

Building the SOA City – Part II: Programs to Business Processes

By Axel Angeli and Lynton Grice

Editor's Note: We've all been there lately. We make a call to a company—local or national or even international—and the person who answers the phone is sitting in a call center located half way around the world. Yet, somehow, this individual knows the answers to our questions with the same certainty as if they were sitting in the company's headquarters. How does this happen? It's all part of globalization, aided by SOA—Service Oriented Architecture. In Part II of their series, Building the SOA City, Axel Angeli and Lynton Grice take us behind-the-scenes as to how the various components of an SOA are used to streamline business processes and make it possible for someone in New Delhi to have, at their fingertips, the information you need to know when you make that call.

Introduction

Panta rhei – “Nothing is as stable as change”

The purpose of an SOA is to provide a fertile ground to allow for restructuring the IT landscape in light of the Internet, and the future of pervasive computing. SOA is not a goal but rather a strategic master plan toward greater, more powerful, and more efficient IT. SOA will easily free up to ninety percent of current inefficiently-used resources, and use them in other ways to gain greater business benefits. The savings are made through technical means like conversion utilities, a common enterprise service bus, and central repositories that make flexible and agile recombination and reuse of existing programs as easy as possible. But that is not all: SOA will literally introduce a new dimension in computing by presenting the ubiquitous ground for parallel and asynchronous processing as the standard way of assembling algorithms. This new gain in complexity will lead to an outburst of new possibilities in computing.

Pizza Pronto?

If you've recently ordered a pizza to be delivered from a local pizzeria, then you may well have unwittingly become part of a new flavor of both Globalization and Service Oriented Organization. Let us assume you live in Pizzburg and you dial the number of the pizzeria. A nice, polite person picks up the phone and says “Pizza

Pronto, Giovanni speaking, how can I help you?” This person helps you make your choice and notes down your order, delivery address, and payment information. Half an hour later, you call the same number again to ask why the pizza has still not been delivered. “Pizza Pronto, Carla speaking, how can I help you?” Now it is Carla and she quickly checks with the pizza driver, and then advises you that the delivery guy has been held up in a traffic jam but is now only ten minutes away. So far, so conventional. But what happens behind the scenes? Did you ever ask yourself who the person on the other end of the phone line might be, where she or he is actually sitting, and how the communication with the pizza driver takes place? If you are very misanthropic, you may think that your polite helper does not really talk with the driver but simply makes up a story and just treats your case as any other case. The truth may be much more hi-tech than you would imagine. The person that picks up the phone is sitting in one of the better call-centers in Calcutta, India. She (or he) is speaking an accent-free English (or Spanish, German, Polish; wherever the call comes from) and, on the monitor, the menu of the pizza service of your hometown is immediately displayed when the call comes in. Whenever the driver changes a place or gets delayed somewhere, he sends an SMS text message to signal the situation. When you complain about the pizza being late, the town map is displayed and the service agent in Calcutta can immediately tell how far the driver is away from your place.

SOA is not a goal but rather a strategic master plan toward a greater, more powerful, and more efficient IT.

Globalization Needs SOA

This scenario is a result of Globalization. In our case, India takes over local services, with the decisive factor here being the ability to speak the language of the caller perfectly well. That is the human part that is inevitable in any service driven world, in addition to the wide range of technical tools to assist the process.

What does this example have to do with a Service-Oriented Architecture? Mainly the fact that this scenario is so dependent on an agile electronic and computer assisted communication structure that it could not exist without a Service Architecture. Let's do a brief anatomy of the technical landscape of a global call center.

- We'll start with the guy sitting in front of the computer and waiting for the next call to come in. Directly in front of him is a computer that is connected to some kind of Customer Relationship Management (CRM) software. This software will store the name of all the pizza products, some pictures of them, and the latest prices. Here we can find the first challenge. How can we keep the offers and prices up to date? Will there be a way to signal if one of the pizzas is no longer available?
- Luckily, our CRM system is able to link directly to the restaurant's computer system where the restaurant employee can immediately mark a product as "sold out". This works with a SOA. The restaurant's computer needs some form of master data management to be able to identify the current offerings as a service in such a way that the CRM system can access that service in real time. Typically, this would be some sort of Web service that can be invoked easily over the Internet.

This scenario is so dependent on an agile electronic and computer-assisted communication structure that could not exist without a Service Architecture.

- Then, we are somehow connected to software that displays a map of the restaurant's delivery service area. We can simply call the Google maps for that purpose, (another public service in our landscape through a standard Web-service client).
- The driver in Pizzburg has a modern GPS (Global Positioning System) that regularly pings the current position of the driver to a central server from which the CRM system can immediately check the

What to do	Traditional (man days)	SOA (man days)
Building the master data link	10	25
Recognizing telephone caller ids	120	5
Display town map	120	2
Inject SMS messages into CRM	Failed *	5

** This component was never developed, as the necessary intellectual and technical capacity to build an SMS bridge to SAP could not be found.*

Figure 1: Comparing Development Costs Between Traditional and SOA Projects

last reported location. The data is queued in a staging area.

Making Money with SOA: 90% Savings Today

Figure 1 shows a brief calculation of how much money the SOA development approach would have saved us. This is a typical calculation that we made from the estimates in man-days, to compare costs of developing the project traditionally or with an SOA approach. Traditional development would mean that we develop most components end-to-end, on our own.

The remarkable point is the fact that the scenario here had so many technical specialities that it would have been difficult (if not impossible) to bring all the necessary know-how into the project. We also did not consider (in the calculations) the gain in quality with an SOA by sheer reuse of the components. The more frequent a component is used, the less likely it is to contain substantial unknown errors. Traditional components are developed individually for a dedicated task. They are developed, tested, and used exclusively by a single consumer.

