System integration

Science, Computer Science

> The term integration is inserted in technical papers, e-mail messages, correspondence, proposals, and even causal conversations. After many years of project work, and many misunderstandings and failed meetings and workshops, it can only be stated that the word has multiple and misunderstood meanings. For technical papers (research and trade), the term must be provided with context, or it is impossible to have a meaningful conversation. Next, multiple alternative definitions (that are valid in the literature for the appropriate context) are presented and explained in some detail.

Research limitations/implications - The paper is not exhaustive, since new definitions of integration may exist or may emerge. Originality/value - The main contribution of the paper is that it yields clarity on a key term that is frequently used in information systems research. The paper is useful to any researchers or practitioners who are focused on enterprise system implementation. Keywords Integration, Interface management, Applications, Information systems, Research Paper type General review Introduction and importance Integration is a common term in the enterprise systems literature.

Seldom does a meeting occur when the word is not used multiple times and often within quite technical contexts. Unfortunately, our experience is that individuals often have a different understanding of the meaning of the word. Loosely speaking, there is a general consensus that integration concerns making applications work together that were never intended to work together by passing information through some form of interface. This is

certainly part of the context, but this paper argues that there is more to be said.

Since the earliest days of computing, the term " integration" has been seed in both the trade andacademicliterature to describe a process, a condition, a system, and an end-state. Given that these competing labels have very different meanings, their indiscriminate usage is often obscure and invites confusion. For example, a sloppy conflation of process and condition encourages circular definitions that possess little explanatory power. Consider the followingadvertisement(Figure 1) from the Oracle Corporation and the corresponding quote from the Oracle CEO, Larry Ellison.

Figure 1 is clearly an appeal for a type of integration that we call " Big l," having all relevant data aligned with a ingle data model and stored only once. The implication is that you can place all of your data for the set of business processes listed in the middle column of Figure 1 inside of the Oracle E-Business Suite and significantly reduce total cost of ownership (TCO). In fact, the advertisement claims that Oracle saved over $1 billion USED per year by implementing Big l.

And also, there are the problems with complexity and managing scope integrity across multiple data sources (Gulled and Summer, 2004). Consider Figure 2 from an unnamed company. Figure 2 shows a situation that is described in the literature as " systems integration;" . E. The interfacing of systems together so they can pass information across a complextechnologylandscape. We call this type of integration a form of "

Little I," and we note that this form of Little I (point-to-point interfaces) is an expensive proposition.

Data must be constantly harmonize and cleansed across multiple data sources, and any changes to one system can lead to complex and costly re-testing or even re-design and coding of interfaces. Clearly, we have presented two extremes, and by and large both have been rejected by large organizations world wide. Most organizations do not want to include all of their data in one application (e. G. Oracle, SAP, Microsoft, etc.) for a number of different reasons, but at the same time, no one wants the problems that are associated with implementations like that shown in Figure 2.

There are other options. In fact there are many options, and that is the point of this paper. All of the options (including the two above) are called integration. So what is integration? As one might guess, it depends on the context, and the usage must be qualified. Big I may not achievable, and it may not even be appropriate. If Little I is appropriate, what type of Little I is appropriate, given the situation and the state of 7 Figure 2. Interfacing systems components to define an enterprise solution emerging technologies?

This paper addresses those questions, and it also categorizes the most used forms of Little I in the context of enterprise system implementation. This categorization and associated discussion is essential, or it is impossible to have a meaningful discourse about application integration. Integration - Big I To establish a baseline, the following definition is proposed for integration. Integration (Big I) - integration implies that all relevant data for a particular

bounded and closed set of business processes is processed in the same software application.

Updates in one application module or component are reflected throughout the business process logic, with no complex external interfacing. Data are stored once, and it is instantaneously shared by all business processes that are enabled by the software application. This is a rather comprehensive and restrictive definition that revivesmemoriesof first generation enterprise resource planning (ERP). The business process implications of Big I are discussed in some detail by Gulled and Summer (2003).

