Business Services Orchestration

The Hypertier of Information Technology

Waqar Sadiq
Electronic Data Systems

Felix Racca
Fuego Inc.

Cambridge University Press

PUBLISHED BY THE PRESS SYNDICATE OF THE UNIVERSITY OF CAMBRIDGE
The Pitt Building, Trumpington Street, Cambridge, United Kingdom

CAMBRIDGE UNIVERSITY PRESS
The Edinburgh Building, Cambridge CB2 2RU, UK
40 West 20th Street, New York, NY 10011-4211, USA
477 Williamstown Road, Port Melbourne, VIC 3207, Australia
Ruiz de Alarcón 13, 28014 Madrid, Spain
Dock House, The Waterfront, Cape Town 8001, South Africa

http://www.cambridge.org

First published 2003

Printed in the United States of America

Typefaces Berkeley Oldstyle 10.75/13 pt and Franklin Gothic *System* LaTeX 2$_\varepsilon$ [TB]

A catalog record for this book is available from the British Library.

Library of Congress Cataloging in Publication Data

Sadiq, Waqar.

Business services orchestration : the hypertier of information technology / Waqar Sadiq, Felix Racca.

p. cm.

Includes bibliographical references and index.

ISBN 0-521-81981-4

1. Business – Computer network resources. 2. Business information services. I. Racca, Felix, 1955- II. Title.

HF54.56 .S23 2002
658.4′038 – dc21 2002074198

ISBN 0 521 81981 4 hardback

Contents

Chapter 1

A Holistic View of Enterprise Systems

1.1 Introduction

The first part of this introduction is for the Kirks, Spocks, McCoys, and Scottys. As it gets more technical, the Kirks may want to skip directly to Section 1.3, from which point the content is more business oriented.

To Orchestrate is to organize the harmony and tempo of various instruments into a single impressive musical delivery, such as in Strauss's *Blue Danube* waltz. If the result is anything but impressive, the orchestration is not worthy of such a name. An Orchestrator differs from an Architect in that the latter designs something static, such as a house, a bridge, or a landscape, that is, something that doesn't vary in time. The Architect designs only the deliverable, not the process used to deliver it. An Orchestrator designs a delivery of music rendered in time, in the harmony and tempo needed to achieve a desired effect.

Orchestration is nothing but a modern metaphor that describes a well-known, but not very well understood, discipline: the automation of business process management. Traditionally, business processes were managed by people. Managers drove the steps that employees – with or without instruments or tools – needed to complete or fulfill a business process. This is analogous to an orchestration being managed by a maestro by keeping tempo, cueing in the different players, managing the intensity of what is being played, and conveying a style to the performance.

The same way MIDI tools manage the automation of musical execution, Business Services Orchestration (BSO) tools manage the automatic execution of business services to fulfill an enterprisewide or interenterprise business process.

The problem with this concept is that, since the advent of the computer era, there have been many waves of automation, all of which could be confused with BSO. The key to understanding the difference between BSO and other types of automation is precisely that BSO automates those cumbersome, iterative, and mechanical things that management was forced to do to run a business, not the highly specialized functions of any specific discipline such as accounting, engineering, law, sales, or procurement. Examples of these generic tasks include prompting people to do their specialized work; prompting systems to do their specialized work; measuring what people and systems do; interpreting, translating, and routing information; time management; resource allocation; and many others. These activities don't require creativity, specialization, or great amounts of intelligence – just a huge amount of attention to detail, patience, and a preestablished plan that rules the execution.

At this point, the technical reader will say: "That's nothing new, that's work flow!" But it isn't. Or at least it isn't work flow as it has been understood by industry analysts and by software providers to date. There has been close to unanimity of opinion in the technical community that work flow (and its more modern successor, business process management [BPM]) are a feature set of enterprise applications (departmental work flow) or a feature set of integration suites (BPM), or in the most generic case, a document management facility. Most industry analysts believe that there is no chance for an overarching orchestration strategy – one that automates the management of processes across people, applications, departments, divisions, and enterprises – to succeed as a product category.

