Microsoft® Business Solutions CRM

Microsoft® CRM Product Architecture

White Paper

Published: April 2003

Abstract

This white paper will discuss Microsoft CRM from both the customer's and the implementers'—partners, value-added resellers, and independent software vendors—points of view. The paper comprises a general overview of Microsoft CRM that includes deployment strategies and product design goals; a detailed discussion of the product architecture, security model, and enabling technology; and brief descriptions of reporting capabilities, internationalization support, and product customization.

Contents

Introduction

Microsoft® Business Solutions CRM is a.NET connected business application. It has been specifically designed to meet the needs of companies wishing to implement a CRM solution.

This white paper will discuss Microsoft CRM from both the customer's and the implementers'—partners, value-added resellers, and independent software vendors—points of view. The paper comprises a general overview of Microsoft CRM that includes deployment strategies and product design goals; a detailed discussion of the product architecture, security model, and enabling technology; and brief descriptions of reporting capabilities, internationalization support, and product customization.

General overview of Microsoft CRM

Microsoft CRM is not so much a technology as it is a business strategy used for helping companies acquire new customers and maintain current customers. Acquiring new customers costs four-to-five times as much as keeping and reselling to existing customers. Microsoft CRM enables companies to build and implement business processes that help retain existing customers.

Microsoft CRM is all about collaborations and interactions between vendors, supplies, and customers. The core goal of Microsoft CRM, as a technology, is to enable and track these interactions in order to learn more about customers. Each time a sales person interacts with a customer, the company learns something new about that customer. For instance, the customer might prefer to interact by electronic mail, but will accept broadcast fax messages. Using Microsoft CRM, learning how customers were acquired is not only possible, but quite easy. Microsoft CRM automatically tracks all interactions, regardless of how they were initiated, between your sales and service employees and your customers. Later each interaction can be recalled individually, or combined with others to create a complete history, commonly known as a 360-degree customer view. Simply put, this 360-degree view captures the entire set of interactions and collaborations between your company and your customer.

Deployment strategies

Microsoft CRM has been designed specifically to be quickly and easily deployed into companies wishing to manage sales and support interactions with their customers. The product can be installed on multiple physical servers so that it grows with your company. Microsoft CRM poses no real limitations on deployments and installations; however, some of the supporting technologies, such as Microsoft Exchange Server and Microsoft SQL Server™, might do so.

Microsoft CRM design goals

Microsoft CRM was designed with several basic design goals:

Provide low total cost of ownership (TCO). The product must be manageable, allow customization, be easy to deploy and upgrade, and be well-supported by both Microsoft and the independent software vendors community at large.

Provide low-cost product customization. Product customizations are very important in a CRM environment. No single, out-of-box solution can meet every customer's needs, so Microsoft CRM has been designed to make customizations very simple and affordable.

Scale up and out to hosted environments. While Microsoft CRM isn't sold as a hosted solution, the product was designed with this deployment model in mind. The hosted application model imposes very high demands on an application, and Microsoft CRM was built to meet this demand. Even if your particular situation doesn't demand this level of scalability, you can feel secure knowing that Microsoft CRM can easily grow with your business.

Scale down to a single user desktop. A CRM solution isn't as effective as it could be unless the product is also available in a disconnected model. Microsoft CRM can be deployed inside of Microsoft Outlook® in both connected and disconnected models. The product was designed so that you can deliver your customizations automatically from your intranet-based solution to your desktop clients, smoothly and efficiently.

Separate, but equal, application and platform tiers. Multiple tiers separate business logic from application rendering logic from data management logic.[1]

Microsoft CRM lowers TCO by implementing a metadata-driven architecture. This model allows customers to extend the underlying database schema easily, without having to change any application or platform code. These upgrades will persist through product upgrades and future releases because the upgrades, in effect, will be isolated from the platform code, which operates

[1] For further explanation of the terms "platform" and "application," see the "General Architecture" section of this paper. From a very high level, the platform can be considered to be any part of the overall product from and including the application program interface (Application Program Interface [API] layer "down" to the database. Everything "above" the API layer, including any custom application extensions that call the API layer, can be considered the application for purposes of this discussion. The layer between the application and platform is generated by the build process and results in two managed Dynamic Link Libraries (DLLs), one that exposes the platform by way of SOAP and one for Component Object Model (COM).

on the base entities.[2] The platform itself can perform operations against custom attributes, such as reading and writing, by simply looking through the entity instance and the metadata that defines that particular class of entities.[3]

Product Architecture

This white paper will discuss the Microsoft CRM product architecture at a slightly higher level than most developers might expect; but both developers and managers alike should find the topics interesting and appropriate.

Microsoft CRM supports a physical, four-tier model, but there are times when it's convenient to think of the individual layers as further divisible. At this level of logic, it's possible to identify up to six possible, unique tiers. Don't confuse the number of tiers (or layers) with "chattiness" or with performance degradation; these are logical tiers—the physical model is more optimized. Each tier in both the logical and physical models performs a specific function and acts as a supplier to those tiers immediately surrounding it.

The user interface and presentation tier

Microsoft CRM has two distinct presentation components, Microsoft Internet Explorer as the rich, Web-based[4] user experience; and Microsoft Outlook as the rich, Microsoft Windows® operating system-based experience. Both components make use of the rendering technology built into Internet Explorer. The internal rendering engine is hosted inside Outlook to present the same general look and feel as the Web application, but without the need for a Web server.

Microsoft CRM uses the Dynamic HTML (DHTML) capabilities of Internet Explorer to place as much processing as possible on the client machine, thus freeing up expensive server resources

---

[2] An "entity" is a class of objects, such as Account, Contact or Case, that captures both content (the set of attributes which make up the entity) and behavior (the business logic that is expected from any object called Account or Contact or Case). As will be discussed later in this paper, the metadata captures more information about entities besides their basic implementation details—information such as relationships to other entities and registered event handlers.

This paper will use the terms "entity" and "class" interchangeably unless there is a specific distinction to be made. The term "object" will be used when discussing a specific (or sometimes generic) instance of a particular class.

[3] In object-oriented programming terms, things in the software that represent real-world business objects are called *classes*. An instance of an *entity* might be the computer representation of a specific object from that class of things. For example, Contact is the name of the class of all "contactable people," but "Elmer Smithson" is a specific instance of a Contact. For more information about object-oriented concepts (beyond the scope of this paper), see *Object-oriented Information Systems* by David A. Taylor.

[4] This document uses the terms "thin client," "web client," and "browser client" more or less interchangeably.

and helping Microsoft CRM better scale to fit business demands. The application tier generates the majority of the client-side code and delivers it for processing to the browser. Microsoft CRM makes extensive use of Internet Explorer behaviors and client-side script to reduce the overall network traffic. Both clients use this functionality to reduce overall load and network traffic.

Application and custom business logic

Just as there are two distinct presentation clients in Microsoft CRM, there are also two application layers. For the Outlook-based client, a lightweight application framework was built to support the Web-based application logic, but in a disconnected mode. The browser client uses Microsoft ASP.NET application components within an application framework. There is little business logic within the application tiers; the majority of application logic revolves around rendering and data validation. The Microsoft CRM application layer only adds a small amount of customized business logic; the majority was "built-in" to the platform itself.

Both application tiers use XML messages to communicate with the underlying platform layer. Communication between the application tier and the platform tier uses SOAP for both Outlook- and browser-based clients.[5]

Service tier: domain logic[6]

The service tier is responsible for creating problem domain-specific objects. These include in the Microsoft CRM space such things as contact, lead, opportunity, account, and business. The goal

[5] In the as-shipped version 1.0 product, the Microsoft ASP.NET application uses COM APIs instead of SOAP APIs for communication with the underlying platform layer. This implementation detail will change in subsequent releases. The Outlook client uses SOAP both online and offline. When the Outlook client is online, all communication occurs from the local running application to a central platform server. When offline, the Outlook client uses SOAP locally (over-named Pipes) to communicate between the application layer and the platform layer. The COM interfaces exposed from the platform layer may be removed in future product releases. The recommendation is that all extensions and add-ons be written to the SOAP interfaces found in *Microsoft.CRM.Proxy.DLL*.

An additional recommendation is that the as-supplied proxy interfaces be used for interacting with the platform layer. Because of the way the platform APIs have been packaged, if two or more interfaces should be combined in the same managed assembly, there is a possibility of namespace collisions for core data types, such as CUserAuth and SecurityPrincipalType. The Microsoft CRM build process normalizes these namespace clashes by constructing a single, managed DLL with common data types collapsed into single type library entries. The clashes are a by-product of the use of Microsoft Active Template Library (ATL) code and common header files between the platform DLLs—many data types are shared by all platform classes, but are exposed in the type library separately. This limitation makes the use of Web service tools, such as wsdl.exe and sproxy.exe, very difficult for generating platform SOAP interfaces.