To preserve clarity throughout this paper, the above definition will always be referred to as " Big l. " Big I is definitely the goal of management, especially for mundane business processes. This implies " one source of truth" for those business processes that are enabled by core ERP solutions. The concept is simple: if all data are stored once and shared, then integrity issues are less likely to occur. The TCO is significantly less, since interfaces across application components are not required. Furthermore, complexity is significantly reduced. MEDS 8 Figure 3 shows how Big I relates to Little I for a simple example related to US Army Logistics. In this example, Army Logistics processes are scoped with the SAP solution as Big l; I. E. There is no interfacing across the SAP components. However, some of the logistics business processes flow outside of the Army. In this case, we indicate the transportation processes that are part of the end-to-end logistics business processes, but they fall outside of the Army, and they are managed by the US Transportation Command (TRANSOM).

The systems that support this segment of the end-to-end process are not SAP, and they are not even owned by the army. This is a classical composite application[3] and some form of Little I is must be implemented in order to preserve the integrity of the business process logic[4]. Figure 3, even though a simple picture, shows much about integration. First, it suggests that large and complex organizations are unlikely to place all of their business processes in a single application.

While assertions of Figure 1 are accurate, there are at least two reasons why single instance ERP will not occur in most firms: (1) he internet opened more options for Little I; and (2) thecultureand control of the internal and external system integration communities will not allow such consolidation. Like it or not, given the current state of technology, we are going to have to live with is a mixture of Big I and Little I, at least as long as the current trends continue.

The reality of this situation is reinforced by the fact that the larger software providers are " opening" their products and making them more flexible for mix and match Figure 3. An example of Big I and Little I in the same enterprise opportunities with Little I. This is evidenced by such products as the Oracle Data Hubs and SAP Interweave technologies. While it is true, Just as Figure 1 shows, that the TCO could be reduced by moving to Big l, most organizations do not have the flexibility nor the desire to do that. However, this does not mean that Big I is dead.

There will always be pockets of Big l; connected by Little I, to other pockets of Big l. This is not a technical assertion, but is directly related to common

sense. For example, one would never " rip" a product like SAP core ERP apart and then interface it back together again. This is self inflicted main, and it can be avoided by Just implementing the product the way it was intended to be implemented[5]. Preserve the integrity of the product by implementing Big I whenever possible, and use Little I to include those components that cannot be included in the integration domain.

One would never dream of separating financial from materials in an SAP implementation, and then interface it back together again. Or even worse, it makes even less sense to stand up independent SAP solutions in different divisions of a company, operating as afamilyor fiefdom, with the absence of an enterprise orientation. We will revisit implementation options later, but before doing that, we must further explore the options for Little I. The choice of a particular little I technology has significant implications for the types of mix and match options that are available for consideration.

Integration (Little I) As previously mentioned, all forms of Little I are some form of interfacing, even though they are loosely called " system integration. " Much has been written on the subject, so we only focus on those types of Little I that are most relevant for the implementation of enterprise systems: point-to-point integration; database-to-database integration; data warehouse integration; enterprise application integration (EAI); application server integration; and business-to-business (BIB) integration.

Point-to-point integration This is the most expensive form of integration. Point-to-point integration is the pair wise development of interfaces among systems. The data model of the target and source system are known, and

someone (e. G. A system integrator) develops the code for passing information back and forth. Sometimes accelerator products are used, a good example being the IBM Miseries of middleware products that are now included as a part of Webster. Miseries does require writing code at both the source and target system.

The approach to point-to-point integration is well known, most frequently involving changing both applications to use a middleware layer, by rewriting the transaction handling code to communicate across the two applications. The traditional model of interaction is through remote function calls. The largest problem with point-to-point integration is shown in Figure 4, a situation that Schafer (2002) attributes to a customer situation. 9 10 Figure 4. Example of point-to-point integration As the number of interfaced components is increased, the number of interfaces to be maintained increases dramatically.

The TCO likewise increases. As a real example consider the financial interfaces to a Navy SAP solution that is shown in Figure 5[6]. Figure 5 is a good example of the previously mentioned case that can arise when financial are separated from materials or assets in an enterprise solution and then must be interfaced back to the ERP product, violating the integrity of the solution. While Figure 5 is reality and could not be easily avoided, the SAP product was never intended to be implemented in this way. The integrity of the product is violated by destroying the Big I that is engineered into the product.

