David Sprott

Componentizing the Enterprise Application Packages

Tomorrow's customers will demand the ability to buy, reuse, and build their competitive-edge solutions: Can the package vendors adapt in time?

The so-called enterprise resource planning applications market was one of the fastest growing and most profitable areas of the software industry during the last three years of the 1990s. Some of this was clearly attributable to the Y2K effect. If you still had time, replacement was often easier than continuing to throw good money after bad into aged legacy applications. But, also discernible, there was a strong desire in many organizations to acquire functionality rather than develop custom solutions, particularly for relatively unexciting transactional applications that did not differentiate the business significantly. But we should also note that some of the packaged application vendors acquired a poor reputation, being better known for the time and cost involved in implementation than the resulting business benefits. So, we observe with interest that enterprise application providers such as SAP, Peoplesoft, Oracle, Baan, JD Edwards, and many others have been investing heavily to upgrade the architecture of their applications over the past two years. No prizes are awarded for guessing why.

The market-leading enterprise applications represent some of the largest, most complex applications on the planet. The complexity comes particularly from the highly generalized nature of the packaged applications and the need to adapt and rapidly evolve to meet requirements in many different situations. The packaged application providers therefore have had a genuinely mission-critical problem to solve. How to continue adding new functionality rapidly and at low cost, while making it easier for new customers to implement, and existing customers to upgrade to that new functionality? We should not be surprised therefore that these vendors were relatively quick to recognize the benefits of components.

ERP VENDORS FACE A RATHER MORE COMPLEX PROBLEM THAN ORGANIZATIONS MANAGING IN-HOUSE DEVELOPED APPLICATIONS.

Enterprise applications are an outstanding case study for the entire software industry. Most of them demonstrate all of the real-world problems that typical businesses have. Legacy applications that simply can't be rewritten, monolithic code not built for easy maintenance, multiple design and execution technologies that need to be integrated, demand for new technology support and last, but by no means least, customers that won't wait years for a solution.

Reprise: What is a Component?

The concept of componentization is common in many engineering disciplines and industries other than software. For example, automobiles, computers, and mobile telephones are all assembled from parts that can collaborate within an architecture. The parts are typically not a homogeneous set, but comprise many different types and standards. For example, in the PC, we have the motherboard, the chips, memory, the keyboard, cables, drives, casing and so on. However, there are agreed standard interfaces that allow the disparate sets of parts to be assembled, to interoperate and to be upgraded with newer parts that conform to the same interface. Some of these standards are dictated by the manufacturer; many are de facto industry standards. When Dell or Compaq release a new Pentium III model, generally a minority of the part numbers are new to that model and the new product will be an evolution and upgrade incorporating older and newer part numbers. This is made possible because the product line is based on an architecture designed to facilitate an evolutionary life cycle.

Regrettably most software produced to date has been delivered as monolithic code and does not implement the very sensible concepts of componentization. Today, however, there is widespread consensus that component-based techniques are essential quality requirements for software. The primary rule is that you should be able to make use of a software component without reference to its internals [2]. The only thing you need to know about the component is the published interface that specifies the contractual agreement necessary to use the services provided. It is also important to recognize that a component is not restricted to technical implementations that use COM, Enterprise JavaBeans, or CORBA. A component can be any form of implementation providing it adheres to the concepts of separation, interfacing, and standardization outlined previously. There are advantages to using the three component standards mentioned, but it would be incorrect to assume that a contract between two processes implemented using XML and messaging software is not capable of providing many of the benefits of componentization.

The advantages of a componentized software architecture are therefore very similar to the architecture of a personal computer, which is designed for incremental evolution of processor, systems, functions, utilities, and so forth. The parts of a componentized software product may be constructed by potentially many different manufacturers and then brought together to be assembled very rapidly. The assemblers do not need to be experts in the internals of each assembled component.

It is useful to consider that the software component has two fundamental constructs that deliver radically different benefits. First the component is a separate, encapsulated entity and by virtue of its separation is easier to manage, upgrade, collaborate with and so on. The granularity, scope boundaries and internal cohesion are important attributes of the component. A fine-grained component will be simple to upgrade, have fewer relationships but will require more management because there are likely to be many more parts to meet the requirement. In contrast a larger component will be easier to manage but will require more effort to modify and implement and, because the scope of the functionality is much broader, the impact of change is much greater. These issues are primarily of interest to the developer. Of course as a consumer I "should" be interested in these characteristics because they are quality measures that will affect the ability of my supplier to respond to my needs, but in principle the granularity and boundaries

of the component are architecture and design issues.