[6] Throughout this document, the terms "service tier" and "domain tier" can be assumed to mean any logic implemented within the Microsoft CRM platform object model. These terms are also interchangeable when discussing other "business" logic implementations.

of the service tier is to implement the service specific rules by manipulating and combining the underlying domain objects.

The service tier does not impose per-business logic. This layer imposes only generic domain constraints. This layer contains the building blocks for an application; but, by itself, is nothing more than a collection of related objects. However, the interaction between those objects within the domain can be assumed to implement more extensible logic, such as the Quote to Order to Invoice processing and pricing logic.

The service tier contains more than just the Microsoft CRM business objects. It's responsible for controlling access to objects and the database, raising events for workflow processes and custom business logic implementations. Think of the platform layer as implementing all of the "hard plumbing" work necessary to implement a complete Microsoft CRM application. The platform layer serves up both inbound and outbound e-mail processing by way of the Microsoft CRM Exchange Connector, and implements a replaceable pricing module for opportunities, quotes, orders, and sales invoices.[7]

Common service data tier

The data tier is the lowest level and consists of a database (loosely speaking) and a well-defined data access layer. The data management tier with Microsoft CRM supports Microsoft SQL Server 2000[8] as the primary data store. The data access layer provides a consistent programming interface for the service developer, which abstracts platform business logic from the underlying data source. The domain tier is the only direct consumer of the common service data layer.

Remapping the logical

As stated earlier, it's possible to identify up to six unique layers within the general product architecture. These layers are strictly logical and, in any reasonably performant business application, need to be combined to achieve optimal performance characteristics. This raises an interesting problem: how can the logical architecture be re-factored to meet these physical demands?

---

[7] Information on creating a replacement pricing engine can be found in the Microsoft CRM platform SDK. The as-shipped pricing engine closely follows existing pricing engines found in other Microsoft Business Solutions products, such as Microsoft Business Solutions–Great Plains®.

[8] The data tier chosen for Microsoft CRM is Microsoft SQL Server 2000 on the platform side and SQL Server 2000 Desktop Engine on the disconnected client side. This choice drives some of the later decisions about data access code and the structure of different queries.

The first part of the solution might seem somewhat counter-intuitive—as in pushing the layers back together. Access paths from layer to layer now need to be minimized. The architecture presented earlier covers each possible layer in the model, but does not require implementing each logical layer as a discrete physical layer. One of the initial constraints that will be placed on the design is that of speed–can a "guarantee" (figuratively, not literally) be made that the system will be performant enough to deploy for the number of users expected within the target market segment?

Constructing the architectural model

There's a natural separation between the application layers and the service layers, one that can be the initial architectural division. It's quite natural to do so, because it's simple to host the different components on different servers, and each of those layers really encapsulates all the information necessary to provide a complete service. The application layer holds all of the application logic, including rendering, custom business logic and data validation. The service layer holds all the service-specific domain knowledge and the store.[9]

However, there is no need to maintain a limit of two layers—it's safe to assume that there is more separation between the domain logic in the service layer and the underlying storage mechanism. This separation is also a natural boundary for most relational database applications–the store hides the data access mechanism and exposes a common interface, in this case the SQL programming language. Microsoft CRM internally uses an object-based data management layer to shield the platform code from the data access code. Just as it's not possible for an application developer to directly access the database; it's also not possible for a platform developer to do so. This object-based data management construct has been exposed to the application developer in the form of the **ICRMQuery** interface, along with the <fetch> query language. These components are discussed later in this paper and in some detail in the Microsoft CRM platform software development kit (SDK).

[9] Throughout the product, it's possible to find that each layer harbors a certain level of distrust of its surrounding layers. For example, the application layer implements a parallel user security model, effectively locking out functionality that is not available to the user. The platform layer doesn't know that the application has already performed a pre-emptive security check, so it performs another security check before actually servicing the application's request. This level of paranoia is necessary because the platform layer can't necessarily "know" that a given caller has already performed a security check and, by that measure, can't verify the identity of the caller, nor the rights of the individual to perform a given function.

Dividing the hierarchy

The overall view of the architecture changes the closer you get to it. Picture those little, wooden, dolls–the ones that are nested one inside of another. The system's architecture resembles that structure. An example would be as follows.

In the Microsoft CRM product, the service tier contains several higher-level boxes, each of which contains multiple lower-level boxes:

Sales Force Automation provides sales professionals with access to critical customer information and tools that enhance the sales force's ability to sell effectively as well as manage time efficiently. Sales force automation tools include contact management, calendaring functions, forecasting tools, and configuration models. Sales force automation encompasses:

- Opportunity and sales lead management—getting and controlling leads and opportunities
- Contact management—helping to make your customer YOUR CUSTOMER by tracking *personal information*
- Account management—helping to make your customer YOUR CUSTOMER by tracking the *customer's business information*
- Sales processes—taking a sales lead, helping to turn it into an opportunity, and helping to move it to a closed sale
- Competitor and referral libraries—tracking where leads come from and who competes for the customer
- Sales force management—managing sales quotas and commissions
- Quote generation—helping to make it easier for the customer to get the right price, fast.

Marketing functionality gives marketing departments the ability to send personalized letters or emails and to import lead lists. Closed-loop lead management is one of the most important functions of marketing automation and relies on integration with the Microsoft CRM data repository and related applications. Marketing tools include:

- Direct marketing—e-mail, paper mail, to get the message out and help generate leads
- Campaign tracking—functions to help track which campaigns are creating leads

Customer Service and Support helps make it possible for businesses to address customer questions, problems, or issues effectively and efficiently. While customer satisfaction is the primary goal, many organizations seek to increase revenues while providing customer service through "cross-selling," using:

- A knowledge base that helps customers find answers to their questions
- Product literature and document management that provides customers with information
- Customer care that manages both pre- and post-sales customer interactions
- RMA that helps the customer get the right product at the right time, and fixes problems quickly.

General Architecture

Now that we've discussed the logical layers that can be used to implement the Microsoft CRM product, along with how to put several of those layers together to create a performant product, let's discuss how to implement Microsoft CRM and how it uses those different layers.

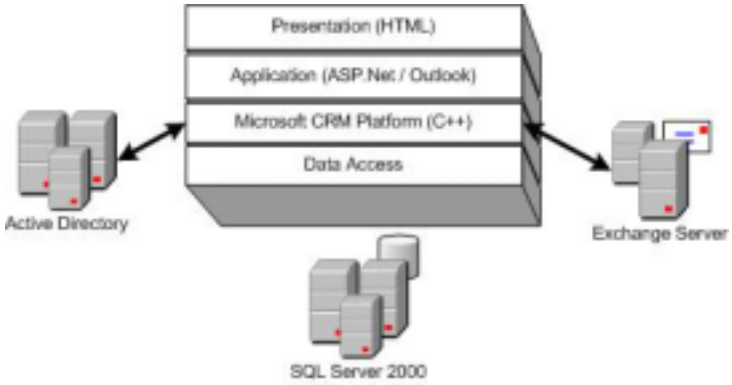


Figure 1. General Microsoft CRM architectural model

In general, there are four physical layers within the Microsoft CRM architectural "stack": presentation, application, business or domain logic, and data management. For discussion purposes in this white paper, we'll combine the data management and data store layers into a

single tier, although they are physically separated by virtue of existing in separate process spaces.[10] Microsoft CRM relies on several external components to perform additional processing. Microsoft Exchange Server is used for sending and receiving e-mail messages. Microsoft Active Directory® service is used for identifying security principals, roles, and groups within the product. Microsoft CRM does not use Active Directory for storage of any business data.[11] The data storage mechanism is Microsoft SQL Server.

Before discussing the layers in detail, it will be useful to discuss the overall object model and security layer within the product architecture. These structures form the basis for all object access by way of the platform API set.