Virtualization

Let's look at the component that determines, in real-time, if a certain pizza can be baked or not. We find that we are dealing here with a sort of master data management task for a restaurant. We, in fact, have the most topical master data stored on the restaurant's computer (material data management system) in Pizzburg, and it needs to be synchronized with the CRM system in India.

While this appears to be relatively easy, we should not forget that there is hardly a modern ERP system that can handle such a scenario. Just think of SAP with two independent instances. How nice would it be to keep the material data in one instance and just use them ad hoc in the other instance when they are needed? Figure 2 shows the process as it is today contrasted with a possible future scenario.

This concept of virtualization is an essential building block of SOA. The purpose of an SOA is to restructure the IT landscape in consideration of the Internet and the future of pervasive computing. An SOA requires breaking down existing enterprise applications into small and atomic components that can be consumed by other programs as services. The SOA provides the technical structure for conversion utilities, a common enterprise service bus, and central repositories. This will allow flexible and agile recombination and reuse of existing programs, as easily as possible. However, allowing for reuse requires the serving applications to have certain characteristics. If (and only if) the older applications can expose the essential logical steps of their behavior as an API, then they are ready and well-suited for the SOA structure. That brings us to one of the premises of an SOA.

An SOA requires breaking down existing enterprise applications into small and modular components that can be consumed by other programs as services. In our example of the virtual MARA, it is important that SAP® introduces a virtualization layer for database table access in the ABAP language.

Today you code:

```
SELECT SINGLE * FROM mara WHERE matnr = 'M4711'.
```

This call reads the table MARA from the currently connected database. You could now read data from a remotely installed SAP instance with an RFC call (e.g., through the function module RFC_TABLE_READ) or a suitable BAPI. But this is not satisfactory; after all, wouldn't it require changing the code wherever you accessed MARA directly with a "Select" statement? In addition, coding a "Select" is not only easier, but is also much more flexible than a function module.

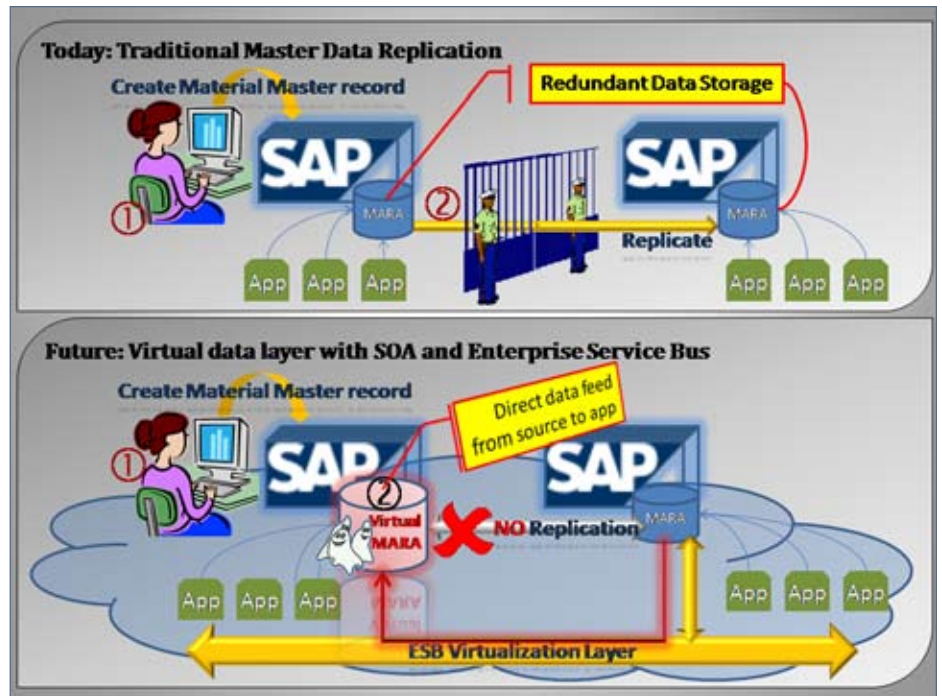


Figure 2: Master Data Management in a Future SOA Landscape

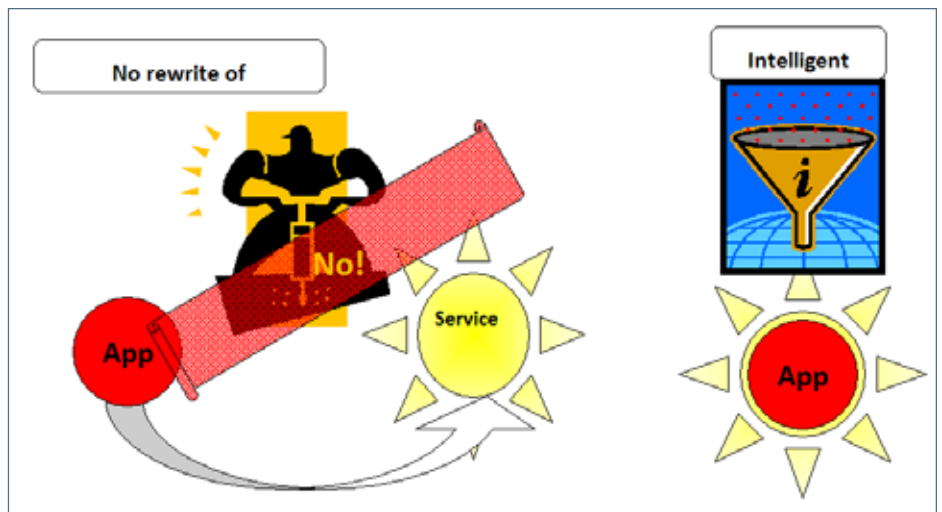


Figure 3: Wrap Rather Than Rewrite the Application

To introduce a virtualization layer into SAP, the correct place would be the ABAP database abstraction layer. This is the place where the ABAP Open SQL statements are translated into real database commands, and all communication with the DB takes place. It would be relatively easy and safe to build functionality that checks whether a table like MARA is located in the current instance, or hosted centrally on another instance as a common shared table. Let's wait and see if SAP comes up with a solution; only recently has Walldorf made a proper U-turn in strategy, with ABAP development once again following the real practical needs. See Figure 3.

