Sage Accpac ERP Technology

Understanding the Benefits of the Sage Accpac Architecture

Table of Contents

Introduction 3

What an Architecture Is 4

- Stands the Test of Time
- Embraces Industry-Standard Technology
- Customizes Easily and Safely
- Deploys Flexibly to New Paradigms
- Scales Up as Your Business Grows
- Allows Greater Independence

What an Architecture Is Not 6

- Claim: Using Microsoft Visual Basic®
- Claim: Using C or C++
- Claim: Using Microsoft® SQL Server®
- Claim: Developing Exclusively for Microsoft Platforms
- Claim: The Product is Written in .Net so it has a Strong Architecture

The Sage Accpac ERP Architecture 8

- Separation of Core Business Logic
  - Interface Services
  - Database Services
- Object-Oriented Design
  - Embracing Industry-Standard Technology
  - Easy and Safe Customization
  - Large Objects Are Ready to Deploy Anywhere
- Design Flexibility and the Hosting Paradigm
  - Designed for Hosting
  - Multiple Companies and Security Systems on a Single Machine
  - Multiple Versions of a Product on a Single Machine
  - Language Selectable by User
  - Browser-Based User Interfaces
  - Linux

Conclusion 17

Introduction

Why is it important to understand the architecture of a business management software system? The answer is straightforward. When choosing a business management solution, you are making an investment. The immediate and ongoing costs include licensing the software, training staff, and adjusting business processes. Selecting a product with superior functionality will deliver a quick payback from more efficient operations. Selecting a product with a superior architecture will ensure that your investment will continue to pay dividends for many years, as well.

A superior software architecture has specific requirements that enable products built within its framework to adapt to fast-changing technology and stand the test of time. A large investment has been made in developing the Sage Accpac ERP architectural framework. Sage Accpac ERP is released in three editions, which, in increasing levels of functionality, are Sage Accpac 100, 200, and 500. All editions share the same architecture and code base.

To separate fact from fiction, it is important to understand the requirements of a true architecture. A software architecture exists in a sea of industry-standard technology, and its benefits cannot be stated without reference to this technology. But using industry-standard technology is not the same as having an architecture. Many software companies without a real architecture make pretentious claims based on the technology they use, but do not own. Using industry-standard technology is important, and the ability to adapt to this technology is one requirement of a great architecture. But the Sage Accpac architecture and its realization is technology that has been created by, and is owned by, Sage. This architectural technology is the key distinction between Sage and its competitors.

The costs of a large, complicated business application built on a poor architecture are significant. Such a system is very hard to maintain and takes a major reworking or rewriting to add new features or to take advantage of new technologies. This reworking of large amounts of existing code to add new features takes more time plus generally makes the product less reliable since there are many more places for bugs to get introduced into existing working features during the process of adding the new features. As a result, you have to wait longer to get less reliable updates, while your competitors using well-architected products are able to quickly embrace new technologies and features without any problems and enjoy the resulting productivity gains much sooner than the company with the poorly architected product.

Before reviewing the Sage Accpac architecture, this document attempts to set the record straight—not with fancy talk, but with real examples of real benefits that flow from a superior architectural design. It is hoped that you can use these examples to help distinguish fact from fiction.

What an Architecture Is

A software architecture is a foundational design specification—not a product. It is not a product any more than a real-world architecture is a building. A superior software product is the embodiment of a superior architectural design, just like a beautiful building embodies a beautiful real-world architecture. The architecture itself is the way that the product or building is organized. This organization determines the stability of the product or building and the limits to which it can extend its overall use and functionality. The way a software product is organized determines how well, or how poorly, it fits into the software landscape of industry-standard technology, how well it can adapt to new technologies and business and computing paradigms, and how well it can scale as transaction volumes and/or the number of users accessing the product increase.

What are the organizational fundamentals of a great business management software architecture? First, the separation of core business logic from interface and database services is essential. With these three layers separated, the core business logic can be connected to new databases and evolving interface components and devices—without costly reimplementation efforts and with the ability to maintain a single business logic code base. The three layers are separated by two "interfaces" (commonly called an Application Program Interface or API), which are the "plug-in points" that join the layers together.

Then, the core business logic itself must also have its own organization. When organized properly, every single piece of business logic is accessible from the outside layers. This is important if business processes embodied in the logic are to connect to industry-standard technologies that can drive the whole application. In the object-oriented software world of today, well-implemented business logic is organized into a collection of components or objects. As objects, their behavior can be modified and customized without making changes to the object itself. In object-oriented terminology, the objects can be "sub-classed" and one object can "inherit" properties from another.

The benefit of a strong architecture and good implementation is that the resulting product:

- Stands the test of time.
- Embraces industry-standard technology quickly and naturally.
- Customizes easily to fit the special needs of your business.
- Deploys flexibly to paradigms such as Cloud Computing or SaaS.
- Scales to the changing size of your business.
- Allows greater independence to make choices regarding, for example, database or operating system.

Stands the Test of Time

A product with a sound architecture does not need to be rewritten every time technology or platforms change. A company with a true architectural message does not change the message every year or two as technologies and platforms change. A Windows® Graphical User Interface (GUI) and an Internet browser-based user interface should both be useable within the same framework. So should different database choices. This does not mean that an architecture does not adapt and shift. It must—the pace of change is too great in today's world. The very essence of a good architecture is that it adapts easily. But the fundamental tenets of a great architecture do not change. If the message keeps changing, it is because the architecture it touts is flawed, which is costly to sustain as a developer and as a user.

"The very essence of a good architecture is that it adapts easily."