Security subsystem

The Microsoft CRM security system was designed with several goals. The first is to support the concept of sharing; where users can be granted access to objects for collaborative effort. The second is the converse: to prevent users from accessing objects that have not been shared with them. The final goal is to categorize types of users; forbidding customer service representatives from attempting to do sales force automation-related work, for example. The first two goals relate to object security, while the third has to do with role-based security. The object security model focuses on primary entities such as **Account, Contact, Incident, and Sales Order**. This is not to say that other entities don't carry the same security rules; it's better to assume that all entities are secure and watch for the occasional non-secure entity. This functionality should be directly exposed in future versions of the product. The current metadata design makes identifying secured entities quite easy, but it would be far more helpful to have a flag on **Entity**, instead. A simple method to identify secure entities is to look for the entities holding an attribute called **SecurityDescriptor**. A simple SQL query against the metadata might be:

```
select e.Name
from Entity e join Attribute a on e.EntityId = a.EntityId
where a.Name = 'SecurityDescriptor'
```

[10] Not only do the data management and data store tiers reside in separate processes, but also they can reside in separate physical servers. Occasionally, the Microsoft CRM team internally talks about the 4½ tier product stack, which combines both data tiers into 1½ tiers.

[11] This decision was made simply to reduce the impact on Microsoft Active Directory. To support the richness of Microsoft CRM business data, significant Active Directory schema modifications would have been required, and proved to be too costly an option for the product's target market.

The role-based security applies to those objects, but also applies to more general considerations, ranging from administration of users to distribution of bulk e-mail.

Business and organization structures in Microsoft CRM

Microsoft CRM uses an uncomplicated, but very powerful, organizational structure for construction of arbitrarily complex business hierarchies. Each installation can hold a single concept, called an "Organization."[12] Organizations are lightweight entities that provide a common location from which to locate other business units. Businesses are not strictly hierarchical in nature; however, they may be parented only by a single business unit.

There are three primary entities within the organizational structure, not including the "Organization" itself. These entities are *users*, *teams*, and *business units*. Users represent real people within the Active Directory; these are the people who use the product. Teams are arbitrary groups of users and will be discussed later in this paper. Business units are the primary container entity within the organizational hierarchy. It's their structure that determines and defines the concepts of *basic*, *local*, *deep*, and *global* access. Consider the diagram in Figure 2. Teams are constructed as Active Directory groups; business units are "child" organizational units (OU) of the "parent" organization.

[12] In the as-shipped, public version of Microsoft CRM 1.0, the installation supported only a single organizational structure. However, subsequently, the platform and database have been designed and implemented to support multiple organizations. The setup process does not directly support multi-organizational installation, but all platform APIs are available. The 1.0 application, on the other hand, was designed with certain limitations built-in, such as the inability to deal with multi-organizational installations (the application does support multiple business units, but only a single organization).

There are a number of technical limitations to multi-organizational installations. The most important limitation is that semantic entity customizations are meaningless when crossing organizational boundaries within the product. That is, a given attribute may be interpreted differently from one organization to the next. The solution for this interpretative variance is to use a transformation engine (such as Microsoft BizTalk®) when moving objects between organizations. Because of this limitation, visibility of objects across organizational boundaries is strictly prohibited by the platform security model.

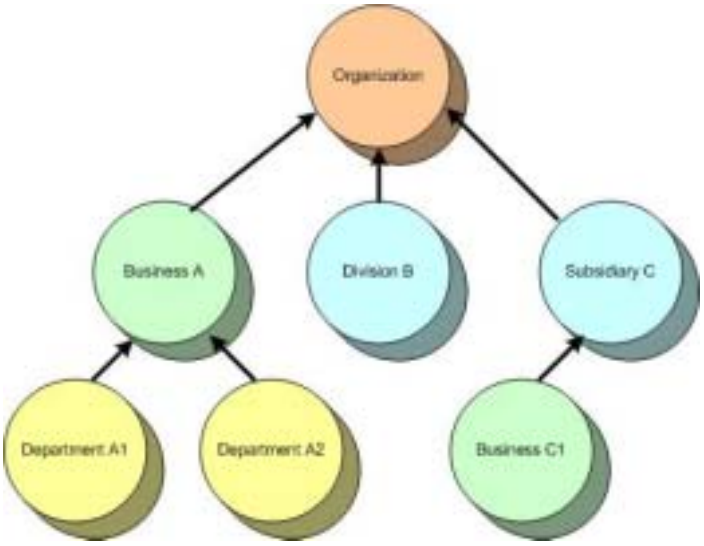


Figure 2. Example of a Business Organizational Structure in Microsoft CRM

The organization contains six business units in a simple hierarchy. The business units directly below the organization are unrelated in any apparent structural way with the simple exception that they all belong to the same organizational structure. The two business units, labeled "Department A1" and "Department A2," are sub-businesses of "Business A." These two business units have a special relationship with users parented to "Business A"–the business units serve up business objects to users within Business A playing a *deep* role.[13] Note that it's not possible to construct matrix organizations within the Microsoft CRM organizational structure.

Within this structure, users can be created from Active Directory users. Users within Microsoft CRM must be parented to a business unit, and cannot be parented to the organization object itself. In the diagram (Figure 2), users can be located at any node except for the top-level node, labeled Organization.

Security principals within Microsoft CRM

Microsoft CRM defines a "security principal" as an entity that can effectively own, or access an object within the system. There are two primary types of security principal within Microsoft CRM: users and teams. With few exceptions, all entities that can be owned by users can also be owned

[13] The concepts of *basic*, *local*, *deep*, and *global* roles will be covered in a later section. The concepts are so intertwined, however, that it's impossible to actually discuss business structures without at least mentioning the terminology.

by teams.[14] Only users can belong to teams; it's not possible to construct teams containing other teams. Only users hold privileges and, by extension, roles; it's not possible to assign a role to a team.

Roles

Roles are collections of privileges that are granted to users. For example, the Salesperson role may contain privileges such as, *User can read accounts*, or *User can write accounts,* whereas the Sales Manager role may contain privileges such as, *User can read account (LOCAL)*, or *User can assign contacts (LOCAL)*. A user can play simultaneously any number of roles; for example, a user can play the Sales Manager role in addition to being a Customer Service Representative, with all the privileges of both roles.

Roles are configurable within Microsoft CRM and may be modified or even removed as necessary to fit the business needs. Microsoft CRM uses privileges, not the role, as the core of the underlying security check. Privileges are "built-in" with the product and are used throughout the application and platform layers. It's not possible to add or remove privileges, nor is it possible to change how privileges are used to grant access to certain functionality. But it is possible to construct new roles from the existing privilege set.

Teams

A team is simply a group of users. A team does not play any roles and does not have any privileges. A team can be granted access to objects in the same manner as a user, and a team can even own objects. User privileges determine each team member's user access to objects shared with the team.

For example: Two users, Joe and Sara, are a member of the same team. Joe is a Salesperson and, therefore, has privileges to read opportunities. Sara is a Customer Service Representative and does not have the same privileges as Joe does. If an opportunity becomes available to the team, then Joe will have access to it, while Sara will not.

[14] The exception is routable entities, such as **Incidents** and all types of **Activities**. Changing the ownership of a routable object involves placing the object in a queue or removing it from a queue. In this context, the queue might be thought of as owning the object; however, this isn't the case, and the two behaviors should not be confused.

Role-Based Security

First, a brief introduction to the main concepts involved. A deployment consists of (usually) one top-level business called an organization. Each business in the system can have "child" businesses. Every user of the system belongs to exactly one business. Users can belong to an arbitrary number of teams. Each team belongs to exactly one business, but can consist of users from multiple businesses. Users and teams both can have access to objects. Each object is considered as belonging to one business, usually (but not always) the business to which the creating user belongs.

The fundamental concept in role-based security is that of privilege, defined at design time, on a system-wide basis. Each user has a set of privileges (there are well over a hundred privileges) that are enabled for that user. However, Policies and Roles grant privileges and simplify the process.

A variety of privileges is available, some of which control access to such actions as allowing the user to send bulk mail, or have private contacts, or create sub-businesses. In addition, there are privileges that relate directly to the objects in the system.

These privileges apply to an entire class of objects, rather than individual instances of those objects. If a user does not have the privilege to read accounts, then any attempt to read an account will fail. In terms of object security, there are three basic kinds of privileges: Basic, Local, and Deep. These are best illustrated through examples.

Basic, as the name implies, refers to the most basic privileges. If a user has the "Basic Read Account" privilege, then that user has the privilege to read accounts owned by or shared with the user.

Local privilege, such as "Local Read Account," means that a user can read all accounts in the local business.

Deep privileges allow the specified access to all objects in all businesses that are in the user's business, or that are in the user's sub-businesses. Therefore, given "Deep Read Account" privileges, that user can read all accounts in his or her business, as well as read all accounts in any sub-businesses of that business.

Microsoft CRM impact on Active Directory

Microsoft CRM uses Active Directory to store teams, roles, and business units. Browsing through the Active Directory objects created by Microsoft CRM should be done with some care to avoid

confusion. For instance, each role shipped with, or created by, the product shows up as an OU. Teams are implemented and behave as expected—when a user is added to a team, the user becomes a member of that team and shows up in Active Directory. However, roles are quite different.

