Data integration solutions
White paper

WebSphere. software

The definition of a business service for customer data integration.

By David Corrigan, product marketing manager, IBM WebSphere Customer Center

March 2006

Contents

Introduction

With the rapid growth in interest in customer data integration (CDI) applications and service oriented architectures (SOAs), many vendors have rushed to define their solutions as customer data integration (CDI), service-oriented applications. The result has been confusion: vendors who have different applications have vastly different definitions of a service, yet they all claim to have a service-oriented CDI solution. Customers struggle to understand the subtle yet significant difference between CDI application vendors. One of the key points of difference is each vendor's definition of business service. This is a critical point for customers to understand, because the primary functionality and interface into a CDI application are its business services. This paper will define different vendor approaches to developing business services and the relative cost implications of each approach for the customer.

Definition of a business service

Gartner Group offers the most complete view of a business service and service-oriented business applications:

"*What makes the current stage of this evolution noteworthy is threefold. First, the standards for defining and accessing services (key to ease of assembly) are becoming widely deployed. These are the Web services standards such as Simple Object Access Protocol (SOAP) and Web Services Description Language (WSDL). Second is the ability to combine these services at the time of execution, not before – so a designer is not challenged to conceive of all possible combinations of activities in advance. Third, there is a growing understanding of the right scale and content of a service – the happy mean between too fine-grained and too complex vs. too coarse-grained and too inflexible.*"*

The key definition points for a business service are that it must:

- *Represent a function*
- *Be offered at a granular, but functional, level*
- *Be offered at a larger-grain level as an aggregate of fine-grain services*
- *Permit integration through multiple technologies directly to the service (Web services) for both loose and tight coupling of the service, as opposed to forcing all integration through message queuing*
- *Be able to be combined with other services into a composite transaction*
- *Permit customization of business logic within the service*

CDI business services represent customer data management functions. The primary responsibility of those functions is to maintain master customer data in the customer database. A service-oriented CDI application contains the following elements shown in the IBM WebSphere Customer Center example in Figure 1.

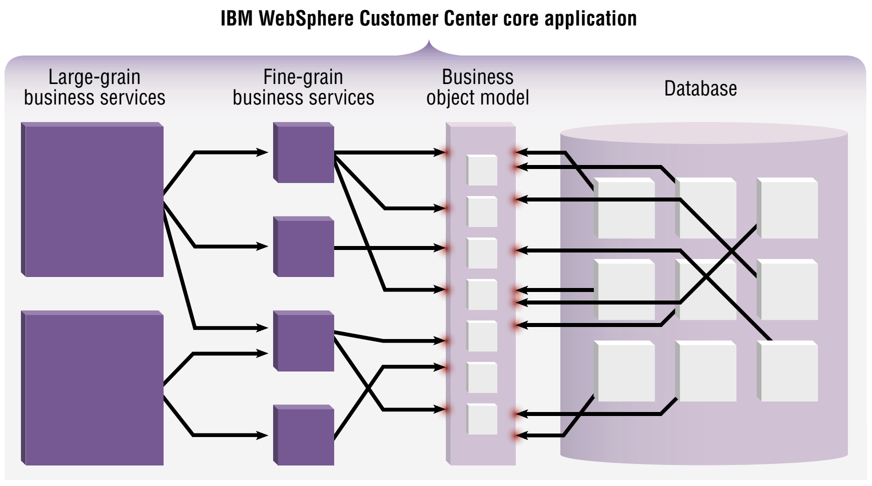
IBM WebSphere Customer Center core application
Large-grain business services
Fine-grain business services
Business object model
Database

Figure 1. Elements of a service-oriented CDI application

Large-grain business services are defined as customer data processes. Large-grain services comprise many fine-grain services and represent a significant function performed by the organization. Examples include adding a new customer or adding a contract and multiple new customers.

Fine-grain business services are defined as more atomic customer data processes or components of a larger unit of work. Fine-grain services encompass one or more objects (and therefore data tables in the database). Fine-grain services represent a level of functional abstraction above table-based create, read, update and delete functions. Examples include adding an address for a party or updating a party identifier.