Embraces Industry-Standard Technology

New software technologies that interconnect software applications are among the most important software technologies for a business application. The Microsoft® Component Object Model (COM) allows multiple desktop applications to work together. For communication over the Internet, Extensible Markup Language (XML) allows servers and Internet devices to exchange information. For wireless devices, there is a Wireless Application Protocol (WAP). To take advantage of these technologies, a product must have defined contact points (or interfaces) into which the technology can be connected. An architecture must define these contact points and have a consistent way for the application to communicate with them. The issue is not that an application can talk to a particular technology, such as COM or XML, but that the application has contact points designed for any interconnecting technology that becomes a standard.

Customizes Easily and Safely

Almost every business has special needs, so some degree of customization will always be desirable. A variety of tools and techniques are used for this purpose, such as importing or exporting information, adding optional fields, customizing screens and reports, writing macros with Microsoft's Visual Basic® for Applications (VBA) language, and changing the core business logic itself. A strong architecture shows itself not by the fact that a particular tool or technique can be used, but that any tool or technique can be easily connected and used. Further, a strong architecture enables the whole application to use this tool—every single component, not just a select few. Can VBA intercept a few data entry fields, or can it drive the complete business logic of an application? Can every database table be exported, or only a select few? Finally, it is important that when a product is being customized, the core code and data are protected. The best technique for achieving this is to employ objects whose functionality can be inherited.

Deploys Flexibly to New Paradigms

Today's products must adapt not only to new technologies, but also to new information technology (IT) and business paradigms such as Online Hosting, which is a rapidly expanding business and IT delivery model. An Online Host is a company that rents hardware and software in a secure location, and supplies connections to personal computer devices in your business in a one-to-many (single installation to many "renters") manner. The Host also provides services that maintain and manage the hardware and software infrastructure, alleviating the renter of those often costly necessities. You may not want to use the services of a Host today, but it is important that the product you purchase today can be deployed flexibly, not only to accommodate this paradigm, but also to accommodate paradigms that will emerge in the future. A key deployment factor is the ability to deploy a software product in multitier environments, with the same core business logic able to run on database servers, application servers, or client machines. The flexibility of Sage Accpac ERP even extends beyond Windows, allowing you to use Linux® as a database server. Flexibility of deployment is a sure sign that the underlying architecture is sound.

Scales Up as Your Business Grows

To scale, a program must perform well over a range of table sizes, transaction volumes, and user counts. A business management software product must run efficiently over a wide range of customer, vendor, account, item, order, and invoice sizes. To do this, a product must interact efficiently with its database services. This means that the underlying architecture must be able to take advantage of database-specific operations to tune the product's performance. A stored procedure is an example of a feature specific to Microsoft's SQL Server® database that is often used to tune performance. Using a product that supports multiple databases is another way to ensure that you can scale. There are limits to how quickly certain platforms can process transactions, and a fast-growing business could outgrow their current database technology. A strong architecture offers database choice, can scale server platforms, and will also use individual features of each database to provide maximum performance in any setting.

Allows Greater Independence

No one wants to be locked in to a single vendor or a single way of doing things. A great architecture allows freedom of choice and independence from single vendor lock-in across the whole software stack. Once you have chosen a business application system with a great architecture, you should then be able to choose the best operating system to use such as Windows or Linux®, and the best database server to run such as Pervasive.SQL, Oracle®, or Microsoft SQL Server. You should be able to choose from a large number of third-party add-on products that seamlessly integrate in and greatly enhance the core business functionality. You also want the ability to change these as licensing rules change and as new versions of one or the other are released with features that could benefit your business. You need the independence to be able to choose and not be locked in.

What an Architecture Is Not

Many software products have no architecture at all, intermixing layers and code in a haphazard manner. Many lack even basic object orientation at the application level. Does that stop them from producing "architecture" white papers? Of course not. So what do they talk about? Technical details of their products are put forward as "proofs" that they have some grand architectural plan. Microsoft technology is often cited, as if by using that technology their products magically inherit an architecture. Let's look at some of the claims that are made.

Claim: Using Microsoft Visual Basic Pervasive PSQL (VB) means a product has a strong architectural foundation.

This is often stated by products that are written as giant monolithic Visual Basic programs. It usually implies that even though their product isn't modular or extensible that somehow they have an architecture just by being written in VB. The claim's credibility is based on Microsoft's cachet. But the truth is that the implementation language has nothing to do with whether a product has an architecture or not. Basically, good or bad programs with good or bad architectures can be produced in nearly any programming language. Using VB has certain advantages: For example, the rich variety of controls that are available for use and the wide market penetration VB enjoys. However the architectural value comes from how the program is put together in a modular componentized extendable customizable fashion. Often it's better to write different parts in whichever programming language or system is best for the job.

Claim: Using C or C++ means a product has a strong architectural foundation.

This can get richly twisted. For example, it has been said that a company's proprietary language has been implemented in C, just like Microsoft Windows and Visual Basic, and therefore the company's products, written in their proprietary language, have somehow acquired an architectural foundation. We have even seen statements such as "Microsoft will never change C/C++ because Windows is written in C/C++." You can see how silly this statement is by seeing how strongly Microsoft is now pushing C#, which could be called a "changed C/C++." Once again, the language of implementation does not create or exclude an architecture.

Claim: Using Microsoft SQL Server for a database means a product has an architecture.

Microsoft SQL Server is a database. A product that uses it may use it well or poorly. Using Microsoft SQL Server-specific features to tune the product means that the product probably runs well on SQL Server. A product that supports multiple databases and takes advantage of specific features of each demonstrates that there is a superior architecture in play. By using a common database interface, all database-specific functionality is managed by a database driver, a single point of contact in the code. This does nothing to impede performance, as the common interface can take advantage of any feature of a specific database. This kind of design also tends to improve reliability, as the same core business logic gets tested in a variety of environments.