It is the belief of the authors that this type of solution that is *specialized* in being generic, automating generic recurring tasks and automating the management of services from inside and outside an organization, is precisely what is needed to fix the problems of the modern enterprise. What are these problems?

- Geographic dispersion makes it difficult to coordinate work.
- Language, time zones, and cultures need to be managed.
- Platforms, computer languages, runtime environments, and disparate, specialized applications need to be made to work the way the business requires them to work.
- Time that could be put to better use is spent by people interpreting data from one application and putting it into another.
- Activities that are recurrent and mechanical are still being performed by people.

The list could go on, but let's get to the most important problem of them all: *Although most companies have plans, these plans are at a high level and do not constitute* ***actionable plans*** *that are understood and agreed upon by all of the management*

team as well as those who will execute or supervise the work required by them. Therefore, companies have trouble understanding and communicating, at a detailed level, which services from specialized people and systems they really need in order to succeed in their objectives.

These actionable plans can be called company execution contracts. They are understood by everybody involved and they rule that involvement. What generic language is there to express these contracts or actionable plans?

The best language invented to date is the graphical representation of process models that explain what people, systems, and other organizations need to do for the business to fulfill its objectives. Process models are to business people and workers as librettos are to the maestro and the musicians.

Now, let's suppose that we can feed those process models ***exactly as they have been designed and understood*** to a software engine and this engine prompts people to do their job, presents them with interfaces that are adequate, receives communication from people as they complete tasks, invokes functions in underlying applications, uses the return arguments as data for function invocations in other applications or to present new data to people, manages due dates, presets times to launch work from people or systems, and creates information on everything that was done, by who, how long it took, what tools were used, etc.

What we have just described is a business services orchestration engine (BSOE). The key here to distinguish BSO from prior technologies is that the process models that are executed by the BSOE need to be exactly what the management team agreed upon. The process models deal with generic management and supervisory issues. Activities undertaken by specialized people, systems, or organizations are business services that the BSOE invokes, coordinates, manages, and measures according to the process models it has been fed.

Therefore, we can say that we are in the presence of a BSO project only when:

- Management agrees among themselves and with appropriate workers on actionable plans across departmental silos and partner companies.
- These plans are represented as process models.
- These same process models can be fed into a BSOE that automates their management and execution.
- These same process models constitute the contract between senior management, supervisors, workers, and information technology (IT) staff on the scope of what services people and systems need to provide the business.

We are in the presence of BSO tools only when:

- There is a process designer that allows the graphical creation of orchestrations by management-level users, and a capability to automatically generate connectors to applications and presentations for human participants by IT-level users.

- The orchestrations are the implementation framework for everything that IT has to do thereon.
- The exact orchestrations that were designed are what runs in the BSOE.
- The orchestrations can be changed without modifying IT infrastructure or having to retrain users.
- The BSOE creates process execution information that is easily exploited by management to continuously improve the processes.

As can be seen, this is clearly a top-down approach toward automating cross-enterprise processes. Traditionally, we have taken a bottom-up approach: we have created specialized automations that do sophisticated calculations and organizations of data. We have attacked the problem correctly; we have solved the greater pains first. However, in today's world the greater pain has become that these specialized systems don't work together, nor are they coordinated with work from employees and business partners.

We dedicate the rest of the introduction to a more technical overview of what has been done to date to support businesses through IT.

Let's investigate what companies have done in the past to orchestrate their business functions using IT. Initially, companies developed internal software to facilitate the work of specific working groups. IBM's business machines provided specific language compilers for different types of applications. Programmers specialized in languages. For example, there were COBOL programmers, dedicated initially to creating administrative applications, and FORTRAN programmers dedicated initially to creating engineering applications. As time went by, programmers started creating applications for Human Resources, Manufacturing, Logistics, Cost Analysis, Procurement, Delivery, and many other functional areas of the enterprise.