An *object model* is defined as atomic functions. It is a specification of the objects intrinsic to the given application, including a description of the object characteristics (attributes), the object's functions, and a description of the static and dynamic relationships that exist between objects. At some point, these objects need to be persisted and, if they are persisted in a relational database, an object-relation mapping must occur. For data-centric applications (like CDI hubs), objects typically represent data functions and are a logical representation of the physical database for master customer data. Functions include creating, reading, updating and deleting data. Granular objects are not truly functional because they represent the database structure, not a process or functional usage of customer data. Objects are not business services.

A *database* is defined as the database in which customer master data is stored in multiple tables.

Different vendors' definitions of business service

As mentioned in the Gartner definition previously, different vendors have different approaches to a business service. Figure 2 illustrates the three different approaches to business services with respect to CDI applications.

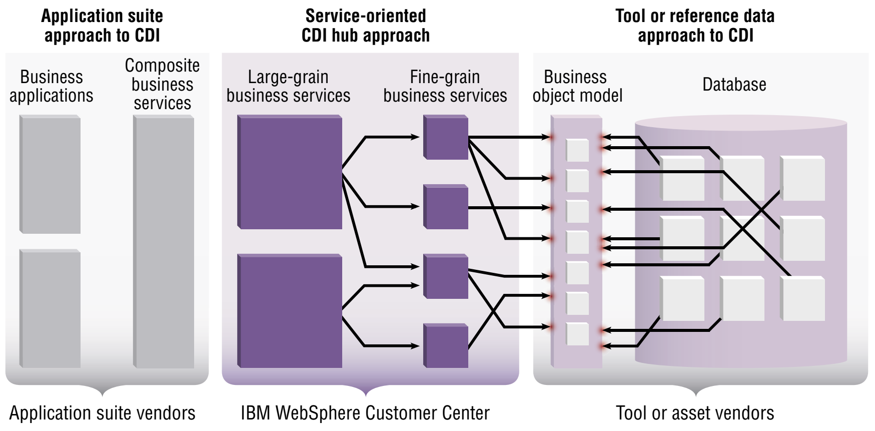
Application suite approach to CDI
Service-oriented CDI hub approach
Tool or reference data approach to CDI
Business applications
Composite business services
Large-grain business services
Fine-grain business services
Business object model
Database
Application suite vendors
IBM WebSphere Customer Center
Tool or asset vendors

Figure 2. The three approaches to business services

Service-oriented application approach

Overview

Service-oriented CDI applications' central design point is the business service that will be used by multiple business applications. These applications were designed from the onset as service-oriented applications. All functionality was designed as a callable service. The result is that different functionality is made available when the original design point is building services as compared to service-enabling of an existing application.

Key points

Service-oriented CDI applications have important differences from either application-suite or tool-based approaches, including:

- *The middle-ground definition of a service: SOA CDI hubs deliver both large- and fine-grain services. Those services offer varying levels of functionality that represent microflows, which are business functions that are more atomic than a business process that would be modeled in an enterprise application integration (EAI) workflow or a business process modeling (BPM) tool. Microflows are larger than granular data-table-level data functions (adding and updating data within a database table). In summary, microflows are atomic to a functional level for customer data management.*

- *Accessibility: SOA CDI hubs offer multiple access methods to directly call the business services. These include Web services, XML, EAI publish-subscribe interfaces and object-level interfaces. The key difference between SOA CDI hubs and other approaches is the fact that the services can be directly called by an application or integration hub—previously accessible only through an unnecessary, middleman EAI or work-flow tool. This offers greater flexibility in deployment and integration of the CDI hub shown in Figure 3.*

