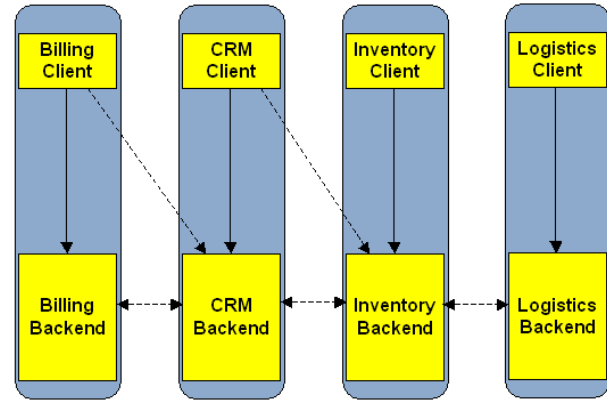


Fig. 1: Traditional departments and system

For this reason, we promoted an organizational separation of front-ends and back-ends. While back-ends have the role of providing core business functionality, it is the task of a front-end to present this distributed functionality in an integrated way that is appropriate for its specific kind of users and technology.

However, this separation was not enough, because we have business functionality that is orchestrated over multiple back-ends but common for different front-ends. In addition, we have to deal with orchestrated services that provide some common functionality affecting different back-ends for multiple front-ends. Conceptually, this is still back-end functionality but there is no specific single back-end that provides it as core functionality or core data access (see the section about order consolidation later for an example). For this reason, we introduced a third type of systems that are responsible for cross-domain back-end functionality.

As a result, we were changing our organizational structures according to Figure 2:

- The (classical) back-end systems and departments provide fundamental core functionality and data access with regard to their specific domain.
- Cross-domain back-end systems and departments provide functionality that is orchestrated to access different back-ends.
- Front-ends deal with the requirements for a specific channel to the consumer, customer, or partner.

Note that these three categories “only” cover the structure of the system landscape in operation and the organizational units responsible to maintain them. In order to be prepared to deal with new requirements that have cross system (and cross department) impact

T-Mobile added another organizational category of departments that deal with “solutions.” Their task is to run the projects that realize new cross-system requirements. The task starts with a high-level or solution design to determine the impact of a requirement on existing systems and departments. The solution manager is responsible for bringing this new functionality into production, which we would like to explain in a bit more detail.

1. The process usually starts with marketing thinking about a new product or controlling or incident management raising the need for a process optimization.
2. A solution manager gets the task of producing a high-level design and finding out how much the new feature or optimization will cost.