The second aspect to consider is that the component delivers one or more services. The service is the function that the user (another server or client) is provided by the component. In general useful services will be those that are designed to operate at a level of abstraction that is understandable by the user. A service might therefore be something very simple such as a LOOK UP SYNONYMS service provided by a Thesaurus component, or as comprehensive as an UPDATE NEW CUSTOMER INFORMATION service provided by a Customer Relationship Management (CRM) application. A series of services provided by multiple, disparate components might be integrated into a common workflow that performs a unified business purpose. What is important is that the interface and service provided are independent of the underlying implementation. As illustrated in Figure 1, an implementation might be provided by a legacy VSAM database today, but I could easily swap out that component and replace it with an object or relational database with no effect on the user of the service, providing the interface remains unchanged.

In the context of application packages it is critically important to understand that the two concepts encapsulation and service provision deliver radically different benefits and that most application packages have concentrated on the service provision at the expense of replaceability and upgradability [1].

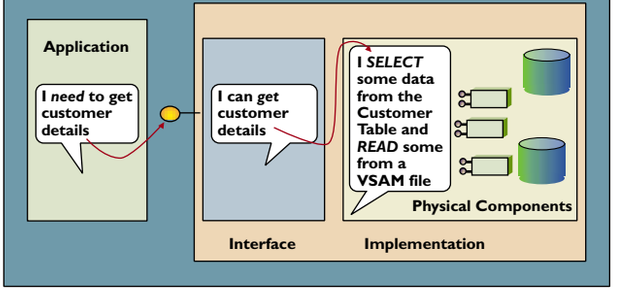
The Challenge for ERP Vendors

As discussed earlier, the ERP vendors face a rather more complex problem than organizations managing in-house developed applications. The problems may be examined from two important perspectives—the product development and marketing challenge for the vendor and the implementation and integration problem for the consumer.

Product development and marketing. Many ERP vendors' packages have evolved from simple beginnings and are comprised of large functional modules such as Finance, Logistics, and Manufacturing. These modules are often called components by the vendors, but in reality they are just large application systems, not components. Some vendors realized they needed to componentize their applications to make their own internal product development operations more effective. For them it was a matter of competitive necessity to deliver new functionality faster than their competitors. Organizations such as SAP embarked on exercises to break up their larger-scale application modules. Others such as JD Edwards rebuilt their applications as components recognizing the commercial necessity for adaptability in the development phase. It is interesting to note, however, that many vendors quickly stopped talking about their internal design and concentrated on communicating their progress in delivering a service provision layer.

The vendors all recognized their packages would need to interoperate with other environments, both packages, in-house applications and legacy. The widespread package implementation activity inevitably led to a huge demand on the vendors to make implementation easier, and in particular context to this article, to facilitate integration. The response of some companies including SAP and Baan was to create service-based architectures that enabled business transactions and information requests from outside the core application. Some companies, notably SAP, encouraged third parties to create a thriving market for add-on components surrounding the mothership application, providing a huge diversity of solutions, enabled by the open, implementation-independent services. Most packaged application vendors were slow to recognize the need to facilitate integration and a number of third-party organizations stepped in to fill this gap by enabling packages to talk to each other.

Figure 1. What is a component?



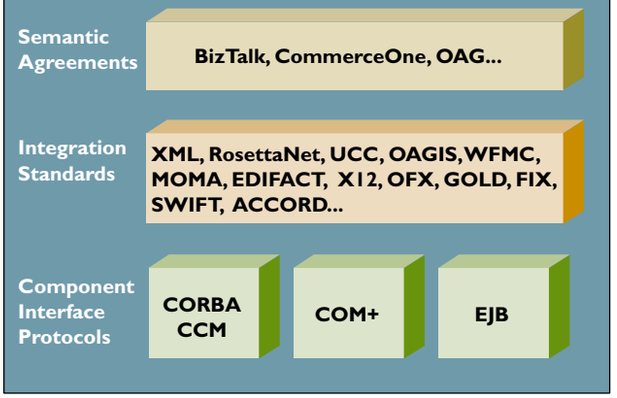
The current status of componentization could be considered as "work in progress" for most of the leading application suppliers, but the benefits are clearly starting to be realized. What is interesting to note is that while this radical architectural change is a direct response to customer needs, the actual change has been motivated entirely by the suppliers. Many existing customers are only starting to understand the implications and opportunities. A significant majority of packaged application users are currently using monolithic releases and will need to go through at least one or possibly two release upgrades before being able to take advantage of the interface architecture and incremental release programs.