Roles, when created by Microsoft CRM, may actually be implemented as three distinct Active Directory objects. Each follows a particular naming convention:

Microsoft CRM ***Role*** (System Admin)–This is the Windows NT® group, which represents the "Local" role. Users who are members of this role will actually appear within Active Directory as members of the Windows NT group.

Microsoft CRM ***Glbl*** (System Admin)–The "Global" role grants members access to corresponding objects throughout an organizational hierarchy. There are no users in this group; instead, the "Role" is added as the member, and members of that group are implicitly added to the Global role.

Microsoft CRM ***Deep*** (System Admin) – The "Deep" role grants members access to corresponding objects from their home business unit throughout all "child" business units (of their home business unit). As in the Global role, the Deep role contains no user members; instead, they are implicitly members because the Local role has been added as a member of the Deep role.

With regard to the "System Admin" role:

- On objects for which the role grants the Global read privilege, Microsoft CRM grants read access to the Microsoft CRM ***Glbl*** (System Admin) group on every object of that type in the organization.
- On objects for which the role grants the Deep read privilege, Microsoft CRM grants read access to the Microsoft CRM ***Deep*** (System Admin) group on every object of that type in the business and all of its "child" sub-businesses.
- On objects for which the role grants the Local read privilege, Microsoft CRM grants read access to the Microsoft CRM ***Role*** (System Admin) group on every object of that type in the business.

Here is one example of this role structure and how it works.

Assume John is a member of the System Admin role, a role defined as having Global read privilege on accounts, Deep read privilege on contacts, and Local read privilege on sales leads.

- With these privileges, John can read all accounts in the organization, because John is a member of the Microsoft CRM ***Role*** (System Admin) group that, in turn, is a member of the Microsoft CRM ***Glbl*** (System Admin) group, and this ***Glbl*** group has been granted read access on all accounts in John's organization.
- Similarly, John can read all contacts in his business and all of its "child" sub-businesses, because John is a member of the Microsoft CRM ***Role*** (System Admin) group that, in turn, is a member of the Microsoft CRM ***Deep*** (System Admin) group, and this ***Deep*** group has been granted read access on all contacts in John's business and in all of its "child" sub-businesses.
- In addition, John can read all leads in his business, because John is a member of the Microsoft CRM ***Role*** (System Admin) group, and this ***Role*** group has been granted read access on all leads in John's "home" business.

When a role has no Global privileges, Microsoft CRM does not create the ***Glbl*** group. The same thing holds for a role that possesses no Deep privileges; Microsoft CRM will not create the ***Deep*** group. However, the Local or Microsoft CRM **Role** (System Admin) group will be created, regardless of any local privileges added to the role. Whenever Global or Deep privileges have been added to the role, Microsoft CRM creates the respective ***Glbl*** or **Deep** groups if they haven't already been created.

Object-Level Security

The other form of security applies to individual instances of objects. There is a fundamental difference between an access right and a privilege: an access right is a right granted to a user on an object), but a privilege is a right granted to a user on a class of objects. Access rights apply only after privileges have taken effect. In other words, if a user does not have the privilege to read accounts, the user will not be able to read any account, whether or not it has been shared.

There are a number of access rights that a user can have on an object: **Read**, **Write**, A**ssign**, **Append**, **Share**, and **Delete**. Note that Create does not appear in this list because the ability to create an object does not apply to an individual object, but rather applies to a class of objects. Privilege, not access, governs the right to Create. The user who creates an object will have, by default, all rights on that object (unless the user's privileges forbid a specific right).

Security descriptors

All "interesting" entities in the object model carry an additional, platform-only attribute called a security descriptor. A security descriptor is a structure and associated data that contains the security information for a securable object. A security descriptor identifies the object's owner and

primary group. The descriptor can also contain a Discretionary Access Control List (DACL),[15] controlling access to the object; and a System Access Control List (SACL),[16] controlling the logging of attempts to access the object. Because of the special nature of security descriptors—specifically how they're constructed by the platform security service—it's not possible to directly import data into the data store. Security descriptors have been constructed for each object within the database by the security service. The security service uses a combination of the calling user's roles and groups, plus the Local, Deep, and Global groups within the calling user's business hierarchy, plus the as-shared access rights to the object to construct the security descriptor. There is no way to "fake" the security descriptor for a given entity or object, and doing so would cause the security service to reject any attempt to access the object.

Microsoft CRM Security Service

The Microsoft CRM security service is responsible for a number of security operations at an object level. The functionality has been packaged in a Windows NT service executable and exposes its interface by way of named Pipes. The security service is responsible for the following:

- Construction of default security descriptors
- Security descriptor adjustment on role change
- Security descriptor adjustment on ownership change
- Security descriptor adjustment on access change (share change).

To reduce the packet size between the security Dynamic Link Library (DLL) and the platform's interface into the security service, object-level access checks are performed in the platform's process space by the security DLL itself. If the DLL didn't perform access checks, the entire object's security descriptor would need to be delivered to the security service; and security descriptors can be relatively large and, therefore, expensive. The platform object model uses a C++ template-based paradigm to introduce security into the classes. For secure objects, there are several possible base templates that can be used based on the object's target use. The workflow process needs access to objects but is limited in that the workflow doesn't have the

[15] A Discretionary Access Control List (DACL) is an access-control list that is controlled by the owner of an object and that specifies the access particular users or groups can have to the object.

[16] A System Access Control List (SACL) is an ACL that controls the generation of audit messages for attempts to access a securable object. The ability to get or set an object's SACL is controlled by a privilege typically held only by system administrators.

caller's security token.[17] Because of this limitation, workflow creates business objects based on a special security template that bypasses user token validation.[18]

Metadata as an Enabling Technology

Microsoft CRM is a metadata-driven product, meaning that the vast majority of its work is focused on abstracting itself away from the implementation details that so often cause problems with upgrades and extensibility. The metadata layer, in essence, abstracts the underlying data storage details, such as schema and data access, from the higher-level constructs of domain logic implementation and user interface. The word "metadata" literally means descriptive information about the elements of a set of data. Think of metadata as a description of the underlying data structures, one that controls how the application—platform and user interface—operates and displays itself.

The platform code uses the metadata to protect itself from changes to the underlying database structures. Should a table definition change, for example, when adding or removing columns, the platform code will continue to operate without any performance or degradation. There are limitations to the level of protection that a metadata layer can afford any platform that implements business logic. Should *key* relationships, those that the platform *expects to be present,* be modified in a significant way, the platform cannot be expected to perform in the expected way. However, other relationships, which are synthesized by the platform in order to construct full entity definitions, may change over time without adversely affecting the platform code.

Using metadata driven technology allows Microsoft CRM to be significantly altered to meet a particular business or vertical definition and still operate without interruption. This capability creates a world in which companies that specialize in creating vertical solutions on top of Microsoft CRM can expect that their investments in additional domain logic will be carried forward through upgrades and into additional modules.

---

[17] The security "token" is a small piece of operation system-supplied information that identifies the API caller. The platform uses NTLM (NT LanMan) security for both authorization and authentication and, as a result, has access to the security token of the caller. This paper will not cover how the token is passed from the end user to the platform.

[18] Most platform APIs validate the CUserAuth structure's contents with the security token on the calling thread to verify that the actual caller is the intended target caller. This parameter must be supplied by any application using the platform APIs. The Microsoft CRM application itself always defaults the **CUserAuth** contents to the "current" user's platform user ID and merchant ID. There are rare cases where the platform will allow a mismatch between CUserAuth and the caller's token. Windows NT® allows mismatches to users who belong to a privileged user group; there are two expected members in this group, one for the user context under which the workflow service runs, and one for the Crystal Reports services user.

The Microsoft CRM platform isn't the only consumer of the metadata. The application layer uses the rules inherent in the metadata to present the exact user experience offered by vertical solutions and customizations. These rules include attribute type definitions, entity definitions, and attribute context rules.

Attribute type definitions describe the underlying type structure of a given attribute, including the fundamental data type (for example, **string**, **integer**, **date**) and information, attributes that effectively limit the attribute's type definition (such as its size and range values).

Attribute context rules describe when and how a given attribute can be used. For example, some attributes are *write-once*, such as order numbers. Once created, these attributes shouldn't be changed because doing so would cause ripple effects throughout the business data. Other attributes are always *read-only* and supplied by the platform itself. The metadata captures all these rules about context, but also captures *business-defined* rules, such as "business recommended" and "business required" attributes. The forms engine can use this information to render the proper user interface and do so in such a way that application code itself doesn't need to change.