Claim: Developing exclusively for Microsoft Platforms means a product has an architecture.

Developing for Microsoft's platforms has some advantages. Often the tools work well together and marketing support from Microsoft is strong. Using "best of breed" tools also has some advantages. As strong as Microsoft is, there are products like Oracle's database and the Linux operating system that are in demand by customers for very good reasons. The truth is that either a "Microsoft only" or a "best of breed" strategy may be a good approach to take. But let's be clear—this has nothing to do with having or not having an architecture.

Claim: The product is written in .Net so it has a strong architecture.

Microsoft® .Net™ is the basis for the current family of programming languages supported by Visual Studio®. These are an updated set of programming languages with a common runtime and a common programming framework. There is certainly a lot of good technology in the set of .Net libraries. However just being written for .Net doesn't mean you have a good architecture; it is just an implementation decision. You can just as easily implement a poorly architected system in .Net as you can in any other programming environment. In .Net you have the additional challenge of providing choices; the easiest way to do things in .Net is usually to use all Microsoft application servers providing single-vendor lock-in of the whole system. Also currently .Net only runs on Windows since .Net support in Linux is not complete and faces several legal hurdles.

The Sage Accpac ERP Architecture

Sage Accpac has a superior architecture based on separation of core business logic from user interface and database services. The truth is that no other player in the midmarket has anything approaching the strength of this architecture.

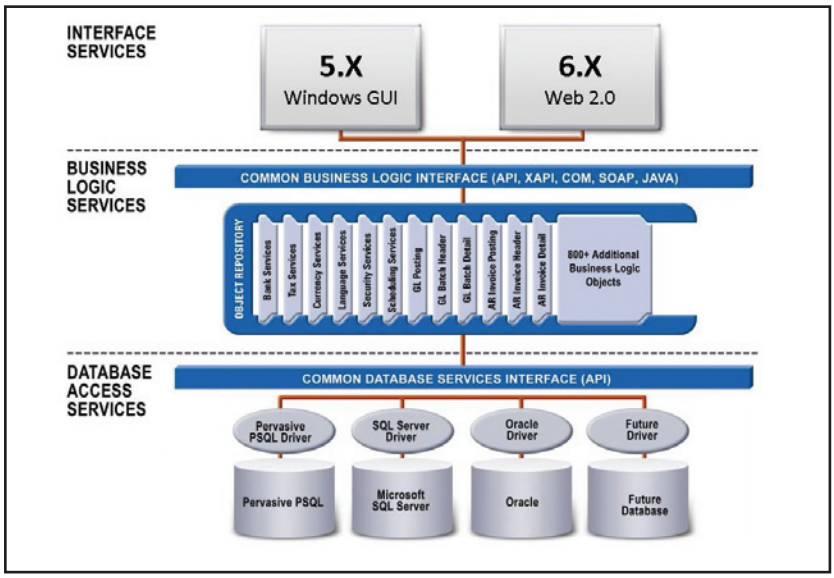

Figure 1—The Sage Accpac ERP architecture separates user interface services, core business logic, and database services, and is implemented in a completely object-oriented fashion. The architecture allows for easy adaptation of new user interfaces and databases, while maintaining a single common code base for business logic.

Separation of Core Business Logic

The Sage Accpac ERP architecture, as depicted in Figure 1, is designed to stand the test of time by isolating and minimizing dependencies on workstation and network operating systems, databases and user interface environments. The architecture features a strict separation of interface services, business logic services, and database access services, with interfaces that provide consistent layer-to-layer communication.

Interface Services. Sage Accpac user interface code is separate from business logic. If a new user interface has to be added, this can be done without affecting any of the core business logic. Exactly the same core logic runs with Windows GUI, an Internet browser or a wireless device. Other kinds of interfaces to the business logic, such as macros and import/export, communicate with the business logic through a Common Business Logic Interface (CBLI), which has various components (API, XAPI, COM, .Net). This wealth of interfaces allows programs written in most programming languages to interface to Sage Accpac, such as Visual Basic, Delphi™, Java®, JavaScript™, Perl, C#, C/C++, and J#. This wealth of interface services allows integration with an expanded family of end-to-end enterprise products, allowing Sage Accpac to expand beyond the usual roles of accounting modules. All these other applications are tightly integrated with the core accounting modules by interacting with the CBLI.

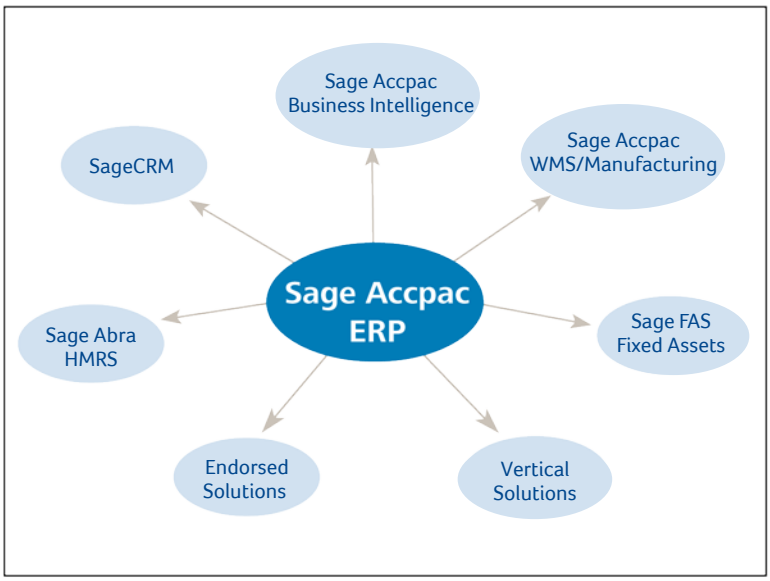

Figure 2—Sage Accpac Accounting is at the core of a whole family of end-to-end enterprise applications.