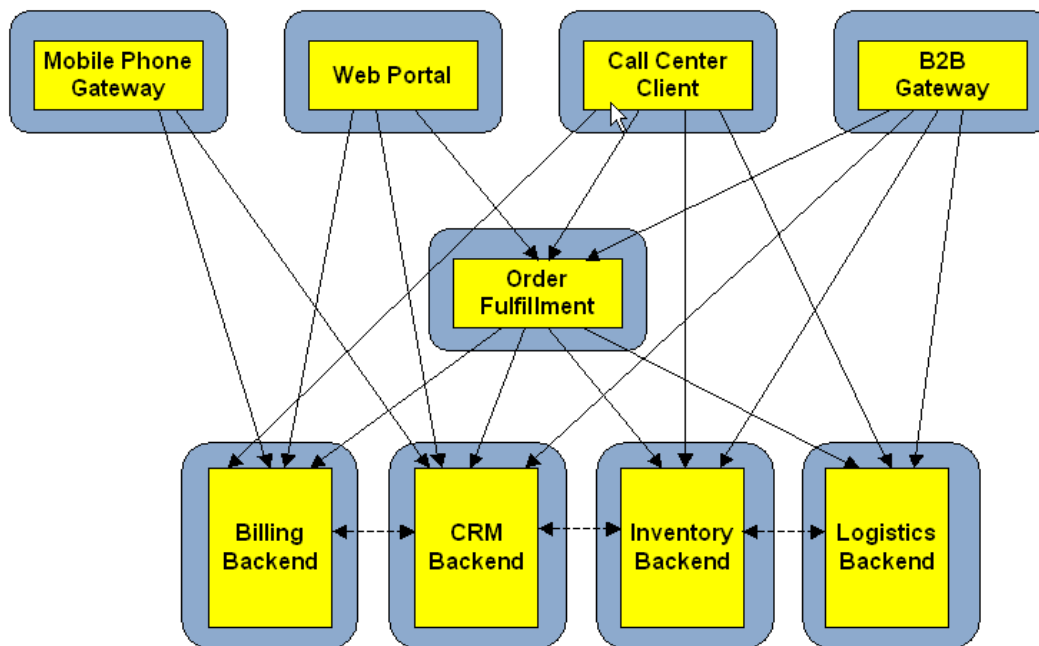


Figure 2: Separation of Front-ends and Back-ends

3. Then, the solution managers, business representatives, and the affected departments come together to decide which requirements are realized next. That is they make a release plan. The goal is to maximize the benefits for the company as a whole at the lowest cost. The departments usually have budgets for each release, and the release plan should ideally use exactly 100 percent of each departments’ available resources. That is, the goal of the release planning is to give each team tasks that will occupy 100 percent of their development (and test) time, and to plan the tasks such that they will create the best possible benefit.
4. During the realization the solution manager has to track progress, deal with problems and with requirement modifications. Ideally, the task is done, when the solution is in operation.

Of course, in practice things are a bit more complicated. For example, developers never have 100 percent of their time free for development. In

addition, you need time to verify that existing functionality is not broken. In fact, only the first half of the allotted development time is typically reserved for the development of new functionality. The second half is for integration and bug fixing.

Last but not least, you must be aware that all your planning can become worthless if priorities change from a business point of view (due to market opportunities, or e.g. if a competitor suddenly introduces a new feature). Interestingly, with the new organizational structures we are able to react fast because we have established appropriate processes to realize distributed functionality in parallel. Distributed development also means distributed resources that help to develop solutions. Thus, having the appropriate organizational structure can become a big market advantage.

Note that these organizational modifications demonstrate that SOA is far more than just a new technology. It’s an IT strategy with important impact on a company as a whole. One reason is that all this

distributed development works only if the people involved are able to collaborate, which among other things requires trust. That is, SOA requires a specific culture in enterprises. This is another reason why SOA is nothing you can realize without top management support.

Infrastructure Evolution

T-Mobile is an international company with mobile phone networks in multiple countries. By nature, in these countries different market constraints, different history and different rules, forms and regulations exist. For this reason, it made sense to introduce different layers inside the ESB. We separate national communication within one country from international

communication where multiple countries are involved. Note that we have

- services with specific individual implementation in a country
- services with one implementation but with different national deployments
- services with one deployment for all countries

For historical reasons we started with five countries (Germany, Netherlands, UK, Czech, and Austria). So we needed an infrastructure that is shown in Figure 3.