The actual metadata *model*: an introduction

This section first presents a quick introduction to the actual metadata *model* for those wishing additional technical information. The discussion in the remainder of this section is more technical than the rest of this paper and is intended for those individuals who wish to see more detail. This model is used throughout the Microsoft CRM product and captures descriptive information about all of the entities, attributes, and relationships that exist in the product.

The core of the metadata model is the **Entity**. Entity, and its related table **Attribute**, capture the base *class* definition of all objects found in the product. When the platform is performing an operation against a particular entity class, it uses the metadata to discover the preferred behavior of that entity. For example, upon creating a new object, the platform will query the metadata to determine if any interested parties have registered event handlers for the "create" event. There are two types of handlers, workflow processes and custom business logic implementations.

Although Microsoft SQL Server supplies a large amount of metadata, that information is limited to a representation of the physical database model. Microsoft CRM needs a large amount of logical information to control *how* the database is used, not simply what is present in the database. If SQL Server had a mechanism to allow for user-defined annotations on tables, columns, and relationships, it might be possible to use more of the built-in metadata; however, there are other

structures in the metabase that have no corresponding constructs within SQL Server. These concepts still need to be managed separately from the SQL Server store.

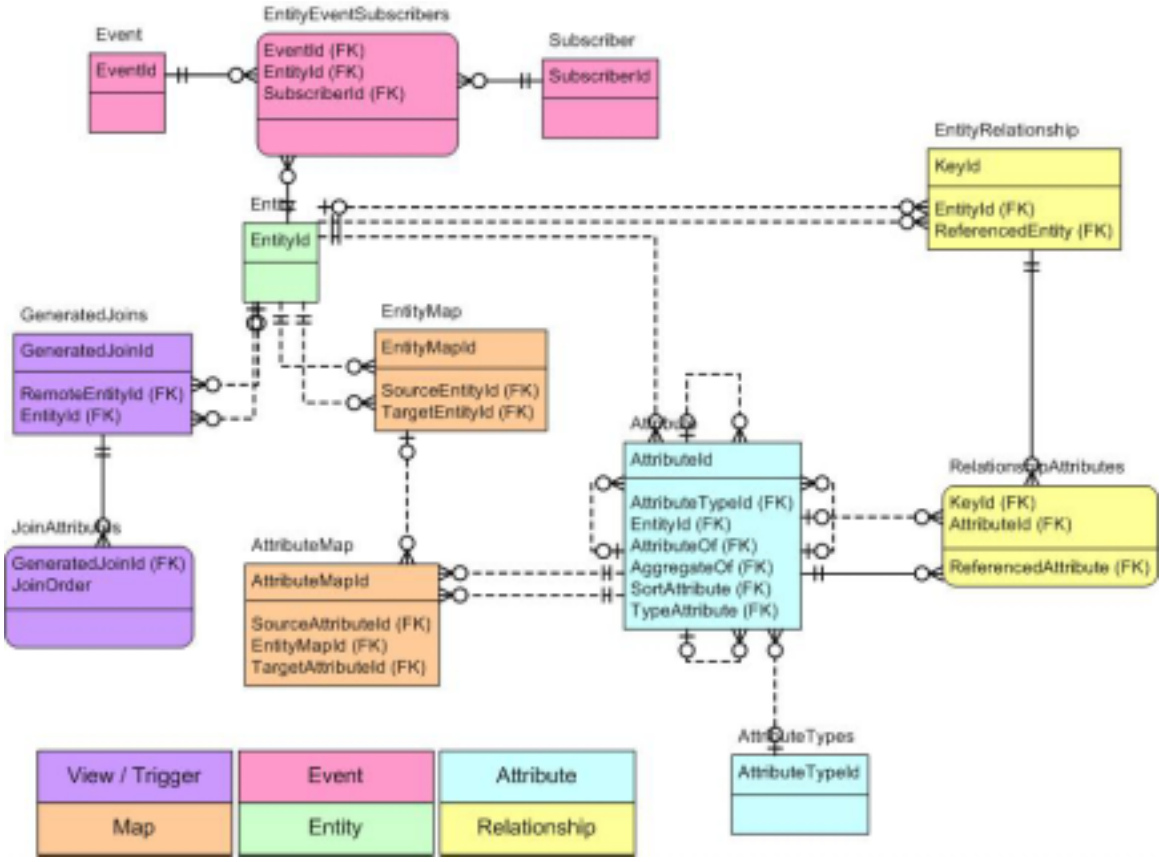


Figure 3. High-level view of the Microsoft CRM metadata model

The Microsoft CRM platform, along with the associated metadata, provides a simple mechanism to build other CRM applications using the platform. For instance, a complete, hypertext-based object navigation system could be constructed using only the metadata and the entity rendering definitions. This hypothetical application could, for instance, provide a feature whereby it would be possible to locate an arbitrary starting point, say a particular contact, and browse everything related to that contact, such as orders placed, opportunities closed, and activities in which the contact has participated. Along the way, it would be possible to "branch off" the main navigation path—from activities, for example—to view other parties of that activity. Those parties could be other contacts, from which the navigation might start again.

Considering schema approaches: history

The Microsoft CRM team investigated four different proposals as possible schema extensibility mechanisms. For background perspective, a brief history and description of each proposal

follows. Note, however, that only the fourth proposal was implemented (as has been discussed in the previous sections).

Inverted property bag–The approach taken for Microsoft bCentral™ Customer Manager was to use a technique called an *inverted property bag*. This approach stores additional attribute in a related table to the main entity, but in a name-value pairing. Each entity (business, contact, lead, or activity) has a set of developer-defined named properties. This approach was an attempt at solving the problems inherent in the third approach. However the inverted property bag approach quickly caused two severe problems. First, performance was unacceptable; each query required multiple outer joins to gather all the detail-level information. Secondly, the rapidly stored data exploded. For each object in the main table, a corresponding set of rows would be present in the extended table. Even if the main table were quite short (that is, few records), the extended table could be quite tall, with one or more extended name-value pairs for each attribute.

Opaque, application-defined blob–The second approach was to expose a single, opaque, application-specific *blob* field on every interesting object. This approach had some appeal because it left all the interpretation to the application, leaving the developer to manage and render this information as necessary. The drawback is that the blob isn't quickly searchable and can't be indexed; full-text indexing is an option, but the query clauses necessary to *reach into* the blob become very difficult to construct.

Another drawback with this approach is that simple queries against the data are difficult to construct and very expensive to run. For example, how would one construct a query that found all persons who brought cats into a veterinary clinic in May and whose cats were serviced by Dr. Smothers? If this data were to be "stuffed" into a single XML blob, the platform could not control the format; therefore, a generic query like this wouldn't be possible to construct.

A secondary difficulty with this *blob field* approach is the opaqueness of the data: neither the application nor the platform has any knowledge of the document structure. To make any reasonable use of the data, the platform would need to be written with the document structure in mind, in which case, the advantage of the extensibility mechanism has been defeated. On the other hand, the application might have knowledge of the structure, but might not have any guarantee on its structure. That is, the structure might need to be interpreted differently for each individual object. If the application were to force a fixed document structure on each class of objects, that action would reduce some of the problems.

Fixed customization model–This third approach supplies a fixed number of customizable fields for each object–say, five or ten *untyped* fields. The problem with this approach is that it breaks the "*zero*, *one*, *infinity rule*." If a product were to offer developers a fixed number of fields, all too

soon it would be obvious that developers actually would need yet one more. If developers were to be told they had 255 UNICODE characters per field, they'd ask for 256. By implementing the extra fields as a sql_variant, the second part of this problem can be avoided; however, doing so limits the field's usefulness because the meaning of the field would be changed in large searches.

Metadata model: Microsoft CRM–The fourth and adopted approach uses a metadata-driven model and "hides" the physical model from the application and platform developers. The appeal here is that each developer has an opportunity to think about the problem and customize the object definition to meet the need. This Microsoft CRM uses this approach for exposing product extensibility. This model also brings other benefits to the overall application environment, benefits that will be discussed in further detail later in this document.