For all of the reasons previously mentioned, point-to-point integration should be avoided and only be used when there are no other options. Database-to-database integration This form of Little I, requires the sharing of information at the database level; hence, providing interoperable applications. The basic replication solution leverages features built into many databases to move information between databases as long as they maintain the same schema information on all sources and targets. There are companies that provide middleware to accelerate this process.

Database and replication software are provided by companies such as Pervasive Integration Architect and Denominator's Constellate Hub that permit moving information among many different database products with different schema. Figure 6 shows the conceptual layout for this form of Little I. While this integration procedure may work well for database applications, it does not work so well for enterprise applications. Most enterprise applications have 11 Figure 5. From defense financial and accounting services to the US Navy Pilot SAP implementations Figure 6.

Conceptual layout for database-to-database 12 multi-tiered architectures, where even though the applications reside at a separate tier, the business process logic is " bound" to the master data. So, if one simply passes information at the database level, it is easy to create data integrity problems. Enterprise software vendors typically publish application program interfaces (Apish) that allow interfacing at the application level, and it is best to use these Apish. If you update the database without using the Apish, then

you are violating the Big I that is engineered into the product, and integrity problems are a likely result.

See that Anonymous (1999) article in enterprise development where some of these difficulties are discussed within the context of interfacing with SAP's R/3 product. For enterprise implementations, this form of Little I should be avoided. Data warehouse integration This form of Little I is similar to database-to-database integration, but instead of replicating data across various databases, a single Martial database" is used to map the data from any number of physical databases, which can be various brands, models, or schema.

In other words, a new data warehouse is created, and information is aggregated from a number of sources, where it may be analyzed or used for report generation. The effectiveness of this approach depends on the sophistication of the tools that are used and the quality of the data that is pulled from the various sources. Once the data are aggregated, reporting is straight forward; however, if business process logic must be applied to the aggregated data, then that logic must be created at the data warehouse level.

The basic layout for data warehouse integration is shown in Figure 7. Figure 7. Conceptual view of data arouses integration If the integration is at the database level, the same problems associated with database-to-database integration that were mentioned above still apply. If the integration is at the application level, then data warehouse integration is similar to point-to-point

integration, and the problems with that approach also apply. This form of integration is quite popular, even though it is expensive to maintain.

The reason that data warehouse integration is popular, is that it allows all parties involved to maintain their individual stove-piped environments while sharing selective data in a auteurenvironment. In short, one is trading Big I for autonomy. An example of a large data warehouse integration effort in the US Army is shown in Figure 8. The logistics integrated database (LIDS) contains aggregates information from many stand-alone systems, with the objective of providing enterprise-level analytics. As the fugue indicates, the input data are aggregated from many sources, and output data are pushed to many sources.

Constant cleansing and harmonistic is required in order to avoid integrity problems. Many enterprise solutions, like those from SAP and Oracle, use data warehouse lotions for reporting and enterprise analytics. However, this static view of enterprise data are not the same as Big l. Even if the concept is extended to include a federated query capability with the data warehouse being a virtual repository of metadata, this is still no substitute for Big l. However, the big problem, as previously mentioned, is the maintaining of business process logic at the data warehouse level.

While this option preserves organizational autonomy, it is indeed costly. The data that are pushed into the warehouse must be constantly monitored for quality, and NY changes in any one of the target or source systems create significant testing and/ or additional coding problems. 13 Figure 8. A conceptual view of the LIDS 14 Figure 9. Hub and spoke architecture for

enterprise application integration Enterprise application integration EAI is the sharing of data and business process logic across hetero/homogeneous instances through message-oriented-middleware (MOM). EAI may be managed by packaged vendors (e. . SAP and Oracle) or through solutions provided by third party vendors (e. G. MM, Webmasters, etc.). EAI is sometimes called application-centric interfacing. EAI is used to connect multiple systems at the application or database levels, using a form of middleware that is sometimes called a broker. The middleware moves information in and out of multiple systems, using pre-engineered " connectors. " The connectors are a source of competitive advantage for EAI software providers, because if a connector already exists for the target and source application, the cost of interface development can be reduced.