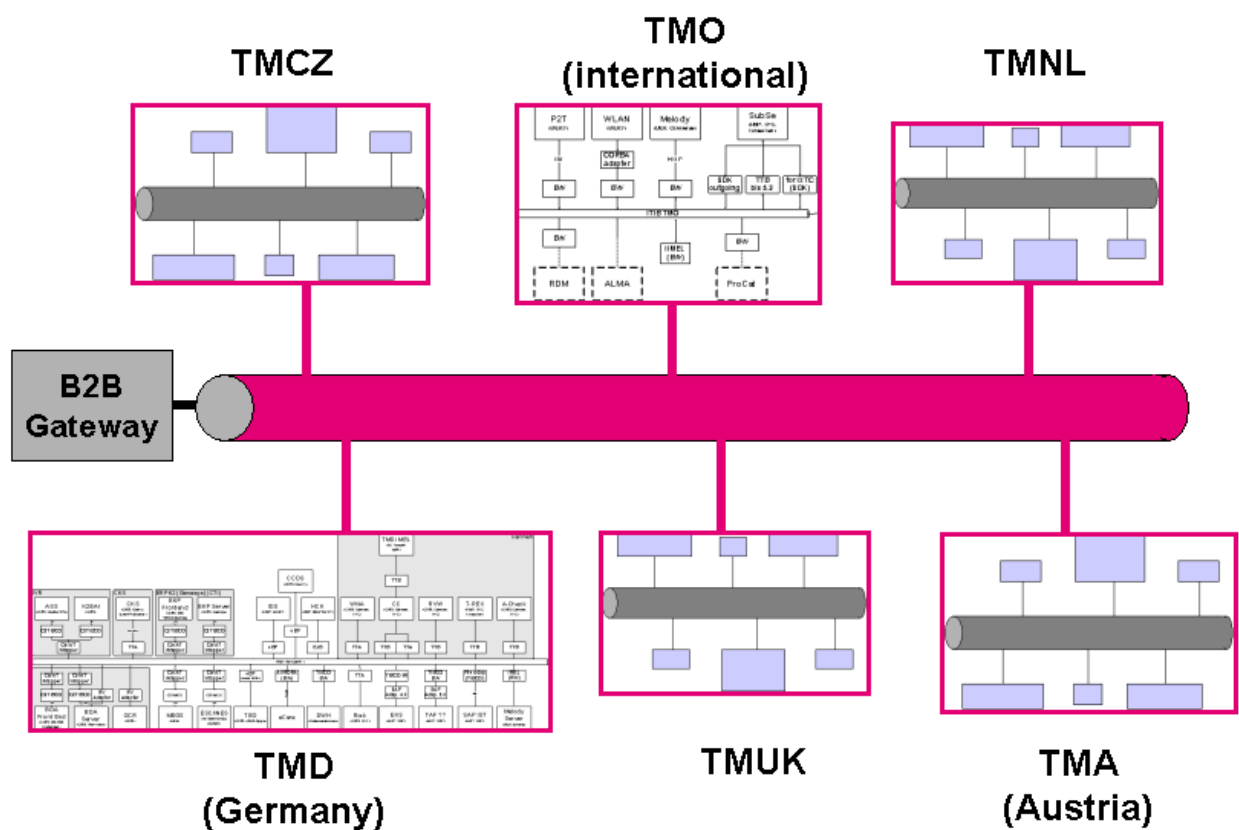


Figure 3: T-Mobile needs and uses different layers of ESBs

As stated above, we started with a Tibco Rendezvous based EI approach which was established in different countries. Because we didn't harmonize ESB aspects in detail, it turned out that we needed routers and gateways that were able to mediate between different countries (e.g. we had to map technical data and some differences in the message exchange patterns used). Over time, of course, more and more

requirements for Web Services based interfaces came up. For this reason we implemented a gateway to map between our Tibco based infrastructure and different forms of Web Services (Web Services unfortunately come in different protocol flavors such as "document literal wrapped", see [Butek03]). The resulting ESB was heterogeneous and similar to what is shown in Figure 4.