Initially, these systems were built in-house. IBM and others were marketing a "General Purpose Machine" capable of running virtually any program, and they trained their customers to build internally developed programs that accelerated business functions.

As programmers left those companies, the programs they created became unmanageable black boxes that were difficult to use and costly to update. In response, a new generation of software vendors introduced department-specific applications that were supported by cost-effective maintenance contracts. Those initial software vendors were domain experts in one or more functional silos, and they specialized in systems that helped the workers in those specific silos to improve their work.

Companies soon realized that they needed to share work with other departments, and that this sharing required that transactions or reports be sent between various departmental systems. Some smart software vendors started to sell "integrated modules" that would later be known as enterprise resource planning (ERP)

suites. ERP was described as the end-all, cover-all, integrated system that would allow companies to conduct business with a minimum of paperwork. The rationale was that by running everything under one system and one database, the company would have access to all the functionality and information it could ever need.

That sounded great, but new times bring new challenges and, as companies realized a growing need to manage their supply chains and customer relationships, supply chain management (SCM) and customer relationship management (CRM) software entered the marketplace. Those systems ran on their own database structures, which had nothing to do with either the ERP or with the customer or supplier databases. The dream of a common, centralized database had been seriously compromised.

Not having been designed as a part of ERP, these new SCM and CRM applications, by nature, have data that are redundant between them and with ERP, and have overlapping processes (i.e., purchase order data and the order fulfillment process). When companies do business in different markets, they usually have at least one or two systems that are not included in any of the "integrated suites." For example, telecommunications firms use billing and OSS systems, insurance companies use policy and claims management, and financial services need case management and branch office automation software.

So, the modern enterprise has deployed ERP, CRM, SCM, and two or more industry-specific applications, their Intranet, Extranet, Internet servers, and content, plus e-mail, plus personal productivity and collaboration tools. This is the scenario in today's market. What do all of these applications have in common? They were built to provide functionality to the intended user according to the *user's* perception of what is needed to do his/her job. People within the companies that use them are in charge of taking these fragmented processes and data representations and producing a real (as opposed to virtual) business process that satisfies their customers.

It's no wonder that, having all of these islands of integrated software, companies are struggling to integrate their business.

The simple truth is that some employees spend a tremendous amount of time swivel chairing from one application to another trying to maintain their data in synchronization. It's easy to see that this is a very cumbersome and error-prone task. Therefore, at first blush, it would seem that companies want to integrate applications to eliminate the swivel-chair operators. However, that improvement alone will not convince management to buy a multimillion-dollar license for integration software, and spend five to seven times that in professional services. No. There must be another reason.

In our experience, the reason companies integrate applications is to improve the performance of their critical processes so that they can better serve their customers and/or be more efficient.

They wish to do this in a way that will produce measurable returns on their investment, if possible within the budgetary year. The challenge is made greater because various parts of these critical processes are embedded and redundant in their CRM, SCM, and ERP applications, industry-specific suites, Web applications, personal productivity and collaboration tools, and the capacities of their employees. Also, in a virtual enterprise, many of these critical processes are done by employees or partners outside the four walls of the company. Still, companies have to manage the overlaps, redundancies, inconsistencies, and white spaces among and between those many applications, people, and organizations.

In synthesis, the problems that businesses need to solve are:

- to tightly connect business execution with business strategy and objectives;
- to ensure the constant reliability of execution by orchestrating the behavior of people, systems, and business partners;
- to do this without disrupting the business's culture or preferences in terms of organization and infrastructure; and
- to avoid impairing the business's ability to change at the rhythm of market requirements, competitor capabilities, and internal or external innovation.

Without any doubt, it's a tall order.

The industry's initial approach to knitting these processes together (orchestrating their business services from applications) was to create a program that took data from one application and automatically put it into another. This approach got old very soon, primarily because of the enormous number of interface programs needed to make it work.

