Introduction

To start, we would like to set the stage. Please read slowly, and use your imagination. Remember the scene in *2001: A Space Odyssey* where the *Blue Danube* waltz is sounding. The space shuttle is rotating on its axis trying to match the rotation of the space station. Perfect synchronization. Harmony. Striking beauty. A fantastic Orchestration.

For these first paragraphs only, we will use *Star Trek* as an allegory while the finely orchestrated waltz of *2001: A Space Odyssey* sounds in the background. We will say that the Federation (as in *Star Trek*) is the modern enterprise, the president of the Federation its CEO. The *USS Enterprise* is one of the many starships that are capable of performing a mission according to a predefined plan (implementing an Orchestration): an orchestration vehicle. Captain Kirk is the mission owner (Orchestration owner) for this orchestration vehicle. He is ultimately responsible for the different possible missions (deliverables) and maintains the captain's log. He is also responsible for overseeing the execution of the process (Orchestration) that fulfills the mission. He is not an expert in managing the engine; that's Scotty. He's not an expert in connectivity; that's Ohura. He's not an expert in dealing with the crew's ailments; that's Dr. McCoy.

The *USS Enterprise* needs to be able to use its own internal services (engine, molecular transporter, computer, view screen, navigation equipment, communication equipment, energy sources, crew services, etc.) at different points in the space-time continuum to complete the mission. The *USS Enterprise* also needs to be able to utilize services from external sources as well as respond to external stimuli such as attacks from the Klingons. The Klingons are the Federation's competitors. They too have starships, with different capabilities in terms of their internal services and ability to respond to external stimuli. The *Enterprise* needs to have security features such as electromagnetic shields that protect against Klingon attacks. Our *USS Enterprise* has many smaller fighter ships that also constitute orchestration vehicles. And the *USS Enterprise* is subject to Earth as the master orchestration vehicle.

So, an orchestration vehicle is a self-propelling system that can manage internal and external services according to a plan to fulfill a mission.

Hear the lazy loops of the *Blue Danube* as you imagine the Earth orbiting the Sun, spawning starships that embark on their elliptic trajectories, which spawn fighter ships that have their own missions and trajectories. If you have been able to see the fireworks while you hear the waltz, you are one of the visionaries who have the ability to orchestrate images and music in four dimensions in your mind, that fourth dimension being time. You can be an Orchestrator; you're a natural, you can play three-dimensional chess, you would be Spock in *Star Trek*. A complete Vulcan could not be an Orchestrator, but, as Spock, who is at least half human, you would be motivated to make an impression on the audience. If you immediately identified with Captain Kirk, you are a Process owner. You are to build the requirements for the Orchestrator to create a flight plan that will allow you to accomplish your mission. If you would rather deal with the health of the crew, or the engine room, or communications, you are a member of the functional staff that participates in the mission in your preferred role.

For Spocks and Kirks, this book is a must read; it's the reference for implementing orchestrations that accomplish missions. In the smaller fighter ships, the orchestrator and the captain are one and the same. Must read. In the Federation we have the Admiral, the Master Orchestrator (CIO). Must read. The Presidents of the Federation may want to read the first two chapters, just to know what the half-Vulcans are talking about. The McCoys need to read the first third of the book, the Ohuras the middle. All of the above may benefit from a careful read of the last third.

The Orchestration engine is the source of energy for the Orchestration vehicle; it's what makes it self-propelling, it's what puts the crew and the automated services in different points of space-time so that they can perform the appropriate activity within the necessary series to accomplish the mission. The Scottys are in charge of keeping that engine up and running. They need to know everything about the enabling technologies for that engine and how to discover and manage the services it will use. The Scottys needs to read this book; they need to master the middle third and read the last third.

We all know that without the Dilithium crystal-powered orchestration engine, the modern *Enterprise* would not fly. Its predecessors, based on fission reactors (work-flow engines), could never attain warp speed. This is why, initially, there was this idea that warp speeds could only be attained by ships without their own propulsion systems, ships that "sailed" on the legacy photon streams and slingshot trajectories of gravitational pull (messages on a universal messaging bus – EAI) from solar system bodies (the legacy applications). Scientists called this “loosely coupled” navigation. However, building the huge phaser belts (adapters or connectors) needed to create the photon streams around each star had proven to be too expensive and complicated and there had been a number of accidents due to the inability to foresee exceptions such as asteroids or worlds passing through the streams. On top of it all, there was the discovery of various parallel universes, whose photon streams were incompatible with ours.