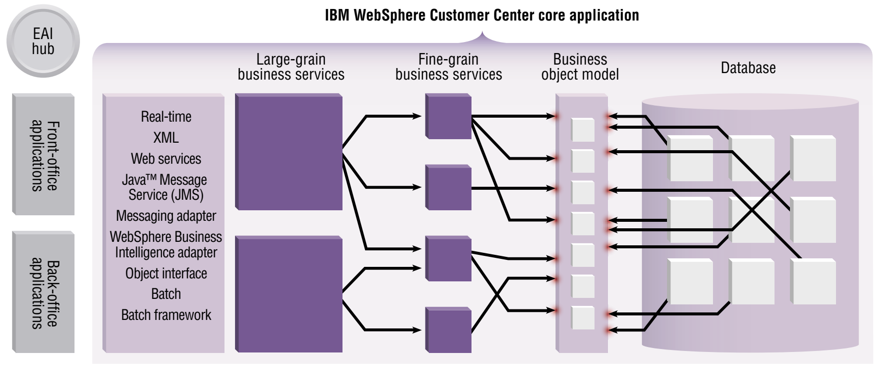
IBM WebSphere Customer Center core application
EAI hub
Front-office applications
Back-office applications
Large-grain business services
Fine-grain business services
Business object model
Database
Real-time
XML
Web services
Java™ Message Service (JMS)
Messaging adapter
WebSphere Business Intelligence adapter
Object interface
Batch
Batch framework

Figure 3. Access methods of a service-oriented CDI application

- *Flexibility: Ability to build composite transactions – Gartner identifies the ability to build composite transactions from existing services as a key requirement of a service-oriented application. An SOA customer hub must offer the ability to build composite transactions at multiple levels: both within the customer hub application as a single unit-of-work service, and within outside applications such as BPM tools and EAI applications. Customers will often need to build composite transactions that meet their exact business requirements, and the CDI hub must be flexible enough to support those customizations without significant and costly rework. Only CDI hubs that expose granular transactions as business services facilitate the building of composite services.*

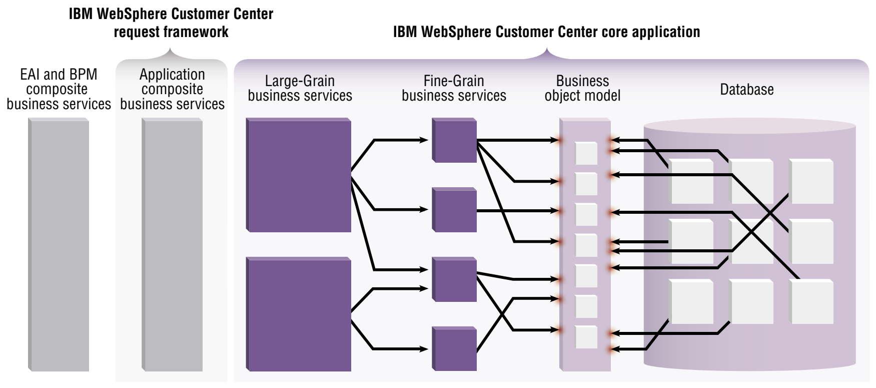
IBM WebSphere Customer Center request framework
IBM WebSphere Customer Center core application
EAI and BPM composite business services
Application composite business services
Large-Grain business services
Fine-Grain business services
Business object model
Database

Figure 4. Building composite transactions within a service-oriented CDI application

- *Flexibility: Ability to extend the product, integrate business logic components and external applications – a service-oriented CDI hub is designed around integration. The business service is the central concept of the application, and the business service is the central point of integration in a transactional context. Application components of the CDI application are integrated with the business services, allowing transactions to trigger business-logic functions. In addition, the services* *should all contain pre- and post-exit points to integrate external applications, including data-quality applications (data standardization and hygiene), external data sources (external identifiers and customer data providers), and other sources of data within the organization (enterprise information integration tools). The service must be designed to accommodate integration with a variety of other components, services and applications as a means of integrating components within the business services. In addition, the service and database must support and manage extensions and customizations to allow customers to add new functions and data to the core product, but still allow customers to upgrade the core product. This design helps greatly reduce upgrade costs for customers.*