Services

When building services for an SOA, we concentrate on building wrappers and proxies to access existing enterprise functionality, and on commonly used utilities, like accessing the ESB elementary components such as email, databases, message queues, and user interfaces. Rewriting an application for SOA may be an option in rare cases, but SOA can only be the trigger to do so. The reason for rewriting a working application can only be that the existing code is insufficient, either in the sense of a lack of functionality or with respect to poor quality. If, however, the program works satisfactorily, it is wiser to build wrappers around it (see Figure 3).

SAP-literate people have used services in the form of BAPIs for quite a long time. They have been designed as a wrapper around the business functionality and the business objects within R/3 that allow access from anywhere. They can be accessed via the RFC protocol, or in later versions of the ABAP kernel (also as Web services) with ease. The SAP ABAP database abstraction layer is also a wrapper that simply acts like a middleman between an existing application and a requesting program. Generally ABAP is a fairly mature SOA application framework, far more mature than most of the competing solutions that are mostly incomplete, or can only act as a bolt-on to an application suite.

Software Development in the SOA City

Thinking in terms of the SOA structure for software development will require a radical change. While we traditionally develop hybrid applications that literally “compile” all the necessary functionality into one single program, SOA requires letting go of this quick and dirty way of writing a program.

Instead, programs need to go and components need to come in. Applications will then be a pure assembly of existing components. The difficulty here is to find a well-balanced compromise of the features a component should possess as a service. If a component is incomplete, it is of no use. If there are too many features, they will not be used.

FUNCTION BAPI_SALESORDER_CREATEFROMDAT2.

```

**-----
***"Lokale Schnittstelle:
** IMPORTING
**   VALUE(SALESDOCUMENTIN) LIKE BAPIVBELN-VBELN OPTIONAL
**   VALUE(ORDER_HEADER_IN) LIKE BAPISDHD1 STRUCTURE BAPISDHD1
**   VALUE(ORDER_HEADER_INX) LIKE BAPISDHD1X STRUCTURE BAPISDHD1X
**   OPTIONAL
**   VALUE(SENDER) LIKE BAPI_SENDER STRUCTURE BAPI_SENDER OPTIONAL
**   VALUE(BINARY_RELATIONSHIPTYPE) LIKE BAPIRELTYP-RELTYP
**   OPTIONAL
**   VALUE(INT_NUMBER_ASSIGNMENT) LIKE BAPIFLAG-BAPIFLAG OPTIONAL
**   VALUE(BEHAVE_WHEN_ERROR) LIKE BAPIFLAG-BAPIFLAG OPTIONAL
**   VALUE(LOGIC_SWITCH) LIKE BAPISDLS STRUCTURE BAPISDLS OPTIONAL
**   VALUE(TESTRUN) LIKE BAPIFLAG-BAPIFLAG OPTIONAL
**   VALUE(CONVERT) LIKE BAPIFLAG-BAPIFLAG DEFAULT SPACE
** EXPORTING
**   VALUE(SALESDOCUMENT) LIKE BAPIVBELN-VBELN
** TABLES
**   RETURN STRUCTURE BAPIRET2 OPTIONAL
**   ORDER_ITEMS_IN STRUCTURE BAPISDITM OPTIONAL
**   ORDER_ITEMS_INX STRUCTURE BAPISDITMX OPTIONAL
**   ORDER_PARTNERS STRUCTURE BAPIPARNR
**   ORDER_SCHEDULES_IN STRUCTURE BAPISCHDL OPTIONAL
**   ORDER_SCHEDULES_INX STRUCTURE BAPISCHDLX OPTIONAL
**   ORDER_CONDITIONS_IN STRUCTURE BAPICOND OPTIONAL
**   ORDER_CONDITIONS_INX STRUCTURE BAPICONDx OPTIONAL
**   ORDER_CFGS_REF STRUCTURE BAPICUCFG OPTIONAL
**   ORDER_CFGS_INST STRUCTURE BAPICUINS OPTIONAL
**   ORDER_CFGS_PART_OF STRUCTURE BAPICUPRT OPTIONAL
**   ORDER_CFGS_VALUE STRUCTURE BAPICUVAL OPTIONAL
**   ORDER_CFGS_BLOB STRUCTURE BAPICUBLB OPTIONAL
**   ORDER_CFGS_VK STRUCTURE BAPICUVK OPTIONAL
**   ORDER_CFGS_REFINST STRUCTURE BAPICUREF OPTIONAL
**   ORDER_CCARD STRUCTURE BAPICCARD OPTIONAL
**   ORDER_TEXT STRUCTURE BAPISDTEXT OPTIONAL
**   ORDER_KEYS STRUCTURE BAPISDKEY OPTIONAL
**   EXTENSIONIN STRUCTURE BAPIPAREX OPTIONAL
**   PARTNERADDRESSES STRUCTURE BAPIADDR1 OPTIONAL
**-----

```

Figure 4: The Sales Order BAPI Suffers Both from Affluence and Missing Functionality