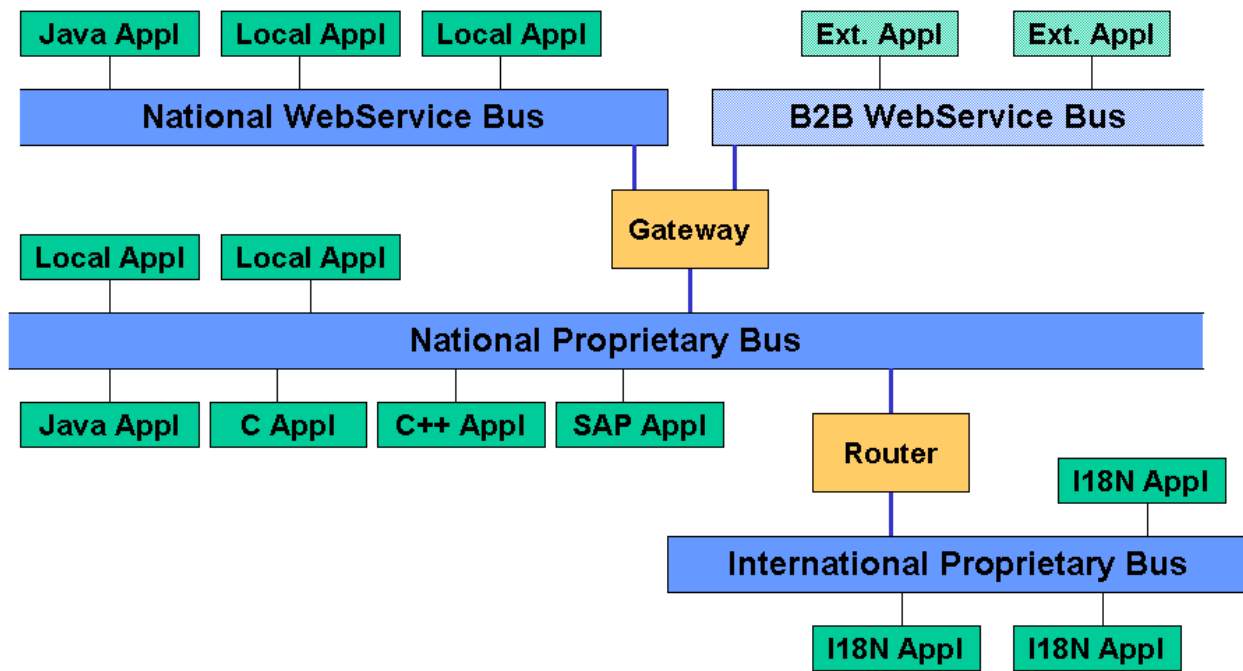


Figure 4: A first heterogeneous ESB grown over years

After some years of operating this kind of infrastructure, we started a project to harmonize and consolidate the different infrastructure technologies to better facilitate and manage business processes distributed across multiple national and international systems.

We decided to use an approach that provided Web Services interfaces to service consumers and providers. However, internally we use interceptors (we call it “common access layer”) to decouple the actual implementation of the ESB from the outside world. With this approach we are able to use any internal ESB technology we like. In fact, internally we use the Tibco-based implementation of JMS, called EMS. Currently we have one central hub-and-spoke server for each country plus one EMS server to transfer service calls between different countries (see Figure 5).

The major advantage of this approach is that we are also able to provide value-added ESB services (note that Web Services inherently do not provide these value-added services due to their point-to-point and protocol-driven nature, see [Josuttis07] for details). For example, we can monitor inside the services bus (including business activity monitoring), are able to

have intelligent routing, and can deal with security issues.

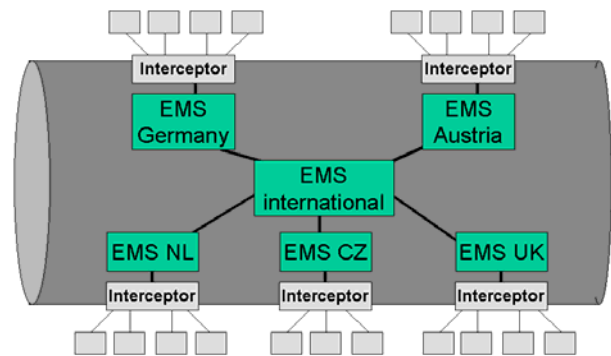


Fig. 5: A consolidated ESB using Web Services APIs with EMS systems inside

The whole infrastructure is deployed using a central registry where each team can specify its local configuration of services provided and consumed. In addition, we can specify different deployment scenarios so that we can deploy different environments for development, testing and production mode.