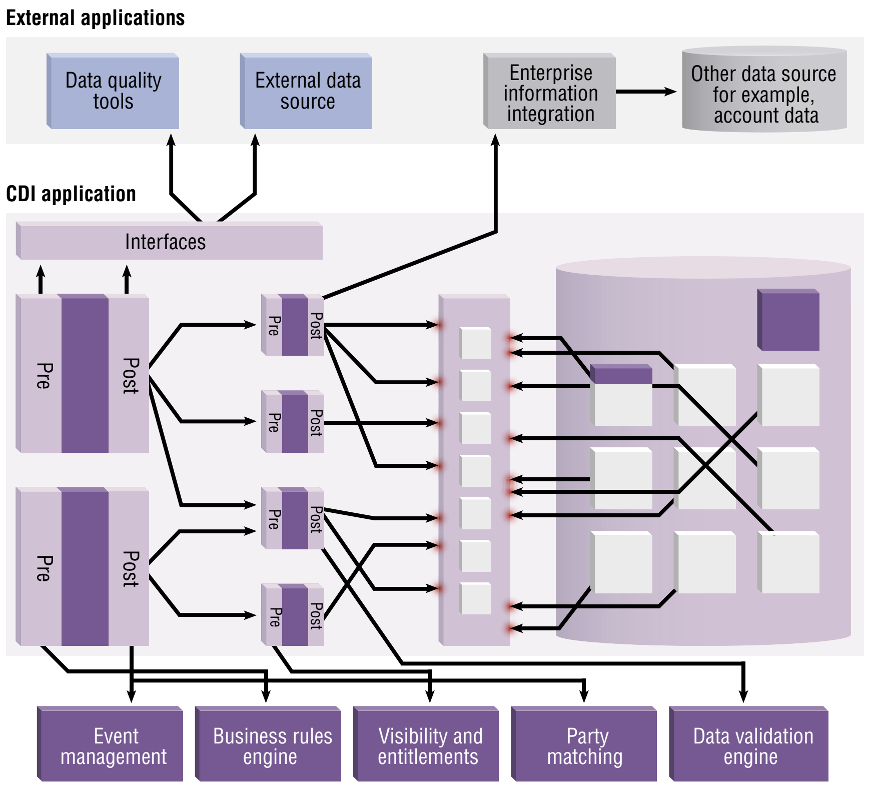

Figure 5. Pre- and post-exit points for integration with external applications

- *Extensibility and compatibility with existing versions: An SOA CDI hub must allow the core product to be extended as required. The hub must allow for client-specific extensions and additions. When architected correctly, the application manages those extensions separately from the core product code. This allows clients to upgrade the core application while preserving all extensions. In addition, the services must be compatible with existing versions to support prior service-request structures and respond appropriately. This provides two key benefits for customers. First, the cost of upgrades is greatly reduced. Second, customers benefit from the ongoing product road map and future release of the core product.*