Accessing the platform through API: an example

This section presents a simple example of reading a list of contacts for a particular user and includes the complete example code, followed by a detailed explanation. This particular pattern is very common throughout the product and represents a very typical usage scenario.

```
1.  using Microsoft.CRM;
2.  using System.Net;

3.  Microsoft.CRM.CUserAuth ua = new Microsoft.CRM.CUserAuth();
4.  Microsoft.CRM.CSecurityPrincipal sp = new Microsoft.CRM.CSecurityPrincipal();
5.  Microsoft.CRM.CRMContact c = new Microsoft.CRM.CRMContact();

6.  ua.MerchantId = "{A8A05D04-2068-45EE-9766-5E89950B8176}";
7.  ua.UserId = "{98197F09-3E7A-414E-9105-5AC2F438477F}";

8.  sp.Type = (SecurityPrincipalType)Microsoft.CRM.SecurityPrincipalType.sptUser;
9.  sp.Id = ua.UserId;

10. c.Url = "http://localhost/MSCRMServices/CRMContact.srf";
11. c.Credentials = System.Net.CredentialCache.DefaultCredentials;

12. string columns =
13. "<columnset>" +
14. "  <column>contactid</column>" +
15. "  <column>accountidname</column>" +
16. "  <column>firstname</column>" +
17. "  <column>lastname</column>" +
18. "</columnset>";

19. string xml = c.RetrieveByPrincipal(ua, sp, columns);
```

Figure 4. An example of reading a list of a list of contacts for a particular user

Lines 1 and 2 add references to the Microsoft CRM platform proxy layer.[19] *System.Net* is included because all SOAP-based clients are based on the types and interfaces found in that assembly.

Line 3 creates a user authorization structure which is needed by all platform APIs. The **CUserAuth** structure identifies, in a textual manner, the intended identity of the calling user. The platform is capable of inferring the caller's identity and actually does so, but uses the **CUserAuth** to implement an act-on-behalf-of functionality.

Line 4 constructs a **CSecurityPrincipal** structure as required by the call in Line 19.

Line 5 creates a **CRMContact** interface proxy.

Lines 6 and 7 fill in data for the user authorization, and

Lines 8 and 9 fill in the security principal structure by using the caller's identity.

Line 10 sets up the proxy interface's target Web service.

Line 11 is very important, as it tells the underlying SOAP transport layer to "flow" the caller's NTLM (NT LanMan authentication) credentials from the running application through to the Web service. Without this code, the SOAP proxy will not assume any credentials, and the Web service will fail the call because it won't be able to gather security credentials for the calling user.

Line 12 sets up a <columnset> list for the API call. Nearly all APIs that return single objects, and all APIs that return multiple objects, allow the application to limit the set of attributes returned for that result set.

Line 19, the final code, makes the call to the Web service, passing in the user authorization, the security principal who is the target of the call, and the attribute list that should be returned. Attribute lists, while called column sets, are actually based on the logical attribute names within the entity or entities queried.

The platform code, which implements the call, verifies the caller's identity, checks the caller's privileges, and constructs a query against the data store using the data management layer. As the data is streamed back from the database, the security layer built into the platform object

---

[19] It's possible to access the platform by using tools such as Web Services Description Language (WSDL) to construct client-side SOAP proxies. However, it's recommended that code uses the Microsoft-supplied proxy layer (which contains common definitions for data types that are normally present in individual web services). Simply add a reference to *Microsoft.CRM.Proxy.DLL* to your project.

model removes any rows from the result set, to which the calling user does not have access. Keep in mind that the platform code itself actually doesn't implement any of the above logic; rather, it focuses only on implementing the API logic–the platform infrastructure upon which the platform code has been built is solely responsible for all data management.

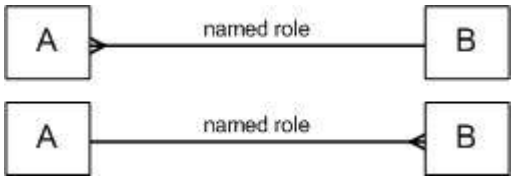
Querying the database: through <fetch>

There are times when it's only possible to construct a generic query against the database. Microsoft CRM exposes an interface designed specifically for this purpose. The **ICRMQuery** interface can be used to send queries to the platform layer for further processing. The interface uses a special query language, referred to as the <fetch> language, an XML-based query language that operates against the logical XML schemas. The language is roughly object-based in that it doesn't expose rows, columns, and tables; instead, exposes the Microsoft CRM entities and attributes.

With <fetch> it's possible to join two or more entities in an arbitrary manner. The language can use direct attribute-based join syntax, or can use a named relationship join. Attribute-based joins are much like SQL join syntax in that the caller identifies the two entities and the attributes on which they're related. Named relationship joins are more powerful join constructs in that they allow the caller to let the platform query processor identify the best join based on a logical entity relationship.

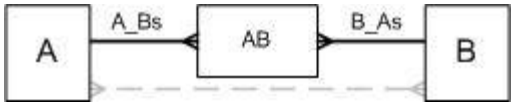
Before discussing named relationship joins further, it's necessary to introduce the naming conventions used within the database.

Named relationships–one-to-one, or one-to-many



The named relationship is a one-to-many (1:m) relationship between two entities that have a logical, named relationship. For example, Accounts have a 1:m relationship with Contacts. That is, a Contact may belong to one and only one Account. This requirement naturally leads to a logical relationship name: account_contacts. On the other hand, a Contact has another one-to-one (1:1) relationship with an Account that is the primary contact. This relationship is named account_primary_contact.

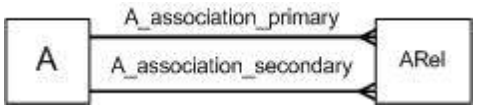
Logical many-to-many



A logical many-to-many (m:m) relationship materializes in the physical model as two 1:m relationships using an

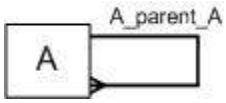
intersect table to hold the m:m relationships. For example, many Contacts have many Opportunities. The logical relationship name in this case is contact_opportunities. Because this logical relationship must be implemented as two physical relationships, we must first create an intersect table, to which we give the logical relationship name: ContactOpportunities. We then name the two relationship based on originating endpoint. In this case, the names are opportunity_contacts and contact_opportunities.

Self-relative many-to-many



A self-relative m:m relationship is a special case of the logical m:m relationship. However, because the originating entities are the same, the names must be changed (all relationship names must be unique within a database). The chosen convention is to use the entity name followed by one of _association_primary or _association_secondary. The intersect table is named ARelationship.

Self-parental one-to-many



A parental one-to-many is a special case of the named relationship. However, in this case, the naming convention highlights that the relationship is somehow different from the usual named relationship. At first, the chosen naming convention might seem contrived and a bit unnatural, but with consistent use, the convention becomes more natural. A Contact, for instance, has a special relationship with (zero or) one other Contact; that is, it is the child of another Contact. This relationship name becomes contact_parent_contact.

A <fetch> example

Figure 5 displays a <fetch> example that selects the set of account objects where the owner is the same security principal as the caller, and where the postal code attribute is not '"98052." The <fetch> language provides support for sorting and filtering. Line 7 demonstrates a simple ascending sort on the account name. Note that the attribute on which the query should be ordered need not be in the attribute list. The filter clause, in Lines 8 through 11, sets up the equivalent of a SQL "where" clause. In this example, the clause contains an "and-" type filter with two conditions. The condition in line L0 is self-explanatory. The condition in Line 9, however, presents a special feature of the <fetch> language–named attribute selection. The "eq-userid" is a parameter representing the calling user principal, allowing reusable queries for the query processor automatically to create parameters.

```
1.  <fetch version="1.0" output-format="xml-platform" mapping="logical">
2.      <entity name="account">
3.      <attribute name="name" />
4.      <attribute name="address1_city" />
5.      <attribute name="address1_stateorprovince" />
6.      <attribute name="address1_postalcode" />
7.      <order attribute="name" descending="false" />
8.      <filter type="and">
9.          <condition attribute="ownerid" operator="eq-userid" />
10.         <condition attribute="address1_postalcode" operator="ne" value="98052" />
11.     </filter>
12.     <attribute name="primarycontactid" />
13.     <attribute name="accountid" />
14.     </entity>
15. </fetch>
```

Figure 5. A <fetch> example.

Common forms Processor

Both user interface components of Microsoft CRM use a common forms processing module. This module is responsible for form layout and presentation, validation and formatting, and data transfer. By using this common forms processing component, Microsoft CRM can provide a common user experience in both the Outlook client and the thin client. To the independent software vendor and developer, this component provides the means to customize the user interface areas of the product once and have those same customizations automatically be available on both clients.