The problems associated with point-to-point integration are reduced by adopting a hub and spoke model for sharing information. The EAI Middleware allows one to rite a single interface between each application and the middleware, instead of individually connecting each application to every other application. An example of a hub and spoke architecture is shown in Figure 9. Once the information is extracted, it is sent to a central server using some sort of messaging system, where the information is processed and routed to the target system.

If there is a gap in required business process logic, the logic can be created on the central server for execution. In theory, any-to-any document swap is possible, considering the business process logic in the source and target systems. Using " connectors," the EAI software processes messages from

packaged applications, databases, and custom applications using a queuing engine. When an event occurs (e. G. A transaction in an ERP package or a database table update), a message is published to the queue about the event.

Subscribers to queue access the event envelope, analyze the content, and if it is intended for processing in the target system, the envelope contains everything necessary for recreating the event in the target system. The queuing engine ensures that all events are processed in the correct sequence, ensuring transactional integrity. Many companies provide pre-packaged EAI solutions, and the market is extremely competitive. The hub and spoke model using connectors has been operational for many years, and the products have reached a mature level.

However, we note that EAI is still interfacing, and while this is a significant improvement over point-to-point integration, EAI can be costly to implement and costly to maintain. The main benefits flow from being able to use " partially configured" connectors, while leverage industry partnerships which yield certified interfaces. Tremendous consolidation has occurred n recent years in companies that provide EAI solutions as the larger software providers have moved in to provide EAI solutions that interact with their Big I products.

For example, SAP now supports EAI as part of its Interweave[7] solution, where previously SAP had used third party providers like IBM and Webmasters to provide EAI capabilities. It is also important to note that EAI is typically used inside the enterprise, as opposed to across the enterprise. For

this reason EAI is sometimes called application-centric interfacing. The objective is to interfaces processes and share data within the enterprise. The inter-enterprise model falls under a class of solutions that are called Business-to-Business commerce, and this form of interfacing will be discussed in a later section.

Application server integration This is the most sophisticated form of Little I that is discussed in this paper. Think of application server integration as the creation of a single, centralized application (logical or physical) that can provide a common set of services to any number of other remote applications. These " services" are common business objects that are shared across enterprise applications. The sharing and reuse of services is the goal of distributed objects and applications servers.

Application server integration enables the enterprise by sharing services across the enterprise. The concept of application server integration is shown in Figure 10. Modern systems invoke shared objects to share business logic and interact with resources (such as databases, ERP systems, or queues). In modern ERP systems these shared objects may be more highly aggregated as " wrapped" transactions. For example, when configuring the SAP solution, one aligns transactions with process steps. A process step could be associated with one or more transactions.

If the transactions associated with a process step are bundled together and " wrapped" as a web service, then they may be shared across other SAP and non-SAP components. SAP calls this aggregated object an " Enterprise Service," and it is the basis of SAP's Enterprise Services Architecture (SAP

GAG, 2004). Application integration occurs through the sharing of business logic, as well as through the back-end integration of many different applications and resources. The application server " binds" the data from a relational or relational-object database to he common shared objects.

The main advantage of application server integration is that 15 16 Figure 10. Application server integration concept the interfaced applications or components are tightly coupled to each other by sharing methods. By our assessment, application server integration is Little I, but given the limits of current technology it is the best approximation that we can provide to Big I. This is because the data integrity checks and business logic bound to the objects are always shared, and therefore, never circumvented. The SAP example is not unique. Most of the major software vendors have a similar tragedy.

For example, Figure 11 shows the Oracle strategy for application server integration. The key component of Figure 11 for our discussion is in the right-center of the figure. The Oracle Application Server manages the shared objects and during runtime " Top Link manages persistence between Java objects and database tables. " At the conceptual level the integration approaches pursued by Oracle and SAP are similar. The widely accepted disadvantage of using this application server integration is that significant changes may have to be made to all source and target applications to