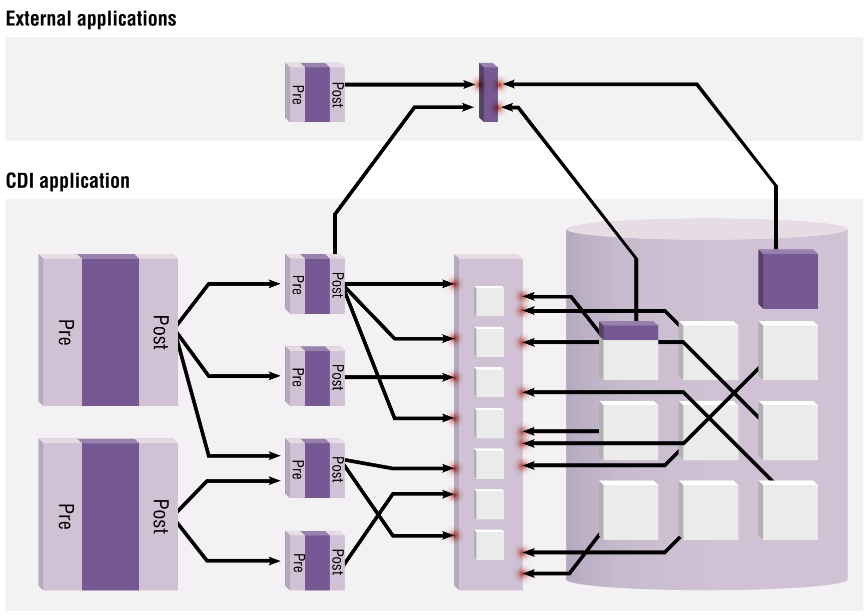

Figure 6. Extensibility of a service-oriented CDI application

- *Transaction management: An SOA CDI hub must be designed as a transactional application. In other words, it is designed as a high-volume transaction processor for customer data transactions. The CDI hub must provide transaction-management* *functionality to control transaction context, transaction rollback, macro- and micro-level logging (large- and fine-grain services), macro- and micro-level performance monitoring, and transaction logging and audit-history maintenance. This functionality is required in order to implement a CDI hub as a system of record for customer data.*

Benefits of the SOA application approach to CDI

Numerous architectural and cost benefits are associated with the SOA approach to CDI, including:

- *Reduced cost: The SOA approach allows vendors to more easily customize the business services to meet their processes—to build extensions, configure new composite business services and integrate other best-of-breed vendor components within the business service.*

CDI application

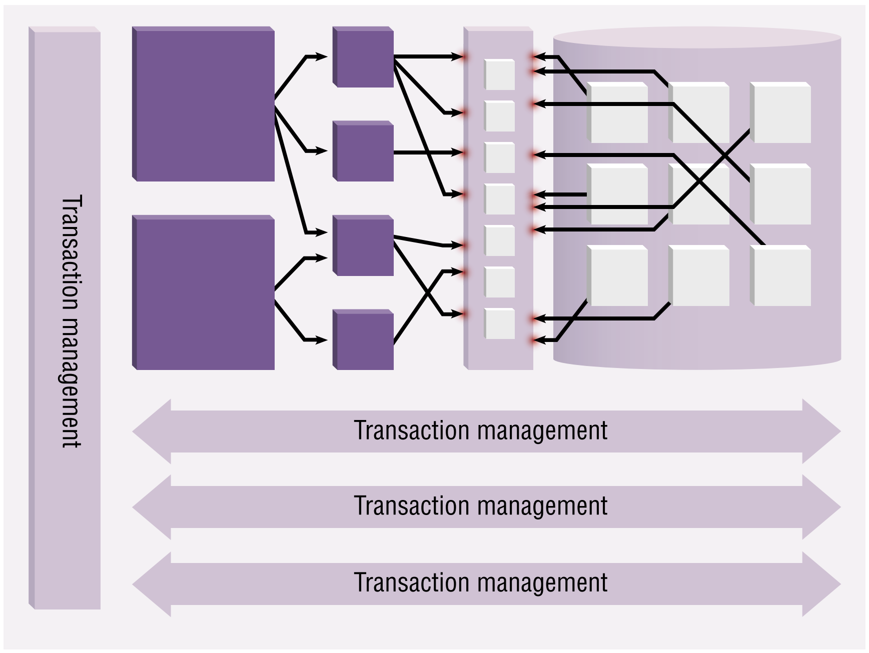

Figure 7. Managing transactions within a service-oriented CDI application

- *Future upgrade path: SOA has the ability to take future upgrades of the product while preserving extensions and customizations. This allows customers to easily upgrade the core product and therefore helps reduce the cost to upgrade the product.*

- *Flexibility: The SOA approach allows for greater flexibility in implementation, and helps organizations to make future changes more easily and cost-effectively.*

Application-suite approach to CDI

Some customer relationship management (CRM) and enterprise resource planning (ERP) application-suite vendors have reused portions of their application suite as a CDI hub. As noted by Gartner previously, many of those application vendors have really exposed application programming interfaces (APIs) and have named them a service. Application-suite vendors have significant existing CRM and ERP applications that heavily influence their CDI applications. As those vendors migrate to a service-oriented design, their initial concept of a service is really an API connecting to their application suite. Much of the business logic remains tied to their front-end applications and the presentation tier, and it is not exposed through business services. The application-suite approach to CDI is shown in Figure 8.