Customer integration. For customers buying packaged software it is perhaps natural to focus initially on selecting and implementing the package

THE THRIVING AND EFFECTIVE APPLICATION INTEGRATION MARKET HAS CHANGED THE DYNAMICS OF THE APPLICATION PACKAGE MARKET QUITE DRAMATICALLY.

rather than examining in detail the requirements for integration. It is an unfortunate fact that a significant proportion of package implementation projects undertaken in the late 1990s massively overran time and cost budgets. It is also a fact that the reasons for overrun were often related to integrating the package with other application environments. This was not usually a technical issue but related to differences in semantics and business rules between different applications that were never intended to collaborate. In fact you might suspect that it was against the manufacturer's interest to facilitate collaboration with other vendors' packages. The packages were and are with few exceptions, designed with an inward focus. Yet many organizations deliberately chose what they referred to as "best of breed" meaning they were choosing the modules with the most relevant functionality, little realizing that to make the modules actually work together was likely to be a major effort which in itself might compromise the functionality of each module because of a lowest common denominator effect.

Figure 2. Multiple standards.



But in the run up to the Y2K date turnover, the mad rush to create and acquire packages seemed to affect the judgement of many vendors and consumer organizations. It is interesting to note that the fortunes of many of the package vendors, in terms of profitability and stock price, have subsequently suffered badly. Because most of the vendors left their transition to components and service-based interfaces too late for the peak of implementation activity, and because they failed to see the requirement for interoperability standards, the vendors created an opportunity for others to provide specialist products and services that enabled packages from different stables to collaborate. Fueled by the massive implementation activity, this market space grew quickly to form what was initially referred to as enterprise application integration, or EAI, and more recently simply application integration.

Application integration techniques and technologies [3] are part of the natural evolution of application delivery that includes improved software componentization and the increasing acquisition of packaged software. In the past our focus has been more on integration of in-house developed applications and components, which is easier when there is a common technology base within the project or same IS department and can be changed to enable integration. Integration was then just seen as part of the application development process. Now we must observe that application integration is effectively becoming a discipline in its own right. We can also see that the huge demand for e-business applications will drive the requirement for continuous reintegration of application packages as new integrated business processes cross organizational and company boundaries and changing business processes require constant reconfiguration to meet business model changes.

Application Standards

Arguably the application integration market would have emerged regardless of what the application package vendors did, but it must be said the thriving and effective application integration market has changed the dynamics of the application package market quite dramatically. The package vendors are now forced into working in collaborative ways. Two years ago the industry was expecting the component models and protocols COM and CORBA to be the

obvious integration platform although it was unclear how incompatible packages would be integrated at the semantic level. The good news, however, is that today we have much more stability because standards that address the broader need have emerged and been widely supported and adopted. The primary reason for industry consensus and rapid action has little to do with altruistic motives of the package vendors, but more related to the driving force of the Internet and in particular the growing e-commerce market, which demands common standards in order to enable trade.

Figure 2 shows three important classes of standards. The first is the widely understood component technology interchange protocol. At the next level we have a plethora of messaging standards that allow particularly looser coupled interoperability, with XML being a common thread amongst them. And at the top layer we have semantic or standards related to the meaning of business concepts, which provide the ability to speak the same language. The Microsoft BizTalk initiative is an extremely important driver of this change, bringing market-led sanity to a Tower of Babel. Also very important for the package vendors is the work of the OAG (Open Applications Group), which has become widely accepted by package and platform vendors. Driven by the market forces referred to previously, there is now much improved opportunity to interchange information and involve transactions from multiple sources in common processes. Even more recently there have been important new initiatives that seem set to establish consensus on wide areas of business meaning that will have profound implications for the package vendors and drive them into greater collaboration and openness.[1] While this activity may take some time to work through, it is predictable that the application integration market will decline as the package vendors are forced to adopt semantic standards and the requirement for glue logic diminishes.

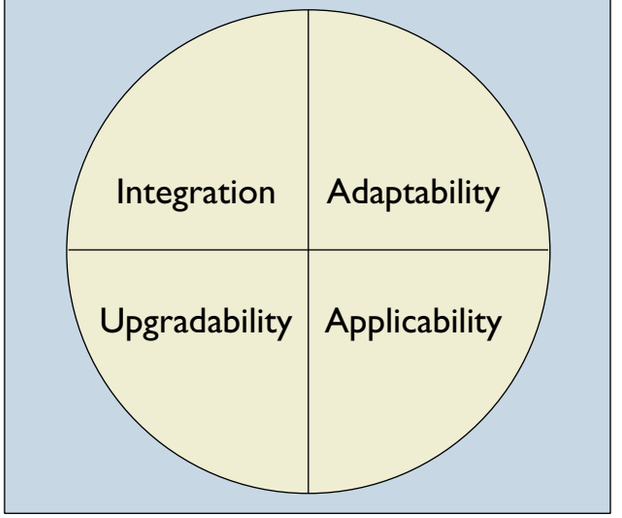
Selection Criteria

Most major organizations use at least one and sometimes several enterprise application packages. Sometimes purely for financial and human resource applications, often for core customer management and transactional purposes.