Several aspects were solved inside the ESB:

- Because the ESB serves as central entry point to track and debug distributed business processes, we need well established ways to log and monitor messages. Apart from general requirements such as to use a specific technical header with correlation IDs or a common error class, we provide tools to retrieve a consolidated output out of the different distributed log files written by all the EMS systems. As a result each team can easily find out whether and how messages were routed inside the service bus (see Figure 6).
- Over time, testing all the different updates and releases turned out to become more and more an issue. In fact, because different systems might have different schedules for upgrades (front-ends deliver a lot more frequently than back-ends) we needed a concept to test any combination of possible distributed system configurations. For this reason we are currently introducing a mechanism to identify specific systems for individual business process tests, which is in fact a way of providing virtual ESBs.

.....T-Mobile

Environment: SOABP_TST2 Logged in as: test | [Change Password](#) | [LogStore](#)

SOA Messages - Selection Criteria

Enterprise Query

Enable Enterprise Query

Target Domains

<input checked="" type="checkbox"/> TMO	<input checked="" type="checkbox"/> TMAT	<input type="checkbox"/> TMCZ
<input type="checkbox"/> TMHU	<input type="checkbox"/> TMNL	<input checked="" type="checkbox"/> TMDE
<input type="checkbox"/> TMSK	<input type="checkbox"/> TMUK	<input type="checkbox"/> All Domains

Query Timeout: seconds

Time Frame [\(Clear\)](#)

Last: Minute(s)

Last: Hours(s)

Specify Date: Time: :

Optional Display Columns

<input checked="" type="checkbox"/> Operation	<input type="checkbox"/> TargetNamespace	<input checked="" type="checkbox"/> Sender	<input type="checkbox"/> ReplyTo	<input checked="" type="checkbox"/> RequestId
<input type="checkbox"/> TimeLeft	<input type="checkbox"/> Priority	<input type="checkbox"/> Target	<input type="checkbox"/> Redelivered	<input type="checkbox"/> Revision

Messages Filter [\(Clear\)](#)

Include Filter	Exclude Filter	Field Name	Field Value
<input checked="" type="radio"/>	<input type="radio"/>	Correlation Id	<input type="text" value="igetAddressData-567123"/> <input type="button" value="⊗"/>
And <input type="radio"/> Or <input type="radio"/>			
<input type="radio"/>	<input type="radio"/>	Sender	<input type="text"/> <input type="button" value="⊗"/>
And <input type="radio"/> Or <input type="radio"/>			
<input type="radio"/>	<input type="radio"/>	Sender	<input type="text"/> <input type="button" value="⊗"/>

Figure 6: User Interface for access to distributed logging

Order Consolidation with Orchestration

Of course, SOA introduction and SOA consolidation is not an end in itself. In addition to the general requirement to be able to connect distributed systems to realize distributed business processes, we also had

the task to harmonize and consolidate our business. In fact, due to the history of silos and monoliths it could happen that different customer channels had slight differences in terms of offers and fulfillment. In addition, it was not possible to start a business process via one channel and track and update it via another channel. For this reason, inside the CRM

department a program was established, which – based on the current SOA infrastructure – aims at harmonizing and consolidating business behavior.

We introduced a central component called OMS/OE which stands for “Order Management System and Orchestration Engine”. As the name implies there are two tasks solved inside this project:

- The Order Management System, OMS, is meant to accept and manage orders from different channels to fulfill them. Note that orders could be a combination of different sub-orders (such as a contract change, a new option, and a bundled piece of hardware). The task of the

OMS involves kicking off processing of such an order and tracking its processing status. It provides a common interface to the front-ends so that they are able to check the state, while the order is in process.