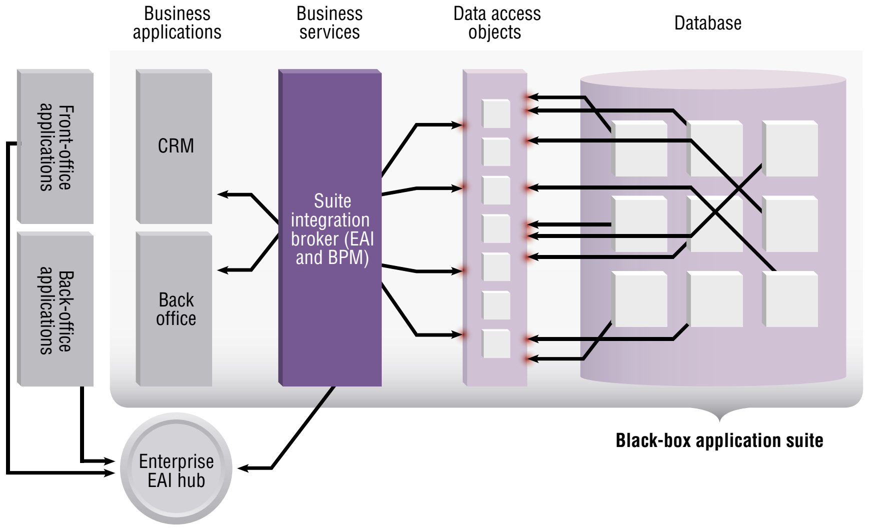

Figure 8. Application-suite approach to CDI

Definition of a service

- *Application-suite vendors' definition of a service is too large to represent an atomic functional unit of work. These services represent an API into their application suite, not a discrete unit of work that is required by multiple applications.*

- *The drawback of this approach is that users cannot interact with the customer hub normally: They have to alter their business processes to meet the vendor's large and rigid API service. The other significant drawback is performance: The architecture of these solutions introduces unnecessary layers (enterprise-message queues speaking to CDI-hub message queues). The performance of application-suite CDI hubs is worse than that of SOA CDI hubs, forcing the customer to add expenses by buying more infrastructure (more CPUs and servers, for example) to achieve the desired performance level. The key point is that the purpose of an SOA is to eliminate redundant functions and components, and the application-suite CDI approach introduces further redundancy by coupling EAI and BPM functions with the customer hub.*

Accessibility

- *The vendors' large-grained API services are offered through only the vendors' proprietary EAI, workflow and BPM tool. This forces all access to go through an internal application message queue.*

Flexibility: Ability to build composite transactions

- *Composite transactions must be built within the proprietary EAI tool. Because fewer granular services are exposed, it is difficult and sometimes impossible for customers to build the exact composite business service that they require.*

- *The drawback of this approach is that companies often have to adapt their business processes to accommodate the shortcomings of the CDI vendor's application suite. The ability to build composite transactions is limited because of the inaccessibility of finer-grain services. This introduces costly application customization work for the customer.*

Flexibility: Ability to integrate business-logic components and vendor tools

- *Business logic is typically allowed within defined parameters and must be customized using the application-suite vendor's proprietary tool set. Some business-logic components are not offered by application-suite vendors because of their application-suite focus. For example, the concept of visibility rules and entitlement management within a service does not exist because application-suite vendors enforce that functionality within the business applications. This example illustrates a common observation with application-suite CDI hubs: They often represent the lowest common denominator of data access APIs with very little business logic. This is by design, because their focus is to sell an application suite, not a stand-alone customer-management component.*