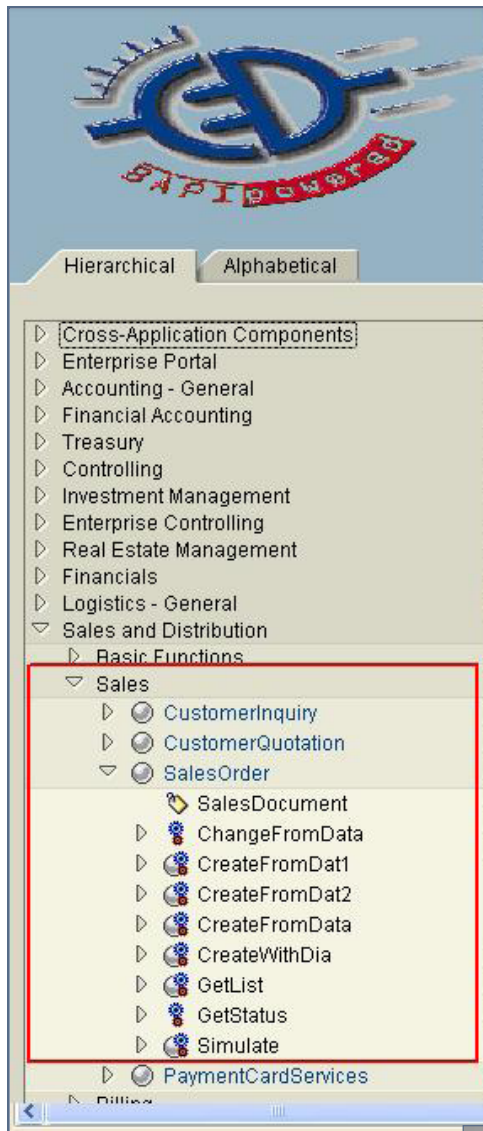


Figure 5: Sales Order BAPI (to Provide a Read Method)

Examples of feature-overloaded components can be found everywhere. Just look again at the sales order BAPI (Figure 4), which has so many parameters that it takes days to understand which ones are mandatory and which are not.

The function module BAPISDORDER_GETDETAILEDLIST does the job partly, but with an incompatible interface, as you can see in Figure 5.

Staging

The focal point of every component development is the interface. Client and server programs need to have a way to exchange data so that both understand the data. While this problem can be solved with mappings, there is a more complex problem when working in a distributed environment like the Internet. Sender and receiver of the data packets may work at different speeds.

If a pizza in the oven is ready, but the waiter is still busy with another client, we cannot leave the pizza in the oven. We need to take the pizza out and put it in a place where the waiter can pick it up as soon as he passes by.

Such a place where we can temporarily store an object (the pizza) is called a staging area. These staging areas are the essential difference between SOA development in a distributed environment, and classical programming. A calling program will no longer talk directly to a receiving component, but simply drop the data container to a staging area where the calling program picks it up (Figure 6).

An example of an incomplete component would be a BAPI that does allow creation of a business object, with no way to modify or read it. Unfortunately SAP neglected this crucial component of the SOA framework in the recent years, so there are still many BAPIs that are incomplete (e.g., the BAPI for sales orders that allows creating sales orders with no compatible function to read the sales order again).

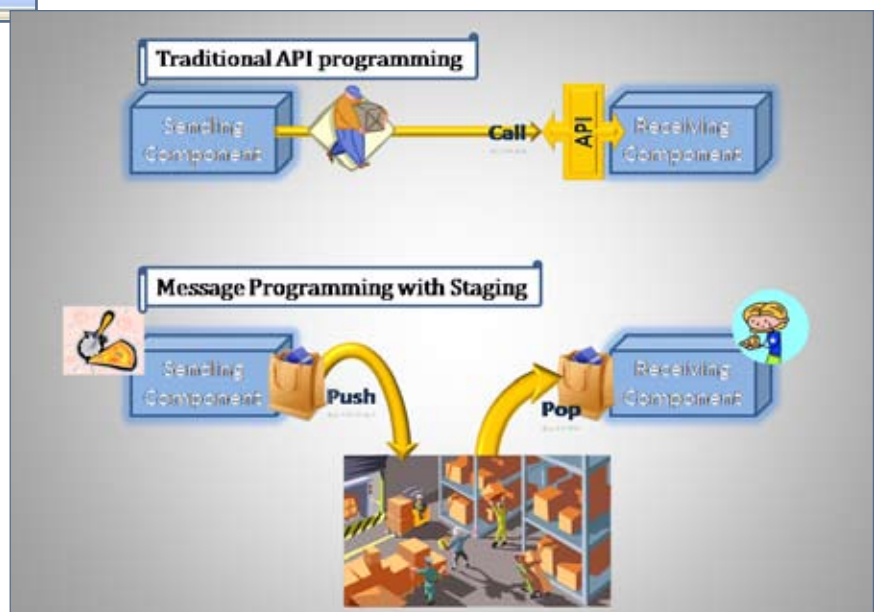


Figure 6: Traditional API Programming versus Message-Based Development

The Magic of Queues

The implementation of a staging area is covered in detail in the documentation for message queues. These queues serve as temporary and reliable data storage. There are many variations of queue implementation. From the name, we immediately think of the very sophisticated, database-driven, professional queuing systems like Microsoft MSMQ, WebSphere MQ (formerly known as MQ-Series), or the Open Source Apache ActiveMQ. However, any file system can be viewed as a queue; picture a scenario in which we interpret a folder as the queue and the files as the messages. Email boxes are also classical queues that receive and store messages, and allow for asynchronous handling. Let us now take a little journey through the SOA garden and demonstrate the real magic of queues by discussing and practically demonstrating a real-life RFID (“Radio-Frequency Identification”) scenario.

Practice: Making RFID Work with Queues

In order to shape your understanding on how queue processing works, we have picked one simple example out of some current SOA projects. We have an RFID reader from Siemens that is already intelligent enough to deliver reader data in a convenient XML data format. The data from the reader is transported to an SAP instance, where it can be used by an application.

Let’s start with the difficulties we’ll encounter.

While the reader can send XML, it does it via a simple, plain TCP/IP Telnet protocol; we need to have an agent that triggers the data stream and listens to the incoming reader data.