A well-implemented product often shows its virtuosity in ways beyond what you would ordinarily expect. Indirectly related to the separation of the user interface layer is the fact that Sage Accpac data entry screens can be run in two different ways—even in the same operating environment. Figures 3 and 4 illustrate an order entry screen being run in "spreadsheet" style and "pop-up" style. This versatility allows the product to adjust to different user skill levels.

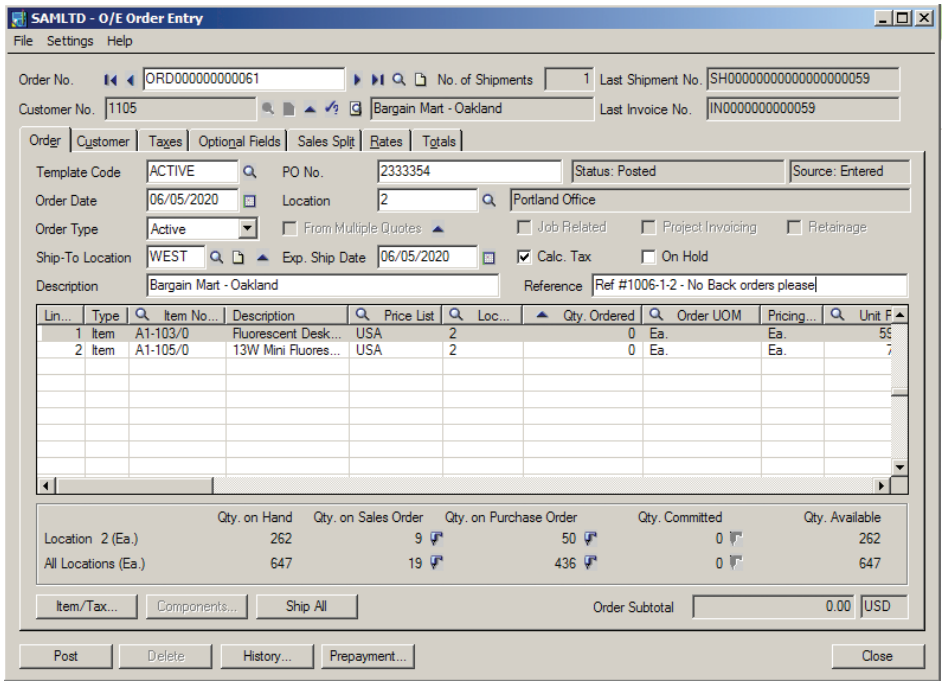

Figure 3—A Sage Accpac Order Entry screen with "spreadsheet" entry.

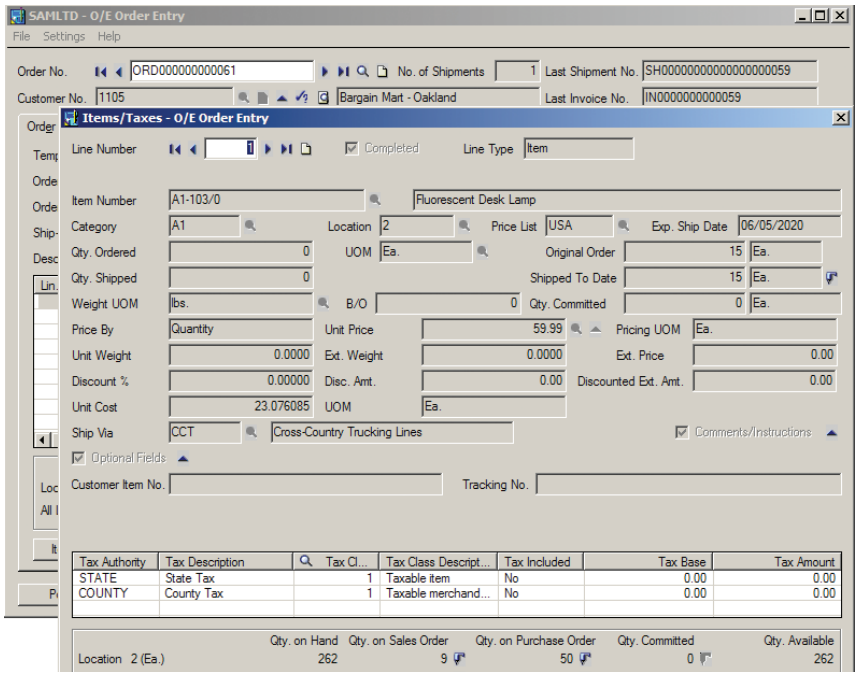

Figure 4—A Sage Accpac Order Entry screen with a pop-up "detail" screen for faster data entry.

Database Services. Sage Accpac currently supports three database choices: Pervasive.SQL, Microsoft SQL Server, and Oracle. Additional databases will be added in the future. When speaking of different databases, different versions of the same database deserve some notice. Database versions change frequently, and the Sage Accpac business logic does not depend on the peculiarities of a single database version. Because all database access is abstracted into a Common Database Services Interface (CDSI), adaptation to a new database version can be done quickly in one place (the database driver), with less detailed testing of the whole application required.