- *The drawback of this approach is that customers are often not able to integrate other vendor, best-of-breed components or other data sources easily. The integration method is often batch because the application-suite vendor's design point is to bring as much data and function within the suite as possible—not to call out to other applications. This introduces significant costs for customers, because they often have to recode vendor tool rules within the application suite's tool set, or to build complex business rules within the application to compensate for the suite's lack of functionality within its services.*

Flexibility: Customizing and extending the application and compatibility with existing versions

- *Application-suite vendors offer limited customization options. Typically, they have defined customization options (naming dummy variables) that are limited and rigid. Their suites do not typically have the ability to add new services and components easily.*

- *The drawback of this approach is that customers are limited in their ability to extend and customize the services (and therefore the application). Customers incur significant cost to add even slight modifications, as the application requires that customizations be done within proprietary tooling environments.*

Transaction management

- *Application-suite vendors offer some transaction-management capabilities using their proprietary EAI tools. Application-suite vendors offer few transaction-management capabilities. This is a result of their design point: They were originally designed as channel application suites, not as transaction-processing engines. Their integration point was designed with the purpose of bringing data into the application suite.*

- *The drawback of this approach is that customers lack confidence in the application-suite CDI hub to become a system of record. These applications lack the transactional context and management functions required to become a system of record.*

Application-suite CDI vendors—bottom line

Application-suite CDI hubs have a different design point from that of SOA CDI hubs. They were designed originally as integration points for the application suite, and later evolved to a customer hub positioning. These applications have a rigid design and do not permit fully flexible integration of vendor components and construction of composite services. The bottom line for customers is that application-suite CDI solutions have a far higher total cost of ownership (TCO) than SOA CDI hubs, and they force customers to make compromises because of the inflexible design of application-suite CID hubs.

Tool and asset approach to CDI

Tool- and asset-based vendors have a far different definition of business service. Their applications adopt a *model-driven design philosophy*. This means that the central design point is the data model, not a business service. The data model is meant to be customized, and granular create, retrieve, update and delete methods are generated for each table in the database. Tool and asset CDI hubs call those granular data functions business services. In reality, those services represent granular objects. They do not represent a customer data function, and are therefore too fine-grained to be usable. Typically, the tool or asset vendor's business services are something like the representation shown in Figure 9.

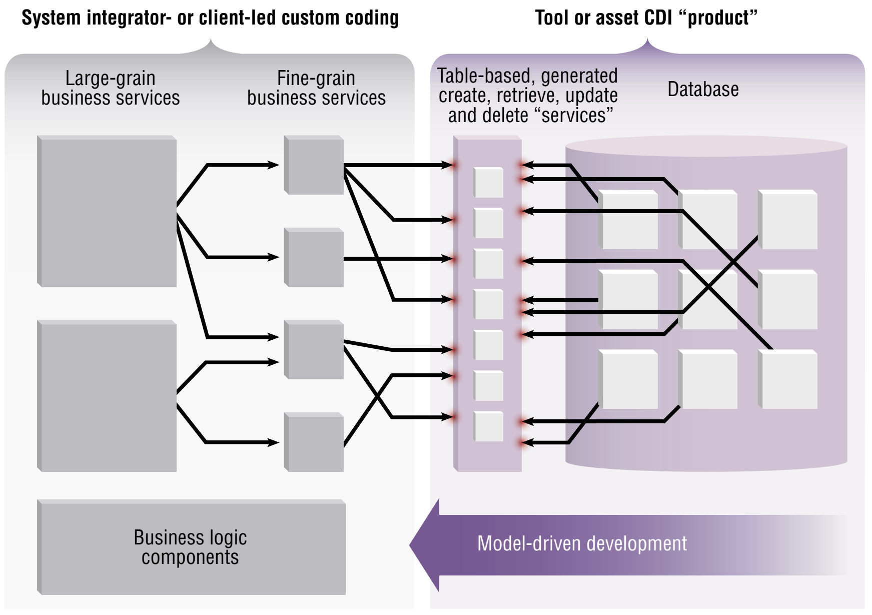
System integrator- or client-led custom coding
Tool or asset CDI "product"
Large-grain business services
Fine-grain business services
Table-based, generated create, retrieve, update and delete "services"
Database
Business logic components
Model-driven development