As an example, let's say that a company has the following applications: General Ledger, Accounts Receivable, Accounts Payable, Human Resources and Payroll, Manufacturing, Procurement, Inventory Management, and Billing. The initial reasoning was, "Let's make an interface program between each of the modules in each direction." So point-to-point interfaces were developed between General Ledger and the remaining seven modules, and then between the Accounts Payable and the remaining modules, and so on. When we were one-tenth of the way through this approach, we realized that we would have to build $8 \times 7 = 56$ interface modules. However, the problem didn't end there! These applications started having different versions, and so, each new version of each module implied fourteen new interfaces (seven incoming and seven outgoing).

It got ugly in a hurry. When industry analysts started calling it the "spaghetti interfaces" approach, we knew we needed a better way.

That new approach emerged in the late nineties when the pioneers of today's enterprise application integration (EAI) suites found a way to avoid creating an almost geometric number of interface programs. There are two common EAI

approaches, Hub-and-Spoke and Messaging Bus, but they are based on the same basic concept that applications are connected to a single broker instead of among themselves. In this way, instead of having fifty-six interface programs to maintain as described in the example above, we would only need to create eight adapters or connectors to the broker; one for each application. Under this approach when a new version appeared, only a single new connector would be necessary.

The rationale was impeccable, but something went wrong. In early 2000, we started hearing the same analysts questioning this new approach that they had contributed to popularizing. "We went from spaghetti interfaces to spaghetti EAI," many observed.

Shortly thereafter, the big Internet economy bubble, already wobbling, burst completely. The "dot com" revolution had, in many cases, been unable to either create a viable revenue model or implement the adequate orchestration of their internal services to fulfill the model.

These problems were already obvious when we started to complicate things further by trying to create transparent marketplaces through business-to-business (B2B) exchanges that stressed the hub-and-spoke paradigm to the utmost. In B2C the spokes were implemented as Web sites and the only orchestration necessary was between the Web application and the back-end applications and people. In B2B, each spoke needed its own orchestration.

Besides, we soon discovered that companies wanted to continue to do business their own way and were less than eager to relinquish their existing models for a more perfect marketplace. The reason for failure becomes obvious when we consider the cost of integrating companies into exchanges, an average of about half a million dollars and requiring four months of effort. Although the business reasons for the failure of B2B exchanges are clear, we can't help thinking that the technological approach and its limitations played an important role in accelerating their demise.

We believe that the e-business revolution has just started. BSO will be a major advancement in making this revolution viable. It will provide an approach and tools that, although building on previous ideas and technologies, will greatly diminish the risk of e-business by improving the time-to-market of solutions and providing the flexibility needed for their continuous improvement. The causes of our present predicament are many and varied but, on the technical side, the rigidity of integration solutions is probably one of the most important. For some reason, early on, work flow and integration were divorced. BSO sustains that they are one and the same thing, and that services from people and services from applications need to be regarded under the same light as process activities.

One of the main causes of inflexibility is that the messaging approach did not completely replace the point-to-point integration programs as we thought it would. Although the point-to-point integration programs were many, they were programs,

not connectors or adapters. What's the difference? The fundamental objectives of a connector or adapter are:

1. Take standard data from the broker, transform it so that the target application can understand it, and then call a procedure in the target application using the transformed data.
2. Take the output parameters from an invoked procedure, transform them into broker standard constructs, and then hand the result to the broker.

In contrast, the objective of a point-to-point interface program is much broader: To do anything and everything necessary to make application A work seamlessly with application B for a given transaction or set of transactions. The interface program might apply rules, manage exceptions, and drive the process of knitting together the two applications. Adapters or connectors simply cannot do this. The problem with an interface program is that it is tightly coupled and point to point. This makes the program difficult to change and requires a quadratic number of interfaces to be built as the number of applications grows, doesn't it?

Not necessarily. The quadratic multiplication of interface programs depends on the layer of abstraction in which the program is built. Initially, they were built as peer programs to the two applications that were being integrated. The issue was in the point-to-point nature of the interface programs, not in the fact that they lacked a central means of connectivity. The inflexibility of those programs had more to do with the fact that they were developed in programming languages rather than generated from process models that could be graphically constructed and changed on-the-fly as needed, and that they were designed as interfaces, not as overarching processes.