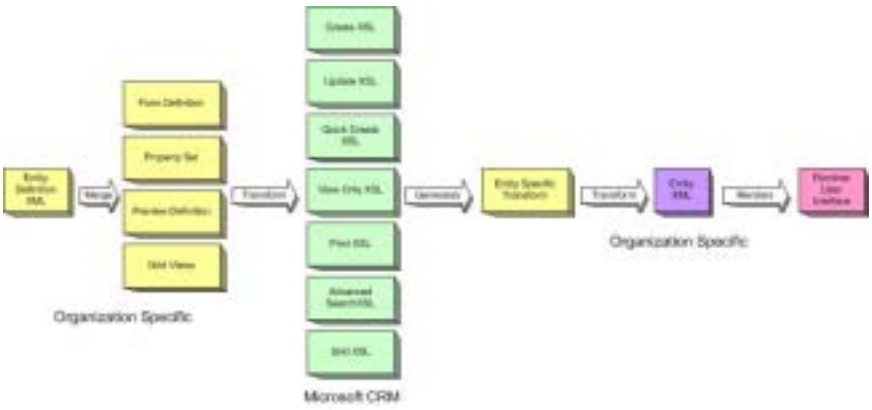


Figure 6. Microsoft CRM forms process components and process flow

The forms processor operates on two XML documents that can be customized: the entity definition and the form layout. These documents are custom views, by-Organization, on the object definition, and on the object presentation. These files are static definitions that can be built during product customization and then installed during the "deployment" product development phase. The customized documents transform during deployment by the Deployment Manager tool using seven Microsoft CRM-provided XSL transforms to build out entity- and (or)

Organization-specific rendering logic. Then, the as-deployed files install to the application's virtual directory. The thin client application layer uses an extensive caching mechanism to keep these entity-specific transforms in memory, providing far better response times than loading the documents on demand could otherwise produce.

User Interface Architecture–Web Client

The Microsoft CRM Web client uses ASP.Net to serve the user interface to Internet Explorer. The application framework has been specifically tuned to support high numbers of users, but without the expense of requiring a large server.

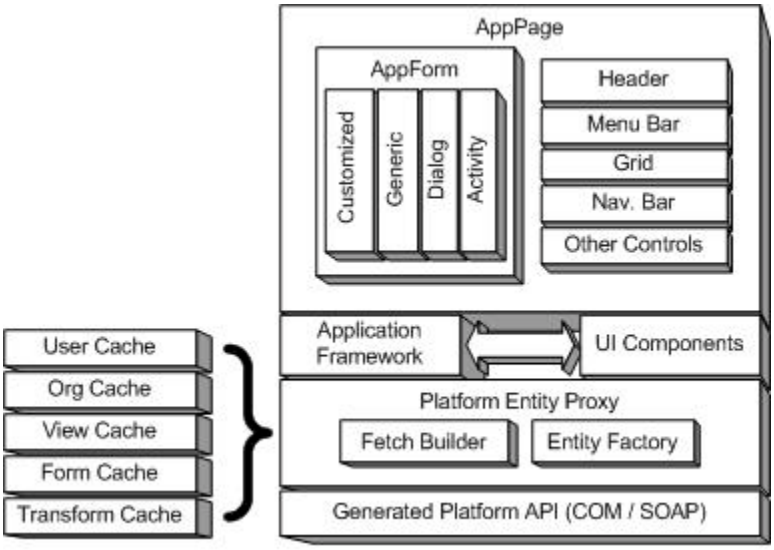


Figure 7. Microsoft CRM ASP.Net application architecture

User Interface Architecture–Outlook Client

Microsoft CRM includes an Outlook-integrated client containing all the functionality found in the Sales Standard Web application. The Outlook client operates as follows:

- Provides a disconnected user experience for salespeople who are not always connected to a corporate LAN. Salespeople tend to take their data with them where they need it. Microsoft CRM provides a completed disconnected client to meet this need.

- Keeps Outlook objects synchronized where overlap between Outlook and Microsoft CRM object naturally occurs. Where there is a natural overlap between Outlook and Microsoft

CRM objects, such as calendar items, e-mail, and contacts, keep these objects consistent between the two applications.

- Maintains product customization across both the Web-based and Outlook-based clients. Customization is a time-consuming and expensive task. Microsoft CRM reduces this cost by automatically moving user interface and schema customizations to the Outlook client.

- Provides a consistent user experience for all users, whether they are using the Web-based or the Outlook-based client. To reduce training costs, Microsoft CRM uses a consistent user interface for both its application clients.

- Maintains security when transferring data between the Outlook local store and the central business data store. Disconnected users should only be able to take "their" data with them when they disconnect from the central server. When reconnecting to the central server, data integrity rules must be honored for all "uploaded" data. Microsoft CRM meets both of these requirements by providing an integrated security model and a powerful transaction playback model.

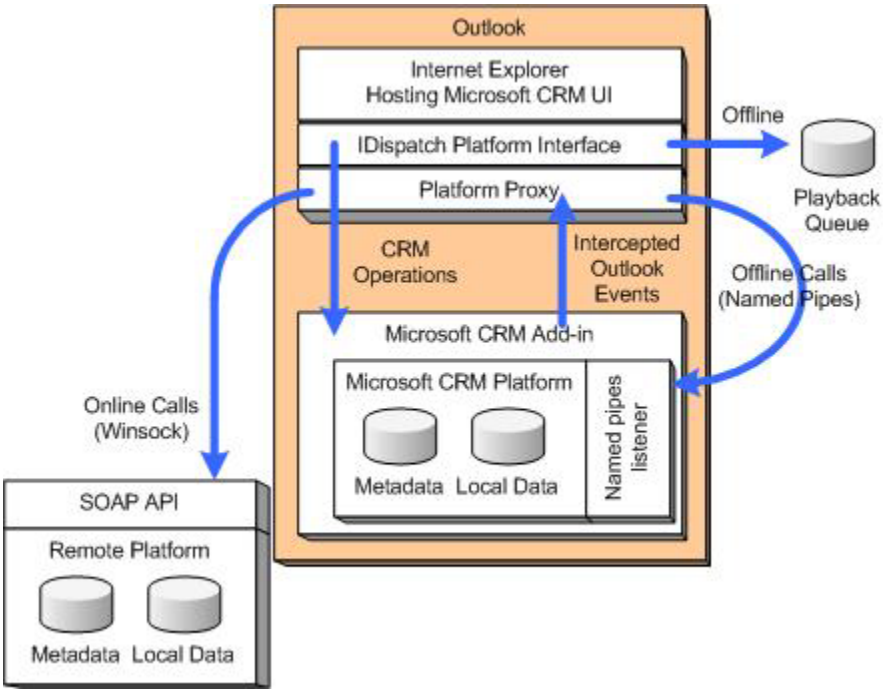


Figure 8. Sales for Outlook online architecture

"Rich client" architecture in online state

The Microsoft CRM Outlook client can be used in both online and offline modes. In the online mode, the client continues to run on the local machine and processes all application logic locally.

However, the primary business logic processing happens on the central Microsoft CRM server. Communication between the Outlook client, and implementation of the Microsoft CRM server, occurs through the use of SOAP over Transmission Control Protocol/Internet Protocol (TCP/IP). This transition between modes is transparent to the client and all client code. A "shim" layer between the application logic and the platform logic controls the message routing.

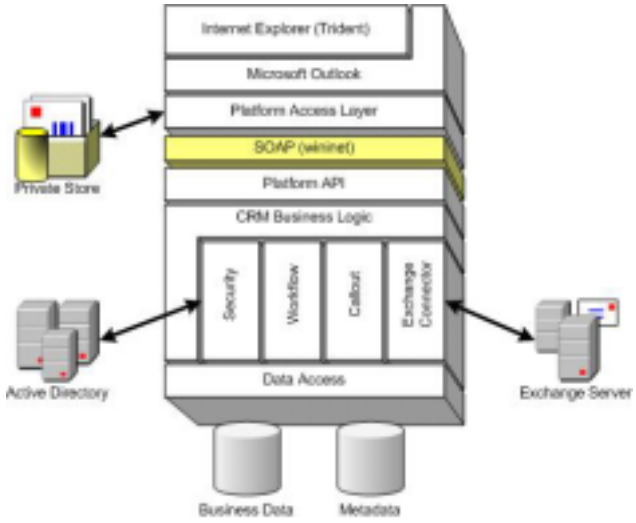


Figure 9. Sales for Outlook common online architecture

"Rich client" architecture in offline state

In its offline mode, the Outlook client continues to implement all application logic locally to the client machine. However, once offline, the shim layer changes the communication transport from TCP/IP to named Pipes. In this mode, the application logic is using a local instance of the Microsoft CRM platform code and business logic. The primary difference is that the shim layer sends the requests to the local platform for processing and stores all "write" requests to a local store called the "playback queue."