Figure 9. Tool and asset approach to CDI

Definition of a service

- *Overly fine grained – their business services offer create, retrieve, update and delete functions at a table level.*

- *The drawback of this approach is that customers have a significant amount of custom work to build functional business services on top of the product's create, retrieve, update and delete functions. This is a significant development effort and cost for the customer that is not part of the core product (therefore the customer owns the services and the support and maintenance costs for those services).*

Accessibility

- *The tool or asset vendor's services might have one or more interfaces. The key issue is that the underlying function is far too granular to be functional, and therefore applications are not likely to interface to extremely granular functionality.*

- *The drawback of this approach is that customers will have to rebuild interfaces to the custom services or functionality (or both) that they build on top of the tool, adding further cost to their implementation. Applications must navigate through the data model, making them more dependent on specifics of the physical implementation of the data model.*

Flexibility: Ability to build composite transactions

- *Tool and asset CDI vendors allow customers to build composite transactions but offer little tooling to do so. Customers have to build composite services and business logic using development tools.*

- *The drawback of this approach is that customers do not have tooling to help build composite services, which adds cost and time to the implementation process.*

Flexibility: Ability to integrate business-logic components and vendor tools

- *Tool and asset vendors do not contain separate components for isolated business-logic functions. Customers have to build custom business logic, which is usually done within the custom code of the service. It is often difficult to integrate with vendor tools directly because of the architecture platform of the core application (for example, COBOL).*

- *The drawback of this approach is that business logic is closely coupled with the service, limiting future flexibility. In addition, it is difficult to integrate vendor components, which further limits functionality. In addition, this adds significant cost to the custom build-out effort.*

Flexibility: Customizing and extending the application and compatibility with existing versions

- *Tool and asset vendors offer the ability to customize and extend the core data model and to generate create, retrieve, update and delete code from the model. Although this seems to be a flexible approach, in reality it is flexible from a data-centric point of view only. The tool or asset CDI hub does not contain flexibility within its services layer to allow for extensions, additions and customizations.*

- *The drawback of this approach is that customers have no ability to accommodate future change or extensions at the service level. The cost of change to the generated application or of building on top of that application is very high.*

Transaction management

- *Tool and asset CDI hubs lack a transaction context at a true business-service level. Their business services are too granular and, therefore, a customer would access multiple services within a real transaction.*

- *The drawback of this approach is that the customer must assume responsibility for transaction management outside the core application, and therefore assume the cost of custom building that functionality.*

Tool and asset CDI vendors—bottom line

Tool and asset CDI hub offerings have a very different design point from that of SOA CDI hubs. Tool and asset CDI offerings represent a base level of functionality at a database-centric level (create, retrieve, update and delete functions for each data table). Because tool and asset vendors do not offer functional services, customers incur significant costs to build out functionally atomic business services on top of the core asset. Customers also incur significant ongoing support and maintenance costs for their custom-built solutions.

Conclusion

Within the CDI market, there are widely different definitions of business services from different vendors. To meet the definition of true business services, the vendor's business services should:

- *Represent a function*
- *Be offered at a granular, but functional, level*
- *Be offered at a larger-grain level as an aggregate of fine-grain services*
- *Permit integration using multiple technologies directly to the service (Web services), not using message queuing*
- *Be able to be combined with other services as a composite transaction*
- *Permit customization of business logic within the service*

The only CDI vendor approach that meets the above criteria is the service-oriented CDI application approach. Both the application-suite approach and the tool or asset approach require significant customization work to be done by the customer, which dramatically raises the TCO of either of those solutions. Therefore, from the point of view of integrating within an SOA, only the service-oriented CDI hub application will offer the lowest TCO.

For more information

To learn more about IBM WebSphere Customer Center software, visit:

ibm.com/software/data/masterdata/launch.html

Gartner Group, Simon Hayward, Positions 2005: *Service-Oriented Architecture Adds Flexibility to Business Processes*

G507-0811-00