We should recall that the EAI approach was initially limited to passing data from one application to another, assuming that the content of those data and the queue in which they were published would suffice for the adapter to invoke all adequate procedures in the target application. For this to be true, two conditions would have to be met:

1. An adapter or connector to an application would have to connect to all methods of an application programmer interface (API), and that API would need to be exhaustively complete for whatever the external world might want from the application.
2. There could be no mismatch between what one application provided and what all others required from it.

When the difficulty of meeting these conditions became apparent, EAI vendors started adding a logic layer inside the brokers. They called it process logic. It should be called event-handling logic, and for a very simple reason: Most EAI vendors

built this supplementary logic tier as a set of rules that reacted to the appearance of certain events on the bus. Therefore, it was logic driven by events, which is the opposite of activities driven by logic (which we consider to be the definition of process).

Although it is also true that a decade before the EAI vendors came up with the idea of integration brokers, work-flow vendors had pioneered the approach of integrating applications inside process definitions, the ability to interact with underlying applications was very poor at that time. The first work-flow products concentrated mainly on routing documents among people, applying process rules to manage the behavior of the document flow, and people activities. BSO sees applications as if they were service providers analogous to people. In today's world, there are already technologies and technology bridges that allow the process to manage the behavior of underlying applications as well as the behavior of people.

As customers realized that the passive-state engine logic as provided by EAI vendors was not enough, systems integrators ended up building auxiliary interface programs that connected to the bus and supplemented the lack of active logic, or they started putting this logic within the connectors or adapters. These addendums seriously compromised the maintainability and flexibility of the resulting constructs.

If the main reason for integrating those applications is to improve the performance of a company's processes, we should ask ourselves: Does it make sense to focus on connecting applications? Or does it make more sense to focus on automating a company's processes as the work-flow vendors did originally? What did we really achieve by trying to make these applications send electronic messages to one another? Haven't the results of this approach been highly redundant, overlapping, inconsistent, and clunky processes that are usually worse than the one driven by swivel-chair users or interface applications, and haven't we ended up building auxiliary interface programs anyway? Haven't we ended up distributing the logic and centralizing the technology, creating a maintenance nightmare, a single point of failure, and a scalability problem?

This critique implies that a partial approach (just throwing messages over application walls into a bus) is insufficient to drive and improve company performance. Even if it were possible to eventually make the EAI approach work by supplementing the magical event-driven scenario with the interface process logic that we were trying to eliminate, the effort necessary for event-enabling applications to be able to integrate them takes too long, costs too much, and is exponentially more difficult to maintain as the number of integrated applications grow. We have traded exponential point-to-point interfaces for an exponential maintenance nightmare.

What the industry needs today is a holistic and self-propelled approach.

The holistic approach starts by focusing on all of a company's generic customer-facing and interdepartmental processes instead of only the company's specialized user-facing applications. These customer-facing and interdepartmental processes

are, by nature, cross-function, cross-division and cross-ecosystem. We call the approach holistic precisely because it starts by analyzing the complete end-to-end services to be delivered, rather than looking at the individual functions from specific applications. The holistic approach recognizes applications, people, and organizations as containers of a wealth of services that can be orchestrated into new, internal or external customer-facing services. This is done through the execution of a process model that automates the iterative and routine tasks that people have to perform to make sense out of disparate processes and data representations in their fragmented application base. It is an approach in which the orchestration engine acts as a hub that runs flexible process models. These process models invoke fragmented services from people and systems, through any technology, and apply business rules to them, transforming them into new and improved business services. This approach does not aggregate applications through one proprietary means of connectivity. Rather, it orchestrates services through process logic applied to existing services and exposed through any means of connectivity.

BSO is this holistic approach. BSO is not only a product or a product category. It is also an approach toward business integration that requires certain types of tools and technologies that respond to a specific architecture. We discuss this architecture in Chapter 3, and describe the approach by using a methodology framework in Chapter 4.