Problems with this scenario:

1. The reader fires several times per second, but may not immediately interpret the reader data content.
2. The reader message format may change as the reader model changes.
3. The data should not be volatile (not be lost when the SAP system is down).

4. There should only be one record per tag sent to SAP, even if the tag is read multiple times (in fact, a single tag may fire a thousand times).

We have not found a single person who can solve all these issues right away. A lot of experimenting and studying is necessary. So, we took the issue list and tried to solve every issue in an isolated environment, independent from each other.

For instance, when we receive duplicate data, we need an agent to review the data stream and aggregate it to a single result record. Therefore we defined a message queue as a staging area where we expect the reader handler to deliver the messages. In our case, we decided to simply write the messages to a folder in the file system with the intention to replace it with an MSMQ implementation eventually. Our aggregation routine will now find all the messages in a folder, and can review them to check if there are duplicates.

The basic Telnet protocol has been kept really basic. It reads the data stream as it comes in and puts it in an in-memory queue. A second asynchronous task empties the memory queue and copies the messages, message-by-message, into the staging area. In Figure 7, we show the overall landscape of our scenario, with a lot of intermediate queuing involved.

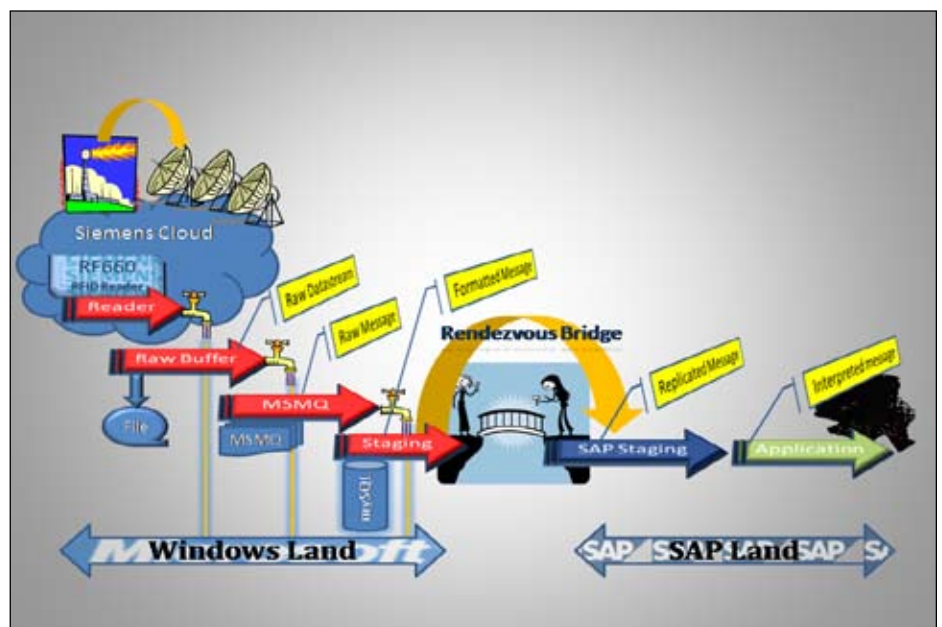


Figure 7: Software Development with Queues

RFID Technical Process Flow

So are you ready to see this RFID scenario in action?

Figures 8 through 14 show the process running, and how the messages are “staged” at each step in the scenario.

Before we start, it is perhaps important to see how communication with the RFID reader is achieved using the traditional protocol TELNET. Figure 8 shows the “host greeting” XML message being sent to the reader and the huge array of messages that stream in after that.

The technical process consists of a number of “Python scripts” that provide the necessary services for the scenario to run from end-to-end. The Python components have been written in a very “event-driven” nature whereby the communicating services react to certain events that tell them, for example, that it is their turn to do something with the incoming RFID tag messages. The “event-driven architecture” (EDA) provides a truly asynchronous/staging sequence of events that is extremely easy to understand and troubleshoot.

Figure 9 shows the start of the technical process, whereby the “RFID receiver service” creates a socket connection to the RFID reader, passes in the “host greeting” XML message (just like the TELNET session above), and then sits back and receives the RFID tag messages.

```

jupiter.logosworld.com - PuTTY
<message><name type="c">hostGreetings</name><paramGroup name="readerGreetings">
messagingVersion GR_XML_2.0 </messagingVersion><appVersion> RF660R Configuratio
n Software V1.1.0 </appVersion></paramGroup></message>
<message><name type="r" status="ack">hostGreetings</name><paramGroup name="reade
rGreetings"><readerName>SIMATIC RF660R Portal Reader</readerName><firmwareVersio
n>V1.1 (01.01.00.00 01.08)</firmwareVersion><fpgaVersion>V0.44.2</fpgaVersion><c
onfigVersion>USER</configVersion></paramGroup></message><message><name type="n">
ter</name><ter>112,2,3,3,300833B2DDD9014035050000,53584</ter></message><message>
<name type="n">ter</name><ter>112,2,3,3,300833B2DDD9014035050000,53584</ter></me
ssage><message><name type="n">ter</name><ter>112,2,3,3,300833B2DDD9014035050000,
53584</ter></message><message><name type="n">ter</name><ter>112,1,3,3,300833B2DD
D9014035050000,53585</ter></message><message><name type="n">ter</name><ter>112,1,
3,3,300833B2DDD9014035050000,53585</ter></message><message><name type="n">ter</
name><ter>112,1,3,3,300833B2DDD9014035050000,53585</ter></message><message><name
type="n">ter</name><ter>112,1,3,3,300833B2DDD9014035050000,53585</ter></message>
<message><name type="n">ter</name><ter>112,1,3,3,300833B2DDD9014035050000,53585
</ter></message><message><name type="n">ter</name><ter>112,1,3,3,300833B2DDD9014
035050000,53585</ter></message><message><name type="n">ter</name><ter>112,2,3,3,
300833B2DDD9014035050000,53586</ter></message><message><name type="n">ter</name>
<ter>112,2,3,3,300833B2DDD9014035050000,53586</ter></message><message><name type
="n">ter</name><ter>112,2,3,3,300833B2DDD9014035050000,53586</ter></message><mes
sage><name type="n">ter</name><ter>112,2,3,3,300833B2DDD9014035050000,53586</ter
></message>
  