The way in which most organizations select and manage applications is on the basis of business features and functionality. Yet, today it is understood that technology and business processes change much faster than ever. It seems strange, therefore, given the high cost of implementation, that the selection criteria should not be very heavily weighted toward the ability of the application to serve the needs of the enterprise over a significant time period. Figure 3 identifies four incremental selection criteria areas organizations should use to choose enterprise applications.

Applicability. The enormous complexity of the application needs to be matched with the similarly high complexity of the business. Some organizations have been persuaded that it is beneficial to adopt "best-in-class" processes and systems, and adapt to the industry norm. The componentized application architecture will change this mindset because there will be more choice. The binary choice between "build or buy" becomes "build and buy and reuse." The published interface architectures create opportunities for both the prime vendor as well as third parties to create clone or extension components providing specific support for functionality that has either general or niche applicability—to establish business differentiation where it really matters most. The emergence of markets surrounding the core applications, where third-party developers create extension and even clone components provides a wealth of additional functionality. This is not a new

Figure 3. Incremental selection criteria.



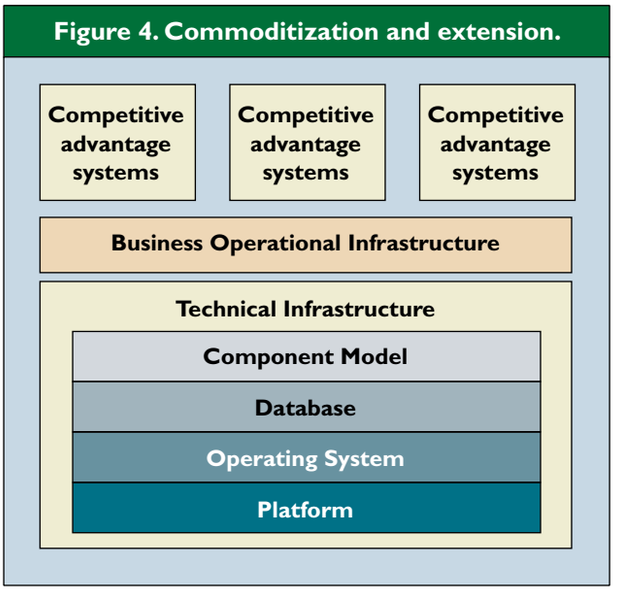
[1]Domain-specific initiatives such as the HR-XML Consortium, a nonprofit group dedicated to the development and promotion of standardized human-resources-related XML vocabularies, which is creating an XML framework designed to enable Web-based workforce management and recruiting services. Global initiatives such as the Electronic Business XML Initiative (ebXML)—see the January 2000 CBDi Forum report "Open market components"—a joint effort of the United Nation/CEFACT and OASIS. ebXML is an open, vendor-neutral initiative to establish a global technical and semantic framework that will enable XML to be used in a consistent manner for the exchange of electronic business data. See CBDi Forum reports including "Semantic interoperability," "Microsoft's interoperability strategy," "Framework for the financial sector", and others in the CBDi Forum Journal, *INTERACT* May 1999.

THERE IS A GENERAL AGREEMENT IN THE INDUSTRY THAT SOFTWARE COMPONENTS WILL BRING PROFOUND CHANGES IN THE WAY SOFTWARE IS DELIVERED.

model, but the existence of interface architectures makes it easy for third parties to create products and keep them up-to-date.

Integration. The ability to integrate applications with other packaged, built, and legacy applications now and in the future, is a key enabler of choice, and a key determinant of the cost and time required to implement and effect change. In the future, choosing the interface architecture together with compliance with de facto and de jure standards should be one of the most important selection criteria for any application. Increasingly, buyers will be able to exercise choice and to mix and match application components to better meet their requirements.

Figure 4. Commoditization and extension.



While this requirement is rapidly being addressed by application integration products and services including middleware, aids to wrapping and scavenging, as well as the so-called EAI products from vendors such as CrossWorlds, Neon and others, the interface architecture of the core package is a critical criteria that can make the integration task inherently easier or significantly more difficult.