At its highest level, the BSO approach consists of seven steps for continuous improvement. Those are:

1. Identify and prioritize the critical services needed for company success.
2. Discover how the company fulfills these services (model the "as-is" processes).
3. Define the improvement objectives in terms of process metrics (cycle time, quality, volume, cost, etc.).
4. Modify the as-is process to try to meet those objectives by modeling the should-be process and determining the services it needs to invoke from the business ecosystem.
5. Identify any existing and useful services that can be provided by people, applications, or third parties.
6. Harmonize these services so they can be consumed by the should-be process.
7. Implement the process and put it into production. Monitor metrics. Return to step 3 and repeat as necessary.

For those who appreciate business process reengineering (BPR), this approach no doubt sounds very familiar. BSO derives a good part of its high-level methodology and approach from BPR, but the objectives and end results are different.

The primary objective of BPR is to help a company discover and streamline (reengineer) its internal processes. The results of a good BPR project are a number

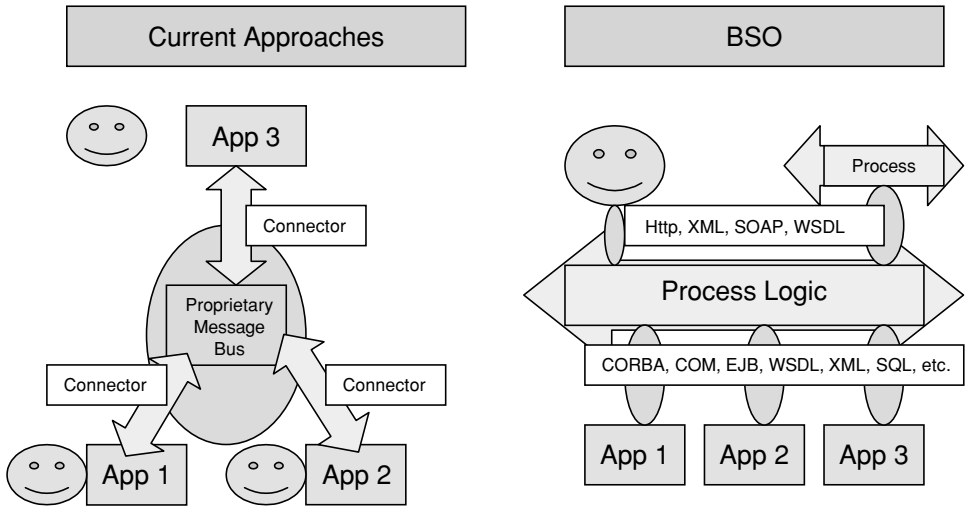

FIGURE 1.1. Current Messaging-Centric Approaches Versus the BSO Approach.

of documents that specify a complete set of internal processes and policies. It's the company's responsibility to train its employees in those procedures, and the implementation of the process logic in the company's systems is done, if at all, through the trained employees. There is no way to enforce BPR processes on third-party organizations.

The objective of BSO is to use existing internal or external services (automated or not) to create a new automated service for any internal or external customer. The result of an orchestration project is a computer-driven process that uses services from people, computer applications, and other organizations to deliver a product or service. BSOs are process models and rules that, rather than being subject to human interpretation and execution, are executed by a computer. This execution is consistent, metrics driven, and easily adaptable to changing business conditions.

Orchestration is a departure from BPR because its goal is to create a computer-driven service. BSO needs the abilities found in traditional work-flow systems intermingled with the capabilities of traditional EAI and business-to-business integration (B2Bi) systems, but the BSO approach differs significantly from the traditional EAI approach because it focuses on creating processes that drive the execution of services, instead of creating adapters that listen to and create events on a centralized messaging bus.

Figure 1.1 compares current messaging-centric approaches with the BSO approach. We notice the following from this comparison:

- BSO focuses on improving company performance by orchestrating the corresponding services, more like work-flow solutions would approach the problem.