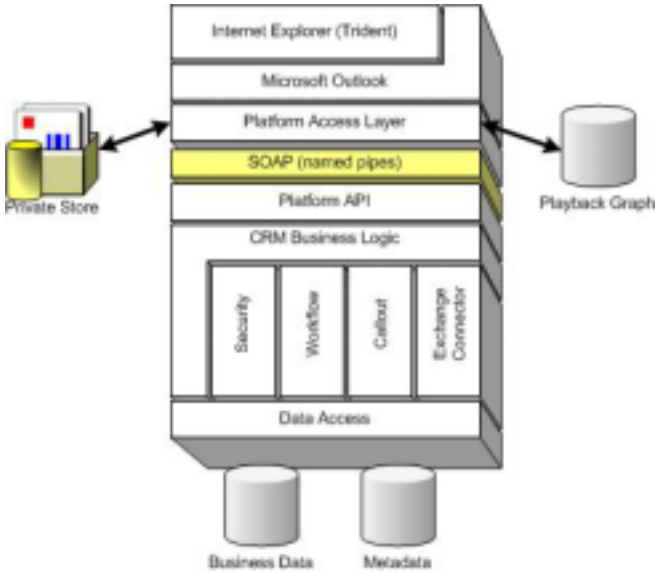


Figure 10. Sales for Outlook common offline architecture

Disconnecting from the primary server: "going offline"

The transition from online to offline (and *vice versa*) is controlled exclusively by the user. There is no automatic transition between modes as this might result in some confusion for the end user, essentially leading to the user asking the question, "Am I online or offline?"

Moving from online to offline can happen in two ways: the first is when the user chooses to disconnect from the central server; the second is when the user chooses to update the local store before actually disconnecting. In the second case, the local application makes a request to the central platform asking it to construct an "offline data set." This package is, simply put, a series of constructed views that SQL Server will use to perform standard merge replication to the client. Data that is being copied to the local client database does not need to be processed through platform business logic and security checks because that data already had been processed when it was entered into the central server.

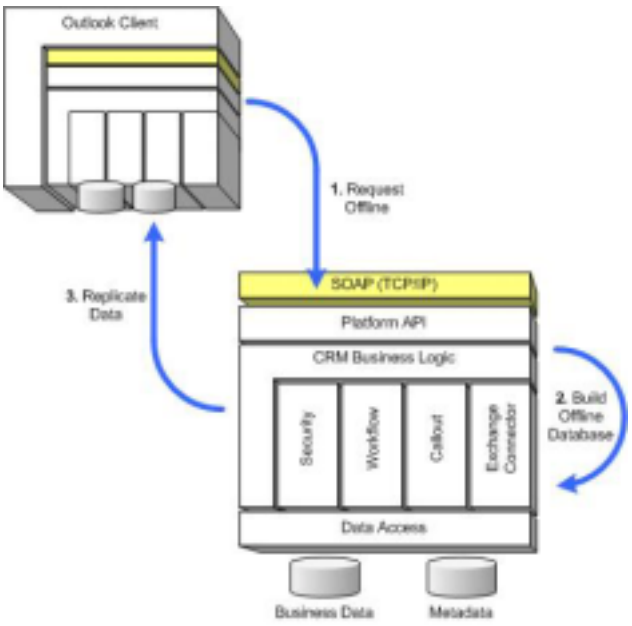


Figure 11. Transitioning Outlook client to offline state

Performing "write" operations while offline

When the Outlook client is running in offline mode, the platform shim makes one additional database write for all successful data modification actions against the local store. This additional write occurs against a second store called the playback graph. This graph records all SOAP-based API calls.

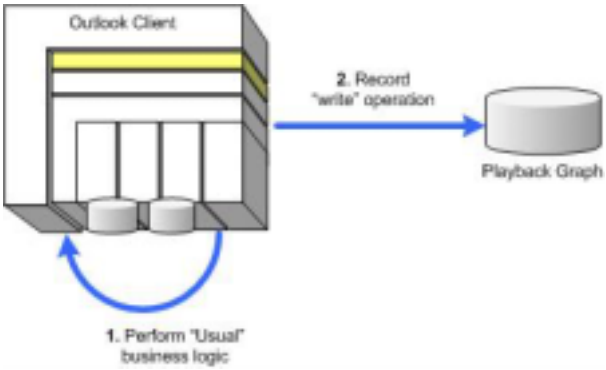


Figure 12. Offline data access and playback processing

Reconnecting to the primary server: "going online"

When the user chooses to reconnect to the central server, the platform shim bypasses the platform, and a synchronization process starts. Unlike the synchronization process for disconnecting from the server, reconnecting is somewhat more involved. Because Microsoft CRM needs to guarantee that all business logic will be performed on any and all actions submitted to

the central server, it doesn't use SQL replication. For example, while a user was disconnected, the user's security privileges might have been altered, or objects to which the user originally had access to might have been modified in such a way as to render that access obsolete. A separate, though related, reason why actions must be played back through central business logic is to guarantee that all objects work within current workflow and callout process rules.[20]

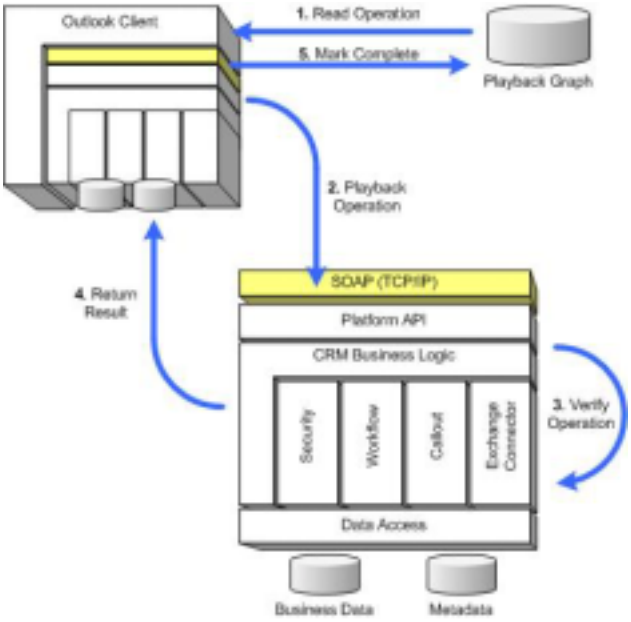


Figure 13. Transitioning Outlook client to online state

Reporting

Microsoft CRM uses Crystal Decisions reporting solution, Crystal Enterprise (version 8.5), for all reports. Access from the report design and report run-time to the database is prohibited. Instead, a platform access shim has been supplied by Crystal Enterprise, which converts design-time and run-time database metadata queries and data access queries into platform metadata queries and <fetch> requests. That is, the Crystal Enterprise tools use the platform API set to access the platform data. This shim allows the platform to manage all data access as if the report engines are simply additional applications. The overall Crystal Enterprise reports integration into Microsoft CRM will not be covered in this paper.

[20] Integration with third-party applications like Microsoft Business Solutions-Great Plains uses the callout mechanism for implementation. Because integration isn't possible for an offline user, the business logic must be triggered for those actions the user performed while offline. Using a playback process guarantees that the necessary processing occurs.

Internationalization Support

The forms processor reads data from the form elements, constructing an XML representation of the delta. While doing this, the forms processor converts the UNICODE data from the form fields into UTF-8 for transport to the platform web services. The platform converts UTF-8 to UNICODE before storage to the database. The forms processor uses a script code to perform the conversion. Without this conversion, the platform API would attempt to convert the data, as posted from Internet Explorer, to UNICODE using UTF-8 conversion; but the POST verb doesn't do the proper conversion automatically.

Product Customization

Microsoft CRM is a product that can be customized to a high degree. Depending on viewpoint or customization goals, there are several touch points that might be of interest. For purchasers of Microsoft CRM, the typical product customizations will revolve around creating a customized database schema with the schema manager, surfacing those changes in the application with the forms editor, creating custom business processes with the workflow editor, and integrating with third-party financial and manufacturing systems. For system integrators (value-added resellers, value-added partners, solution developers, and independent software vendors) who are looking to create vertical solutions on top of, or along with, Microsoft CRM, the focus also will be on creating pre-packaged custom versions of the application.

Conclusion

Microsoft CRM is a.NET connected business application. As such, Microsoft CRM makes use of all the concepts normally associated with Microsoft .NET applications: Web service interfaces, Microsoft .NET Framework, multiple abstracted tiers, and the Microsoft server products. Microsoft CRM lends itself well to customizations and extensions by exposing the core business logic as XML Web services. Microsoft CRM is highly customizable in terms of database and user interface extensions. During the development process, security was a primary concern—apparent in the way that the independent layers interact with each other. The product can be easily upgraded and customizations can be moved from site to site.