- The orchestration Engine, OE, has the task to actually execute a process or sub-process as requested by the order. Here we use a BPEL engine to orchestrate the service calls to update the different back-ends including error-handling.

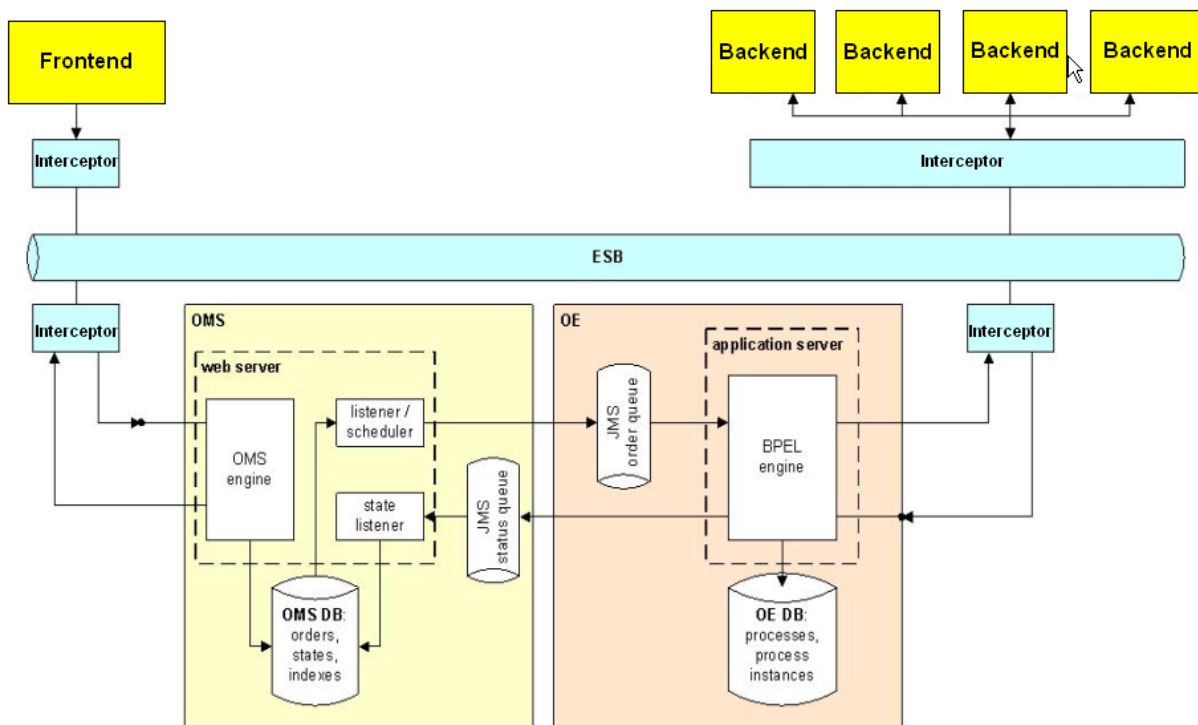


Figure 7: Architecture of Order Management and Fulfillment System

The principle architecture is shown in Figure 7. Of course, the question came up, whether to build or buy an appropriate solution. Especially some vendors offer integrated ordering and orchestration tools (which both provide an order based view and control the fulfillment). The problem we found with those was that a classical static order model does not provide the flexibility required for our order structures. In addition, as we want to be able to sell new products or product bundles on short notice, changes in the order structures are frequent. For this reason, we have the requirement to treat an order as a

flexible aggregation of sub-orders of any type. Because the commercial solutions usually map order structures to relational databases, such changes would result in frequent schema changes.

For this reason we use a more generic approach that manages orders as XML documents. These orders are stored as BLOBS (technically CLOBS) in a local database with their state and data to find and manage them. The interfaces are such as createOrder(), queryOrder(), etc. instead of changeTariff() or updateAdress(). That is, the exact business functionality is a parameter of these generic order

services. It may especially be a combination of different sub-orders (e.g. we could have one order that changes a tariff, changes an address, and enables a special feature or option).