Data sources, employees, applications, and external systems alike are seen as containers of business services to be used as needed by the orchestration intelligence.

There is no distinction among EAI, B2Bi, and B2C. Under the BSO approach, companies build and use services – internal or external, human or automated, standard or nonstandard and from any technology – to improve enterprise performance.

For orchestration to be possible, BSO technologies must provide a capability that we will call *harmonization,* which we address at length later in this book. In essence, harmonization provides the ability of an orchestration to work with services from any type or origin.

Like work-flows, orchestrations are self-propelled; it is the orchestration engine that invokes business services from people, applications, and organizations and not the other way around; this allows processes to run independently of the implementation of the underlying services.

- BSO uses the process logic as the hub that drives the integration of people, applications, and organizations, therefore allowing this process logic and these services to be shared. This is in contrast to the EAI approach, which only shares data among application systems. Also, in contrast to the point-to-point interfaces that are peers with two applications instead of a layer of logic on top of many, it is in perfect agreement with the work-flow approach.

BSO is, in many ways, the opposite of current approaches, and herein lies the difficulty that people have in understanding it. It's a logical view, not a physical one. In this chapter, we want to make sure that we leave you an image that you can refer to as we get into more detail and the trees start hiding the forest.

Imagine a hub and its spokes. In the current approaches, the hub is in charge of connectivity through one technology and the spokes are in charge of process (applications), which in turn interact with people. People do not interact directly with the hub. In BSO the hub is in charge of executing process models and the spokes are in charge of connectivity *through any technology* to services from people, systems, or other process hubs. The orchestration engine in Figure 1.1 is clearly a generic connector *across many technologies.*

The fundamental difference between BSO and the traditional EAI approaches is that the traditional EAI approaches are technology centric. This is sort of a loaded claim because in IT, everything could be perceived as a technology. EAI, however, relies on proprietary technologies to build adapters and connectors to applications. This requires an IT department to make key strategic decisions on an integration platform. This is so because the adapters and connectors to the applications being integrated are proprietary. The BSO approach that we have talked about so far,

and will continue to dive into, is IT-strategy independent because it relies on the process of harmonization across one or multiple strategies. As long as the services that are being integrated have been harmonized by creating metadata about them and populating standards' registries, they can be easily orchestrated. One might argue that you need technology to express metadata and discover services. That is true, but with standards-based technologies, such as Web Services Definition Language (WSDL) and UDDI, that can be used to describe metadata and discover it; and with synthesis tools that can discover metadata in already-existing component repositories such as CORBA, COM, Java, EJB containers, and others, orchestration becomes a logical configuration rather than a physical one.

Figure 1.1 should reinforce the concept of BSO as a holistic, logical approach toward orchestrating services from any origin through any enabling technology, which is, among other important things, the clear remedy to "spaghetti EAI."

1.2 Business Services

Let's define what we mean by business services. Activities such as catering, transportation, and legal and financial work are classic examples of business services. However, for the purposes of discussing orchestrations, our definition of business services is broader.

At a high level, a business service is anything a company does to fulfill a request from an internal or external customer. This may include actions a company employee takes on behalf of an internal or external customer, a task a company system performs for an internal or external customer, or virtually any combination of the above. One characteristic of any business service is that the customer has the power to request it or not. Unsolicited activities are not considered to be business services. Those unasked-for actions are more properly called business waste or business noise.

To be usable by an orchestration engine, a business service must expose an interface that makes it accessible for invocation by a computer program. So, a more complete definition would be:

A business service is what a company does to fulfill to a programmatic request from an internal or external customer.

You might ask, "Are Web services business services?" The answer is yes because Web services provide a simple programmatic interface implemented on standard Web protocols and are addressable by a URL. Web services are the ultimate way to externalize business services on the internet. However, not all business services are Web services. There are business services that need not be exposed as Web services, such as database queries, the invocation of an API from an ERP system, or human services driven by the invocation of a graphical user interface. In this scenario it is

illogical to try to make any technology, including Web services, the centerpiece of an orchestration. The centerpiece of any orchestration should be the orchestration itself. The services are the instruments that are being orchestrated into a whole. Therefore, as we said earlier, in BSO the hubs of multiple EAI approaches become the spokes, and the spokes with their integration logic become the hub.