```

Figure 8: RFID Tag Messages Streaming Back Using Telnet

```

def handle_read(self):
    data = self.recv(10000)
    buffer = self.prev_right_buffer + data
    buffer = buffer.replace("</message>", "\n</message>")
    msg_list = buffer.split("\n")
    last = msg_list[-1]
    if last.find("<message>") == -1 or last.find("</message>") == -1:
        self.prev_right_buffer = last
        msg_list.pop()
    else:
        self.prev_right_buffer = ""

    memory_queue = self.memory_queue
    for msg in msg_list:
        memory_queue.push(msg)
        self.event.set()

def handle_write(self):
    sent = self.send(self.buffer)
    self.buffer = self.buffer[sent:]
  
```

Figure 9: Real-Time RFID Tag Reads

¹ Python is a dynamic object-oriented programming language that is a true winner when it comes to agile SOA development. (<http://www.python.org/>)

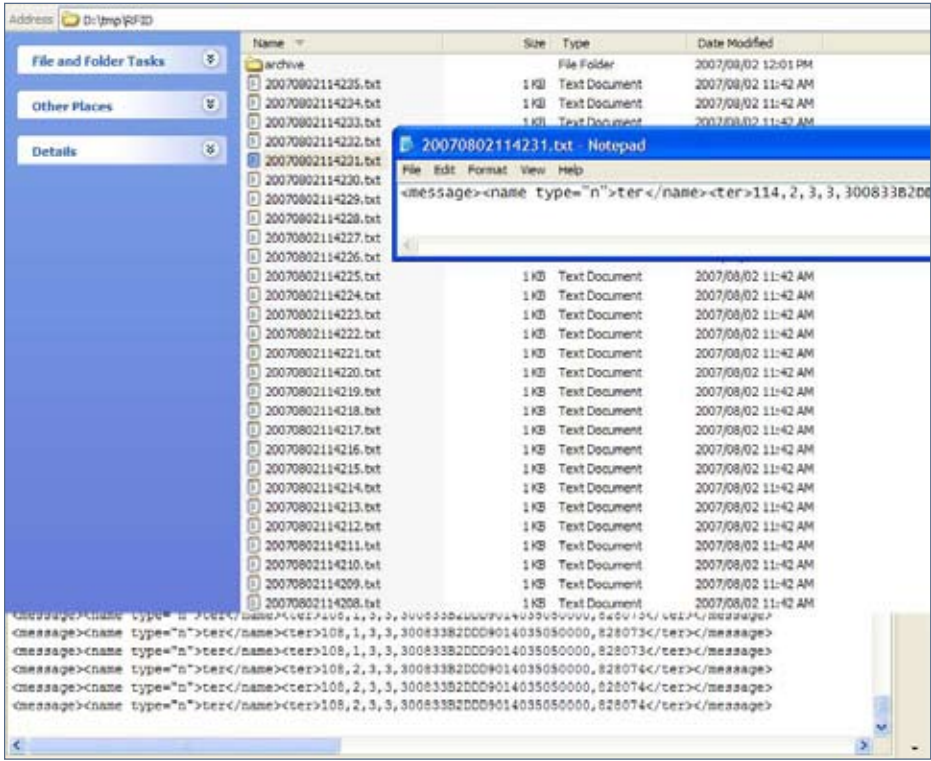


Figure 10: RFID XML Tag Messages Written to the File System (Staging Area)

As soon as the RFID tag messages are received, they are immediately written to a “memory queue”, and once there, they are given a unique “timestamp” and then are written to the file (queue) system. Figure 10 depicts the real-time RFID tag reads being written to the first “staging area” – the file system.

From here, another Python service (that is running) immediately recognizes that new files have arrived, and sends the messages off to Microsoft Message Queue (MSMQ). Figure 11 illustrates the running code and the debug messages, showing that new files arrived and were immediately transported to MSMQ.