To be able to deal with queries from different channels the orders get enriched by additional attributes such as the customer identity (name, address, phone number). These are used as indexes to be able to retrieve a certain order currently in process from the OMS database, when queried by a front-end.

To start with the execution of an order OMS passes it via an internal JMS interface to the orchestration engine, which actually executes it using a BPEL engine. Here the different sub-orders are executed as BPEL (sub-) processes.

Note however, that we don't model all aspects of the BPEL processes as they are finally executed. Instead, the model of the pure business process is enriched at deployment time with some aspects we don't want to be part of the model. For example, error handling (retries etc.) is needed again and again for all business processes in basically the same way but tends to interfere with the business process model to render it unreadable. So we add this functionality through generation just before the actual deployment (which is a way of aspect-oriented programming).

Beside these local aspects of the realization of order management and orchestration we face some general problems. Many of them have to do with the role and responsibility of the different systems and tools involved. Regarding BPEL for example it is an important question how to combine the "back-end workflow" of services with "front-end workflows" in user interfaces and batch programs.

By nature a running order (process service) can't simply interact with the user. Services are batch programs that usually have the idea to run without human interaction from the beginning to the end. For this reason, front-ends have to (pre)validate input (order data) so that in the usual case the fulfillment is successful. As a result, front-ends use a workflow for the user interface and as part of this workflow services are called to read and validate data to prepare an order (see Figure 8, adapted from [josuttis07]). Validation can also be enforced proactively by giving users only those options that are valid in the current context. The result of the order entry phase is typically a valid order, which is subsequently submitted for execution.

For a clear system design you need a clear understanding of which system is responsible for which functionality. For example you can handle possible tariff options in two different ways:

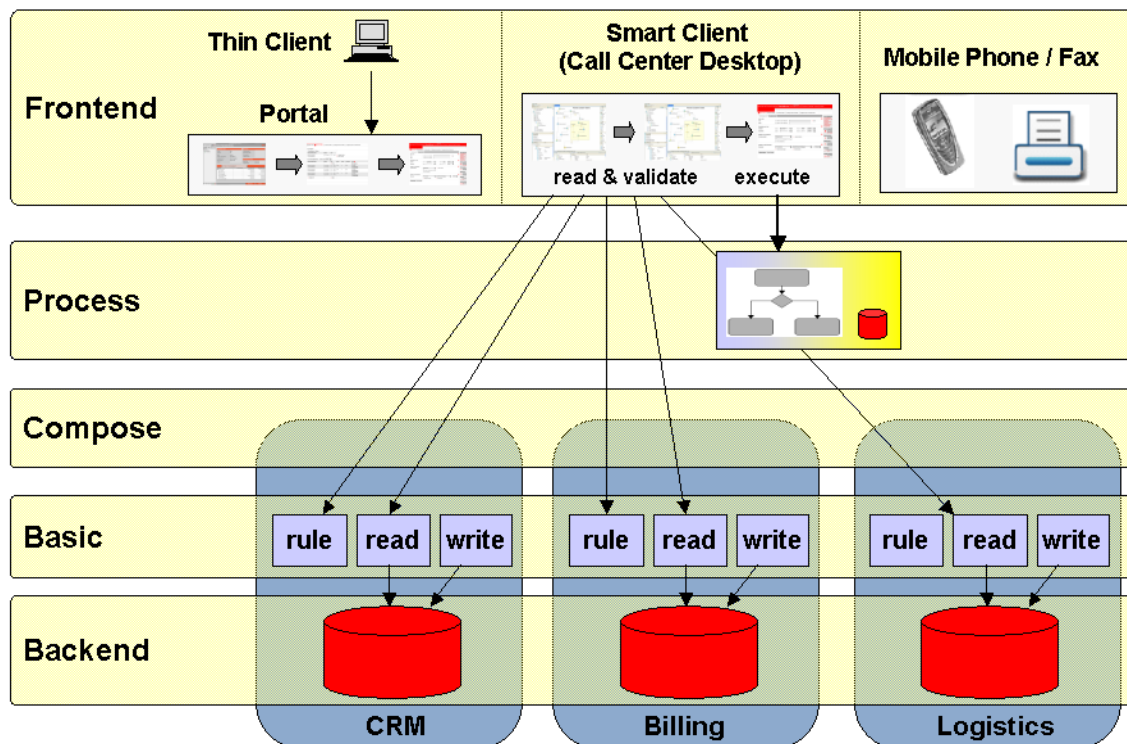


Figure 8: Process-Services deal with prevalidated order for different front-end channels

- A CRM system might provide a service that returns the possible options a concrete customer has in regard to a concrete contract. Internally this service uses other services to process actual business rules and configurations (such as current products we prefer to offer).
- A CRM system “only” returns current customer data and some other system (or the front-end) correlates this data with data from systems that provide business rules and preferred offers.

However, even with all (pre-)validation you can't guarantee that fulfillment is successful. One reason is that you can't prevalidate all aspects of complex business processes in advance because then you'd have to perform the whole business process in advance. In addition, at runtime things can change and fail so that expected behavior doesn't work as expected (a system might fail or some product might not be on sale any more). The general approach (besides canceling a running process) is to bring the process into a state that signals that some interaction is necessary. In this case, front-ends have to reconnect to the running process to deal with the error. For this purpose we have error workplaces. Note that the connection with these workplaces is not standardized yet. While there is a proposal called BPEL4people (see [BPEL4people]) tool vendors usually provide proprietary solutions, which can be used but are not portable.

Lessons Learned

While we are still going on with the establishment of the SOA landscape, things seem to become more and more mature and we see some effects. Nevertheless, we've learned a lot of lessons. The most important are:

- SOA is a strategy that takes years to get established and has an impact on organizational structures and culture. For this reason top management support is a key success factor.
- You can't buy SOA. While tools help, there are several things so solve, install and establish that you can't buy. For example, you have to make architectural decisions, establish policies and processes, and govern the establishment of SOA as a whole.
- Due to the reasons above there is no other way than establishing a SOA landscape step by step. Currently, we implement even some new business functionality with the current technology. At the same time we migrate

existing implementations business process by business process to the new architecture.

- Distribution has a price. It affects the way we design solutions and has a significant impact on realization, test and operation. For example, it is a big task to provide distributed test data so that you can test distributed business processes before bringing them into production mode. In addition, note that your infrastructure (ESB) to some extent becomes your distributed debugger. For this reason you have to be able to log, monitor, track, and correlate messages. Distribution can be a dream if it works; if not, it might become a nightmare.

References

[BPEL4people]: WS-BPEL Extension for People, <http://www-128.ibm.com/developerworks/webservices/library/specification/ws-bpel4people/>

[Butek03]: Russell Butek: Which style of WSDL should I use?, <http://www-128.ibm.com/developerworks/webservices/library/ws-whichwsdl/>

[Conway68]: Melvin E. Conway, How do committees invent?, *Datamation*, 14, 4, April 1968, <http://www.melconway.com/research/committees.html>

[Josuttis07]: N. Josuttis, *SOA in Practice – The Art of Distributed System Design*, O'Reilly 2007

Additional Information

Acknowledgements

Thanks to Nicolai Josuttis, who helped to prepare and provide significant parts of this submission.

Copyright

Copyright © 2008 by T-Mobile Deutschland GmbH, Germany.

T-Mobile and the author(s) grant a non-exclusive license to the Integration Consortium to publish this document in full on the World Wide Web (prime sites and mirrors) and in printed form. Any other usage is prohibited without the express permission of the author(s).