Object-Oriented Design

Sage Accpac business logic is implemented as a collection of more than 750 objects that communicate with one another in a common object repository. As an example, Figure 5 shows the objects that are involved in receiving inventory items and the messages that they pass to one another.

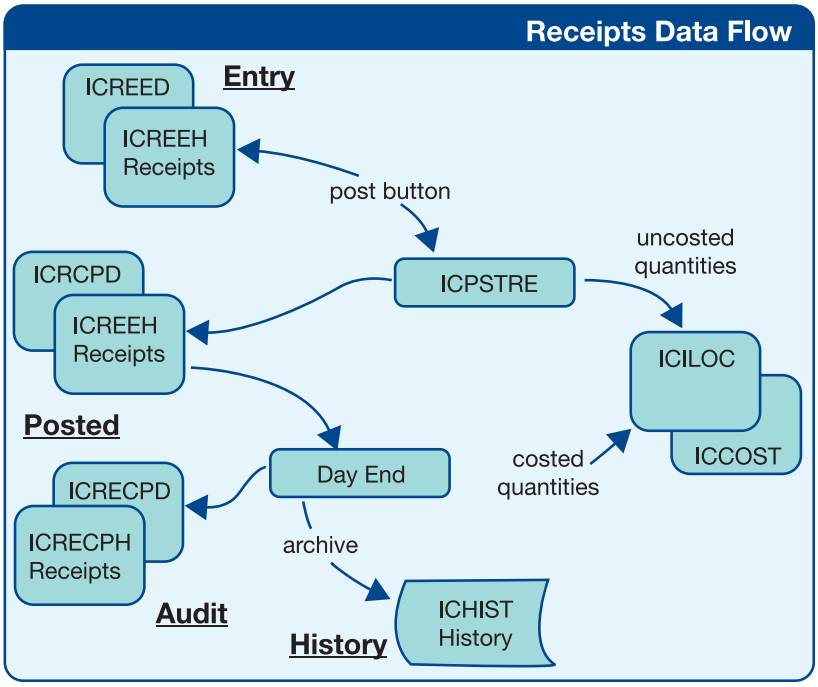

Figure 5—Sage Accpac business logic objects are implemented on a common communications framework and are implemented by "major" function. This communications framework ensures consistent interaction among all objects, and the "large" object size allows complete business functions to be referenced, which provides easier customization by third parties.

Embracing Industry-Standard Technology. A well-implemented set of objects behaves in a consistent and uniform way (follows protocols). The objects are constructed from a common "template"—business logic that all objects share and that ensures that protocols are consistent. If new protocols are necessary, it is usually possible to simply change the common template and re-create ("re-instantiate") all objects with the enhanced template. New protocols are used to connect to new technologies and standards. This organization creates incredible leverage. To make all Sage Accpac objects behave as .Net objects requires only that the core protocols are .Net enabled. To make all Sage Accpac objects XML enabled requires only that the core protocols are XML enabled.

We can continue: **Every single** Sage Accpac business logic object can be driven by VBA; every single business logic object is Internet accessible through the Sage Accpac SOAP Web Service; every single Sage Accpac table can be exported or imported. With today's constant introduction of new technologies and standards, the consistency and uniformity of an object-oriented design is a key element of any architecture.

Easy and Safe Customization. In an object-oriented design, objects inherit functionality from one another by "sub-classing" functionality and modifying part of the behavior. This means that customization is easy, as a wide range of development and customization tools can be used to manipulate objects. Availability of third-party software products that modify the core behavior in Sage Accpac is an important form of customization. The inheritance model Sage Accpac uses allows multiple third-party products to modify the core behavior of Sage Accpac objects, with each third-party product behaving as though it is the only third-party product present. The customizations are also safe, because none of the code of the original object has changed.

Figure 6 illustrates inheritance to two levels. Core Object 1 is part of a program and communicates with other objects through four messages—a, b, c, and d. When another object, Customization Object 2, subclasses Core Object 1, it modifies the behavior of two of Core Object 1's messages, a and b. A third object, Customization Object 3, can sub-class Core Object 1 and Customization Object 2, modifying messages a and d.

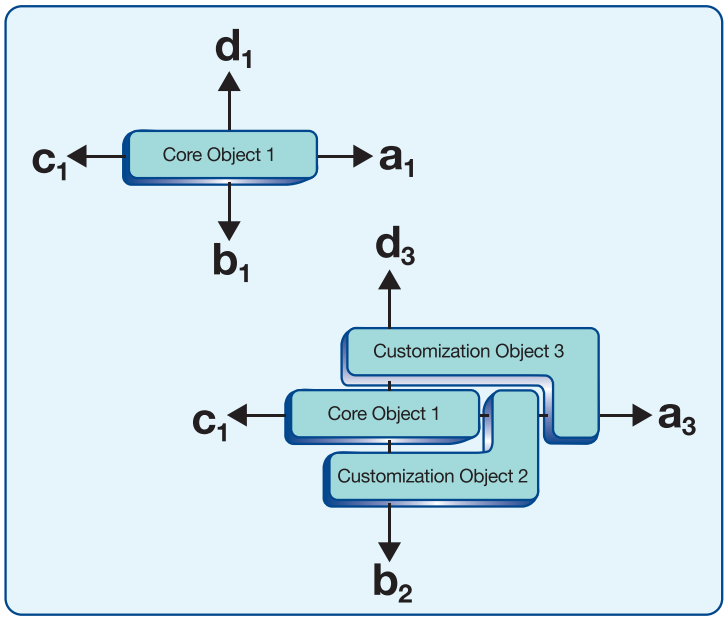

Figure 6—Sage Accpac business logic objects support multilevel inheritance, providing customization options without jeopardizing the integrity of the core business logic.