Many visionaries had been telling us about the need for self-propelled orchestrations for a couple of decades. We didn’t want to listen. Self-propelling spaceships are like off-road vehicles. They are perceived as risky because they leave the main spaceways. This means navigating uncharted space, having to build charts of your own and being able to change them on the fly. We thought we were incapable of doing it. We thought that our legal counsel would never approve of it – too much risk. We thought this approach was for savage Klingons who wanted to reinvent the wheel and didn’t care to die. We wanted to follow the charts of our legacy and thought that just piecing them together would give us a chart of the Universe; we would be navigating the Perfect Economy! We were wrong. There was much more white space than charted space; furthermore, the Universe is in continuous expansion but charts are not. The Klingons were taking over the ever-growing white space. It took the bursting of the supernova bubble that took most of the phaser-belt adapter companies along with it for us to come back to our senses. We needed to go back to the principles of the Free Enterprise and pioneering that made our Federation great.

We finally reacted. Now, let’s get on with the principles of Business Services Orchestration (BSO) and then . . . let the waltz begin!

Navigating Through the Galaxy

Let’s be honest. The galaxy is large and complex. So we will divide it into two virtual sectors. Sector 1 comprises Chapters 1 through 4 and Sector 2 comprises Chapters 5 through 9.

Sector 1 was designed so that the Kirks, McCoys, Spocks, and Scottys could all feel reasonably comfortable in them. Although BSOs have the primary goal of increasing business performance, the orchestration of business services ends up weighing more heavily on the Spocks and Scottys than on the Kirks and McCoys. Therefore, Sector 2 is a place where only the Spocks and Scottys would feel comfortable.

In Sector 1, we have Chapters 1 through 3, which introduce the main concepts of BSO. All of those chapters have parts that are definitely in the Kirks’ domain of expertise, but few that are in McCoys’ domain. They are primarily oriented toward the Spocks. The Scottys will probably get impatient and want to move on quickly through Chapters 1 and 2, but we recommend that they look closely at the definitions.

Chapter 3 is where the desired architecture for the *USS Enterprise* is discussed. The Spocks and Scottys must master this chapter because it's the basis for understanding Sector 2; the Kirks may want to read it to understand the lingo.

Chapter 4 delves into the methodology framework for planning services that the enterprise will provide to the Federation. This chapter is a must-read for Kirks and Spocks. A crucial part of any stellar mission is the planning. The Scottys will probably fall asleep here and the McCoys will protest that they have real work to do. So please, Scottys and McCoys, read this chapter as a curiosity. The Kirks and Spocks should pay special attention to phase 2 of this chapter because it provides an end-to-end example that may prove to be very important for lucrative missions.

Sector 2 comprises Chapters 5 through 9 and is mostly for the Spocks and Scottys. This sector describes the technologies that enable development of BSOs.

Chapter 5 gives an overview of the important capabilities required from any platform used to build business services. The Scottys will find useful reference material in this chapter; the Spocks will find a good overview of the necessary capabilities. Even Kirk and McCoy may pick up some knowledge about the technologies, enough to be dangerous.

Chapter 6 starts talking about the technologies implemented on top of platforms and starts putting into perspective some guiding principles behind different integration approaches. Spocks among the readers will find this chapter most useful. The Scottys will find discussion of the technologies related to Web services interesting, but the Kirks and McCoys may just want to gloss over the chapter.

Chapter 7 is mostly for the Spocks. This chapter introduces metadata concepts related to the BSO reference model introduced earlier in the book. The Scottys may find it useful as well because the metadata are managed by the underlying infrastructure.

Chapter 8 discusses in detail the capabilities required to form a BSO system. This chapter talks about the basic and advanced concepts involved in process automation. These concepts are discussed through examples of an XML-based language, describing the data items that need to be captured for them. This chapter borrows the concepts and features available in many of the available orchestration languages that exist today.

Fundamental to BSO is its ability to integrate human services. It accomplishes that through work portals. Chapter 9 discusses involvement of human participants in the orchestration through the portal interface. This chapter will be of interest to Kirks, McCoys, Spocks, and Scottys alike.

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