For a more complete understanding of the new paradigm, and to provide vocabulary that we will be using throughout the book, let us classify business services according to the following categories: complexity, source, and enabling technology.

1.2.1 Classification by Complexity

What a customer sees of a business service, as we explained previously, is the service's programmatic interface. So, from the customer perspective, the complexity of a service is nothing more than the complexity of its interface. There are three basic levels of complexity in interfaces: discrete, composite, and orchestrated.

1.2.1.1 Discrete Services

These are services that present a discrete interface. This means that a single interaction with the interface completes the service, and this interaction is, to all intents and purposes, executed as a whole without parts. This is the case for services that consist of posting a message, listening for an event, or for some API services that return arguments. Examples include services that allow posting of an e-mail, returning of the temperature in a turbine, posting a message to a Pub/Sub bus, presenting a graphical interface to a user, geting a purchase order from a back-end system, etc. Discrete services can be synchronous or asynchronous.

1.2.1.2 Composite Services

These are services that require more than one interaction to be completed, and therefore consist of multiple parts. For example, some APIs that allow the customer program to query data in a back-end system return a reference to a record set. To get useful data, the program that invoked the initial service must also invoke a new service such as "get next record" from the record set. When programmatically querying a relational database, there are services such as open cursor and get next cursor that allow the calling program to read a set of selected records. In the same way, there may be human services that require a series of interactions to be completed, for example, a negotiation, where the customer solicits an offer and man makes counteroffers until an agreement is reached. From a technical perspective, composite services can usually be decomposed into their parts and their parts considered as discrete services. However, from a BSO perspective, it is important to try to abstract them into a discrete service that offers a single interaction.

From a business perspective, services must have a granularity that makes business sense, not technical sense. This is why any orchestration must accept, natively,

composite services from people or applications as well as other appropriate discrete services, and compose them into a new discrete business-level service or, as we will see later, an orchestrated service. If this activity is done simply from an aggregation perspective without any orchestration, we can call it composition. These compositions also present either a synchronous or asynchronous interface.

Composition is done by creating intelligent business objects (IBOs) that abstract the complexities of one or more composite or discrete business services into an object that offers discrete services. We call them intelligent, not because they are inference engines, but because they "know" how to integrate with the native services from applications or people, making the implementation transparent to the process that uses them. Therefore, the process relies on IBOs to do the right things from the implementation perspective. From the academic perspective, it might be more appropriate to refer to them as "abstract business objects." We will talk more about composition in Chapter 3 and throughout the rest of the book.

1.2.1.3 Orchestrated Services

Certain services present the customer with a complex interface that requires a series of predefined, timed interactions controlled by a business-level protocol. We call these higher-level services *orchestrated services*. As an example, consider an order fulfillment service requiring the following interactions: Send the PO, wait for proposed delivery dates, accept or reject delivery dates, wait for bill of materials, notify when delivery is complete, wait for invoice. This is what we call multisynchronous interaction, meaning that it is composed of a sequence of synchronous or asynchronous interactions spread out in time, and always following a predefined protocol. The service that implements the above interface is by necessity an orchestrated service. Orchestrated services are essential when attempting to synchronize two business processes. There is no easier and more secure way of implementing orchestrated services than BSO.

1.2.2 Classification by Source

A fundamental step toward understanding the BSO paradigm is the ability to see people, systems, and organizations as sources of services or as "service providers." Therefore, it may be helpful to classify services according to these three primary sources.

1.2.2.1 Services from People

This category consists of actions taken by individual people for internal or external customers. These services can be invoked by computer programs, and can range from tightening a screw on an assembly line to handcrafting a piece of art or approving a multimillion-dollar contract. There are certain attributes that distinguish business services from other activities that people do.