Large Objects Are Ready to Deploy Anywhere. What does it mean for an object-oriented design to be implemented at the right scale? It means that objects are the right size. For analogy, a real-world object like a car is thought of as its major components: An engine, a chassis, a transmission, axles, and so on. Smaller objects like bolts, washers, and wiring are less functionally significant in the overall design. Sage Accpac has millions of lines of code, but only slightly more than 600 objects. Competitive products that use object-oriented tools, but have thousands of small objects, take more effort to manipulate, because each high-level task involves patching together ten or more small objects with snippets of code.

When large business logic objects are used, these large objects represent real, high-level application tasks, such as posting to a general ledger. This high-level functionality is available to any user or third-party developer who wants to customize—without the need to modify the actual business logic code.

Design Flexibility and the Hosting Paradigm

The Sage Accpac ERP architecture is designed for flexible, scalable, and cost-effective deployment in a Hosted environment. A key cost factor in the hosting scenario is how many independent companies can be run on one server and managed easily from a central point. The Sage Accpac architecture has a unique combination of design features as illustrated in the following table and product screens:

Sage Accpac Design Features	Sage Accpac Hosting Advantage
Site directories separate unrelated companies running on the same server. Each company identifies its own security and access rights for groups of users.	A single server can be used whether there are 50 users in one company or 50 companies with ten users each. Sharing a single machine does not mean having a single security methodology.
Multiple versions of the product can be installed on the same machine. Installation on a machine does not activate the version for all companies at once. Users or consultants can activate a version one company at a time.	As the software moves from one version to the next, conversions on a single server machine can be phased as convenient. Companies may therefore control the timing of a conversion process.
The language is identified for individual users of a company. Sage Accpac currently supplies product translations for simplified Chinese, traditional Chinese, French, and Spanish.	A single server machine can be used for companies in different countries or to run a company that has multiple user language requirements on the same machine.

Multiple Companies and Security Systems on a Single Machine. In an Online Hosted environment, multiple companies may be set up on a single machine (or server), each with its own security systems. If business management software does not support this, then each client must be set up on a separate server. This is not a true hosting model, but rather a managed services model. The problem with this approach is that it limits fail-over and load balancing flexibility, which are key high-availability factors. In a high-availability hosting environment, large, redundant network storage devices are used to balance loads so that multiple servers are available to process data for multiple companies. Multiple security systems on a single data store is a requirement for this setup. Figure 7 shows the Sage Accpac User Authorization administration service and the users and the applications they can access.

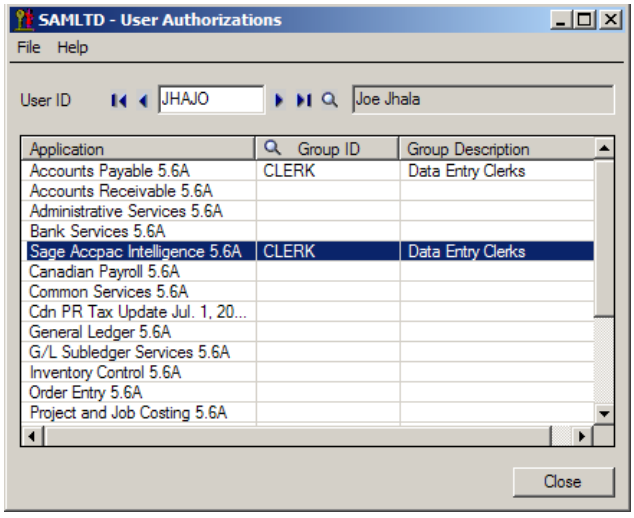

Figure 7—Sage Accpac security is well suited to the hosting paradigm, as it allows security to be defined by company and user on the same physical machine.

Multiple Versions of a Product on a Single Machine. Multiple versions of Sage Accpac modules can be installed on the same machine. Installation on a machine does not activate the version for all companies at once. Users or consultants can activate a version one company at a time. Figure 8 illustrates the Sage Accpac System Information screen, which contains installation information showing multiple versions of Accounts Receivable and Accounts Payable installed on the same machine. The tick mark indicates, for the active company, which application and version is currently active.

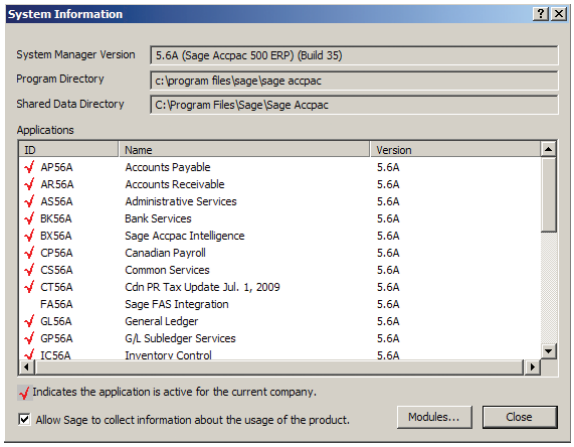

Figure 8—Sage Accpac allows multiple versions of different modules to be installed on the same environment and activated by individual company.

Language Selectable by User. Several remote offices in different countries may all need to access the same data. Different countries often mean different languages. It is desirable that the same data be accessed by users using different languages at the same time. Sage Accpac users within the same company may be set up to use different languages by the administrator, as shown in Figure 9.