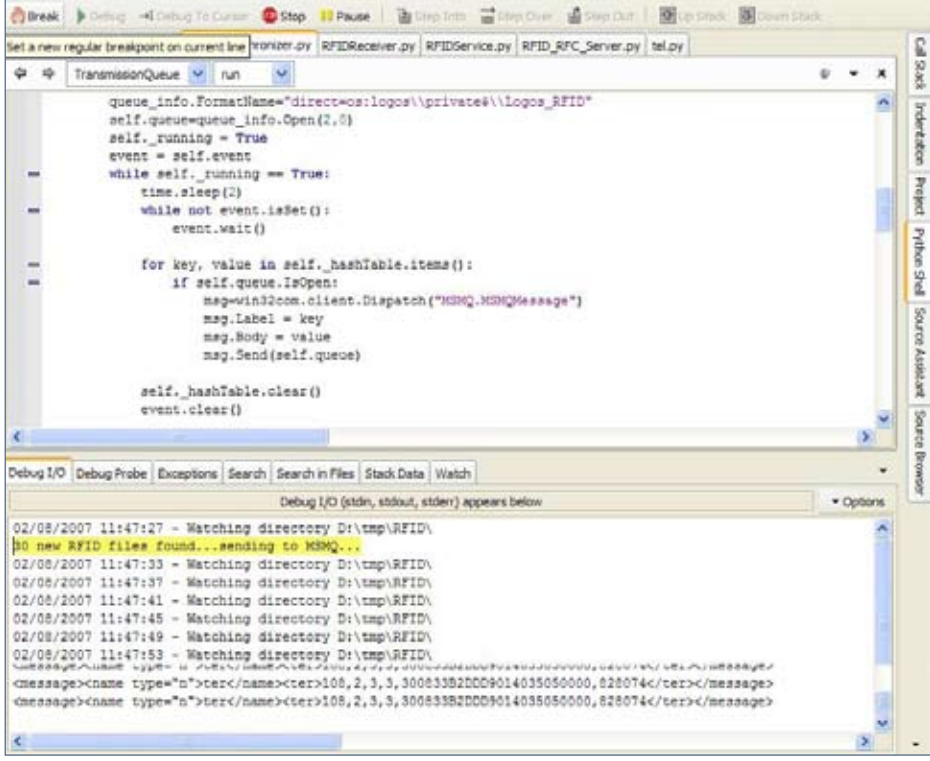


Figure 11: RFID XML Tag Messages Transported to MSMQ

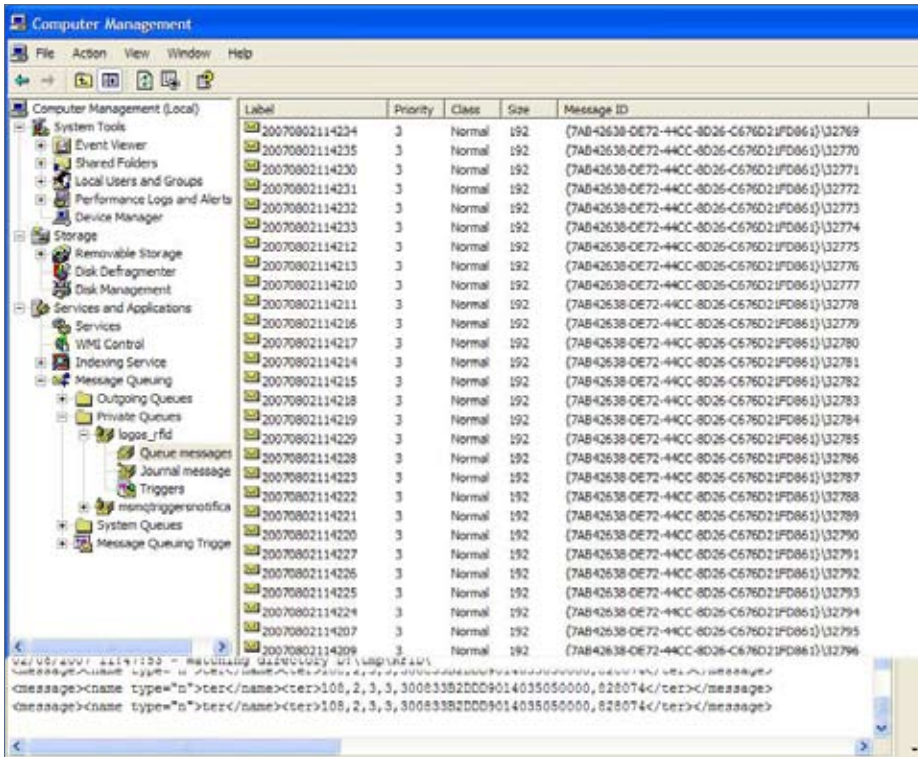


Figure 12: RFID XML Tag Messages on MSMQ

Figure 12 simply shows the RFID tag messages sitting in MSMQ. Queues bring about such a great asynchronous flavor to integration, and MSMQ fits perfectly into this scenario.

Once the messages have arrived on MSMQ, an event is raised to tell another Python service to pick up the messages and move them into an “RFID table” on MYSQL. Figure 13 shows the Python service recognizing that new messages have arrived on MSMQ, and immediately transporting them to MYSQL. Finally, the messages end up in an “RFID” table in MYSQL. This is depicted in Figure 14.

How Much Time Does This Take?

Our task was to connect an arbitrary RFID reader to an SAP system. The challenge is