Adaptability. Few organizations are going to remain static in the future, therefore adaptability is an essential characteristic of any application. The extent to which an application can change to meet new circumstances is a critical criterion. Types of change that we should look for include reconfiguration of components to work with other applications and or components and changes in platform and interoperability technology. For example, the extent to which the application system components can be implemented independently will vary greatly and needs careful examination.

Upgradability. A major advantage of componentized applications is the incremental release and upgrade process. This is beneficial in the initial implementation as well as ongoing enhancement. Unfortunately all suppliers and most existing application users will have to undergo considerable upgrade pain before they can achieve this situation. The realities of upgrading are also not as simple as might be supposed. For example, SAP recommends that even if only one application system component (sic) is being upgraded, a complete integration test should be undertaken because of the high levels of interdependence between the components.

Application Markets

The huge increase in quality of the componentized applications will also allow the vendors to deliver the functionality in a quick and economical manner, which would have previously been impossible. There is a high probability that the increasing dominance of application providers in delivering basic business support will lead to a commoditization of the business objects that are most commonly used. Figure 4 shows a new perspective on the conventional layered platform architecture diagram. Already we see component models and technical infrastructure being subsumed into the operating system and commodity layers. We may expect that over the next few years, the basic business support will also become commoditized. This will be reinforced by the move to application hosting and the huge size of the total market opportu-

nity makes this shift almost inevitable.

Competitive pressures to acquire market share at all levels will drive the price down, leading to similar effects we have seen in the technology layers. The improved application architectures are also enabling rapid entry into new markets, both vertical and horizontal. Customer management, sales force automation, and call center support all represent huge, untapped markets. Even insurance, banking, and financial services, the last bastions of custom development, are under development. Now that the Y2K crisis is past and many organizations have installed basic ERP packages the resulting installed base represents a formidable share of the general business transaction market, which provides a ready made customer base for organic growth for the package providers. However the basic applications, enhanced by interface architectures available in the installed customer base, make extension and integration with new business areas a significantly easier task than implementing major new applications from scratch. The functionality implemented in basic ERP packages may be considered in many cases as core business transactional support or the business operational infrastructure.

Where differentiation is critical is in how this operational infrastructure is exploited in for example e-business applications. In this and other areas of extension, the availability of interface architectures, open interface standards and application integration techniques, products and services will make the extension market place highly competitive. For example the interface architectures will facilitate the acquisition of standard functionality for say auction management, shopping cart management, shop windows, credit authorization and many more, but the package providers will need to compete with specialists in these areas. So while the package providers will and do provide extension and competitive edge capabilities, this area is likely to see significant acquisition of components from multiple sources which provide the consumer organization with added choice and differentiation of the business solution.

While it is clear the market share and dominance of the enterprise application providers is not about to decline (perhaps because of the level of "lock-in" that the average package exerts on its user rather than levels of customer satisfaction), the package vendors will come under considerable and increasing pressure as they are forced to open up their products, and open market components together with custom development become increasingly popular alternatives to customizing and extending base application packages. In fact the wave of e-business driven activity will be a stimulus for differentiated solutions.

Future Trends

There is general agreement in the industry that software components will bring profound changes in the way that software is delivered. The application package vendors have started to respond to this challenge and those that are successful will in time become true component providers themselves, having rearchitected their products into sets or kits of components. The service layers that many of the vendors have introduced to date will rapidly become inadequate, as componentized products exhibit vastly superior adaptability characteristics demanded by fast-moving e-business and e-commerce environments.

The successful vendors will provide their customers with a flexible and economic operational infrastructure, easily integrated with open market components, that allows their customers to exercise considerable choice in procurement to create customized solutions from readily and widely available building blocks.

Those vendors that succeed in this will continue to thrive post Y2K. Some vendors will fail to make this transition. Tomorrow's customers will demand the ability to buy, reuse, and build their competitive edge solutions to fit their needs, because they know this is a key business differentiation. Componentization will become a key business productivity action for suppliers and consumers in the application market. C

References

1. Sprott, D. and Wilkes, L. *Component Based Development.* Butler Group Report, 1998.
2. Szyperski, C. *Component Software: Beyond Object-Oriented Programming.* Addison-Wesley, 1998.
3. Wilkes, L. and Sprott, D. *Application Integration Management Guide.* CBDi Forum, 1999.

David Sprott (david.sprott@cbdiforum.com) is Principal Analyst and Co-chair of The CBDi Forum (The Forum for Component Based Development and Integration); cbdiforum.com is an information resource company with a mission to assist IT and business managers to better understand the realities of advanced application delivery based on practical end user and industry expert experience. Business and IT managers may join the Forum free of charge at www.cbdiforum.com