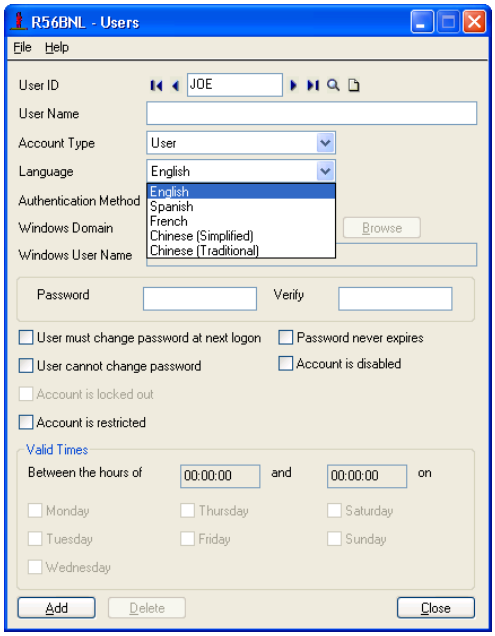

Figure 9—Each user may be assigned a separate language within the same company.

Once users sign in, they each operate the product in a different language, as shown in Figures 10 and 11.

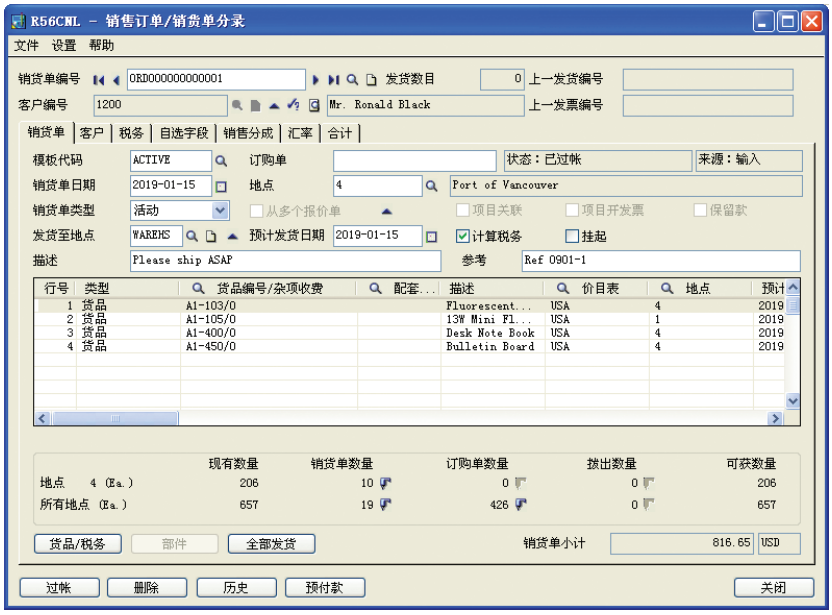

Figure 10—The Sage Accpac Invoice Entry screen in Simplified Chinese.

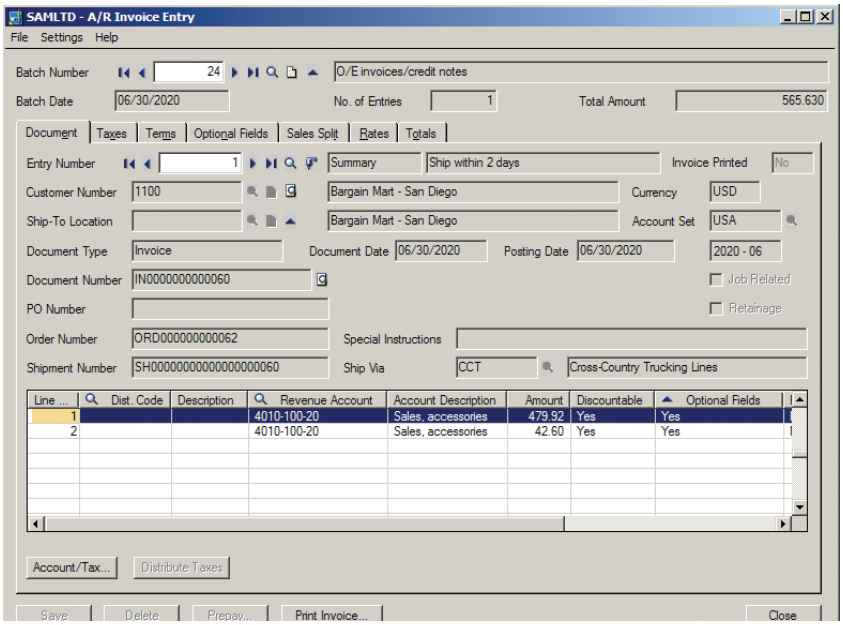

Figure 11—The Sage Accpac Invoice Entry screen in English.

Browser-Based User Interfaces. It is impossible to talk about the promise of hosting without discussing whether an application can be "run from a browser." It seems as though the entire software world wants to run its applications from a browser, without going through a terminal services interface: Everyone has announced an intention to do so. So why have so few delivered? The answer is that the task of rewriting hundreds of user interface screens with complex user interactions takes a lot of effort. If user interface code is intermixed with business logic and database services, the effort is exponentially greater.

An application like Sage Accpac has an architecture with the user interface code in a separate program that talks to the business logic by sending and receiving messages. The effort to implement Sage Accpac in a browser is further reduced by taking advantage of the business logic's object orientation. Sage Accpac has leveraged its objects by developing common ActiveX "controls" (buttons, fields, and forms on a screen) whose behavior is controlled to a large degree by the business logic objects. These controls can be run from a "desktop" environment or a "browser" environment. Figures 12 and 13 show the same Bank Services maintenance screen running in both Windows GUI and Internet browser environments.