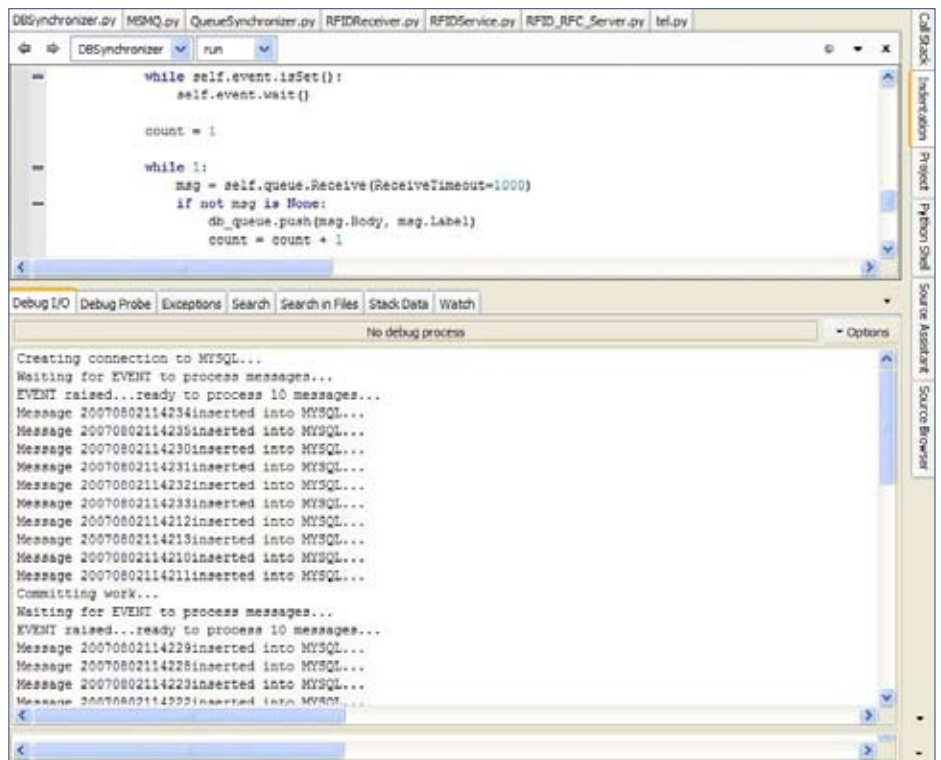


Figure 13: RFID XML Tag Messages Transport from MSMQ to MYSQL

QUEUE	SEQUENCE	DB_TIMESTAMP	TAG_TIMESTAMP	STATUS	MESSAGE
	78	2007-08-02 12:04:0	2007-08-02 11:42:3	100	<BLOB>
	79	2007-08-02 12:04:0	2007-08-02 11:42:3	100	<BLOB>
	80	2007-08-02 12:04:0	2007-08-02 11:42:3	100	<BLOB>
	81	2007-08-02 12:04:0	2007-08-02 11:42:3	100	<BLOB>
	82	2007-08-02 12:04:0	2007-08-02 11:42:3	100	<BLOB>
	83	2007-08-02 12:04:0	2007-08-02 11:42:3	100	<BLOB>
	84	2007-08-02 12:04:0	2007-08-02 11:42:1	100	<BLOB>
	85	2007-08-02 12:04:0	2007-08-02 11:42:1	100	<BLOB>
	86	2007-08-02 12:04:0	2007-08-02 11:42:1	100	<BLOB>
	87	2007-08-02 12:04:0	2007-08-02 11:42:1	100	<BLOB>
	88	2007-08-02 12:04:1	2007-08-02 11:42:2	100	<BLOB>
	89	2007-08-02 12:04:1	2007-08-02 11:42:2	100	<BLOB>
	90	2007-08-02 12:04:1	2007-08-02 11:42:2	100	<BLOB>
	91	2007-08-02 12:04:1	2007-08-02 11:42:2	100	<BLOB>
	92	2007-08-02 12:04:1	2007-08-02 11:42:1	100	<BLOB>
	93	2007-08-02 12:04:1	2007-08-02 11:42:1	100	<BLOB>

Figure 14: RFID XML Tag Messages in MYSQL Table

that the reader type may change easily and that the SAP system is not available at all times. In addition, we had no resources familiar with all of the involved technologies. As a practical matter, this resulted in a sequence of components that could be developed by different people. In order to guarantee the data communication between the components, we agreed on simple data storage like the file system or an ODBC database. We chose these as we knew that any data store can essentially be seen as a message queue, so that we could assume that at least one of the IO mechanisms could be read (and written) by the developers.

Writing the actual components typically took less than a couple of hours. Testing could be accomplished in one day, on average. In our case, it was two people testing for half a day. Putting the components together depended on the number of links; we had four of them. Altogether, the whole scenario was developed in approximately 10 man-days. We have to add an overhead of another 20 man-days of training and research, to get acquainted with the somewhat new technologies like Python and MSMQ, but this will not occur when we build the next interfaces.

Less Is More: Saving Money and Time and Win Quality

In this installment, we demonstrated that it's possible to save money and time when developing for an SOA.

The savings are so impressive that we can even allow for “playing”, research, and training within the same budget limits of a traditional development project. But that is not all; the holistic way of development in SOA also adds to the quality of the product (system), and makes solutions possible that could not have been achieved otherwise.

Axel Angeli, logosworld.com. Axel is a senior SAP and EAI advisor and principal of logosworld.com, a German-based enterprise specializing in coaching SAP and EAI project teams and advising IT management on implementation issues. Axel has been in the IT business since 1984, and throughout his career, he has always worked with cutting edge technologies. Axel's SAP experience stems from the good old R/2 days, and he is an expert on SAP's NetWeaver™ technology and any kind of ABAP development. A speaker of several languages, Axel specializes in coaching and leading large multinational teams on complex projects with heterogeneous platforms and international rollouts. Known for his intensive and successful trouble-shooting experience, Axel has been nicknamed by his colleagues as the “Red Adair” of SAP projects. He is the author of the best-selling tutorial “The SAP R/3 Guide to EDI, IDocs, ALE and Interfaces.” You may contact the author at SAPtips.Authors@ERPtips.com. Be sure to mention the author's name and/or the article title.

Lynton Grice, logosworld.com, is an SAP integration specialist contracting in Johannesburg, South Africa. He has a passion for programming and has a particular interest in the “business process and SOA space” incorporating things like workflow, BPEL, SAP X,I etc. Lynton is a champion of the Blue Elephant League of SOA Excellence. You may contact the author at SAPtips.Authors@ERPtips.com. Be sure to mention the author's name and/or the article title.

SAPtips *Journal*

The information in our publications and on our Website is the copyrighted work of Klee Associates, Inc. and is owned by Klee Associates, Inc. NO WARRANTY: This documentation is delivered as is, and Klee Associates, Inc. makes no warranty as to its accuracy or use. Any use of this documentation is at the risk of the user. Although we make every good faith effort to ensure accuracy, this document may include technical or other inaccuracies or typographical errors. Klee Associates, Inc. reserves the right to make changes without prior notice. NO AFFILIATION: Klee Associates, Inc. and this publication are not affiliated with or endorsed by SAP AG. SAP AG software referenced on this site is furnished under license agreements between SAP AG and its customers and can be used only within the terms of such agreements. SAP AG and mySAP are registered trademarks of SAP AG. All other product names used herein are trademarks or registered trademarks of their respective owners.