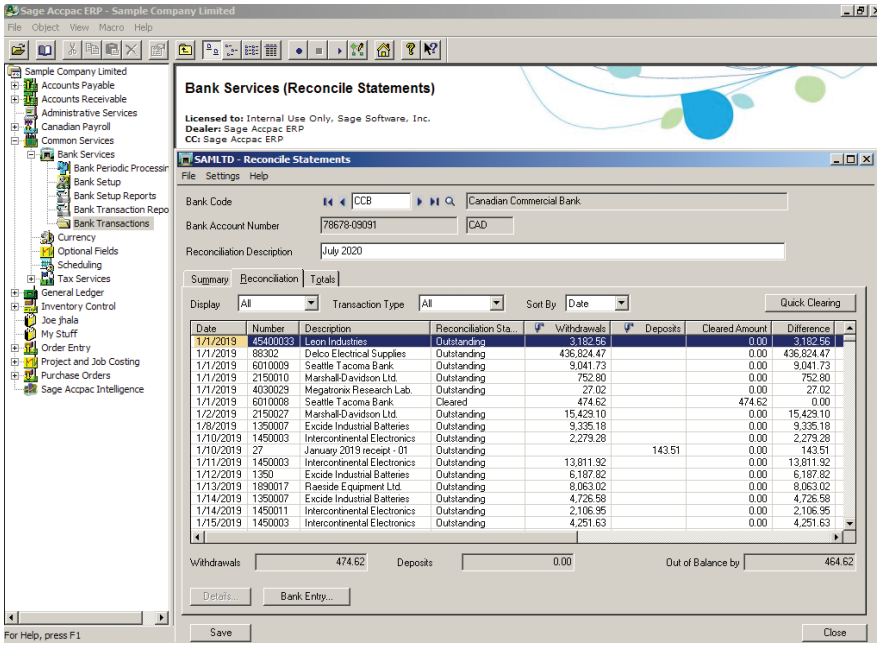

Figure 12—Sage Accpac Bank Services screens in a Windows desktop environment.

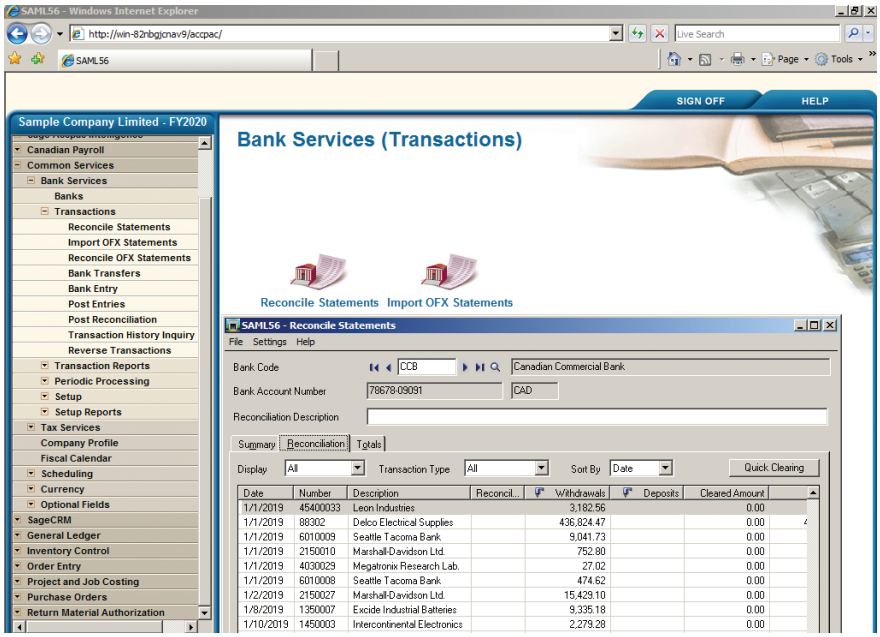

Figure 13—Sage Accpac Bank Services screens running within an Internet browser.

Linux. Sage Accpac allows you to run on a Linux database. In today's competitive market, Linux has been steadily gaining market share due to its stability, performance, functionality, and total cost of ownership. Many companies are adopting a wide-scale deployment of both Linux servers and Linux workstations. Sage Accpac can happily exist in a mixed environment of Windows workstations and both Linux and Windows servers, all of these working together seamlessly.

Conclusion

A software architecture reflects the way a product is organized. The way a software product is organized determines how well, or how poorly, it fits into the software landscape of industry-standard technology as well as existing and evolving technology and business paradigms. It also determines how well it scales from both a transactional and user-count perspective. The foundation of a superior business management architecture is that it is organized into layers: There is a separation of core business logic from interface and database services.

The benefit of a strong architecture and good implementation is that the resulting product will:

- Stand the test of time.
- Embrace industry-standard technology quickly and naturally.
- Customize easily to fit the special needs of your business.
- Deploy flexibly to new paradigms.
- Scale to the changing size of your business.

Allow greater independence to make choices such as database or operating system. A superior architecture is not created overnight. Creating a product with a superior architecture takes a huge investment over a number of years. Sage has made that investment. Over the past decade, as software fads have come and gone, the architectural message and object-oriented approach of Sage Accpac has remained unchanged. The Sage Accpac ERP architecture stands the test of time and is ready to deliver value to your business for years to come.

The information contained in this material represents the views of Sage on the issues discussed herein current as of the date of publication. As market conditions are always subject to change, the information contained herein shall not be interpreted as any commitment from Sage. This material is for informational purposes only and Sage makes no warranties, expressed or implied.

Sage
13888 Wireless Way, Suite 120
Richmond, BC
V6V 0A3
1-604-207-9480
www.SageAccpac.com

10-20904 02/10