This article is an in-depth examination of the Notes/Domino architecture, which it presents as a series of layers radiating out from a central core of operating-system-style services -- object store, security, replication, directory services, and more.

**Executive Summary**

How many versions can a software product sustain before its original design is overwhelmed by feature creep? Can a product integrate unexpected technology demands at the kernel level or do new initiatives glue-on, reducing performance and retarding ease-of-use? Can a product offer leadership differentiation where standards do not exist, while it incorporates industry standards? Each of these questions has been asked about Notes. Fundamentally, they are questions about the quality of a system's architecture.

The quality of the Notes software architecture has determined its useful life as a software product. Notes today is both fully mature and readily adaptable to network requirements, whether client-server, intranet or Internet. In this article, we describe why Notes is robust enough to power future generations of development platforms.

When the Notes architecture was designed, no one could have predicted a number of subsequent developments. Most notably, this included the precise sequence of events that exploded into public view as the "Internet." However, disciplined adherence to an operating system-like model resulted in a foundation that easily accommodates Internet-standard protocols and data formats required for interoperability. HTML, Java, SSL, NNTP, IMAP and LDAP are already integrated. As CORBA/IIOP and JavaBeans mature, their integration will further increase development flexibility. (For readers unfamiliar with these Internet terms, please see our glossary sidebar.)

Notes was conceived as a networked environment enabling knowledge workers to create, share and manage their document-centric communications. From observation and experience, its designers knew these processes were different in kind from data-driven clerical work already well supported by transactional systems and relational databases. The integrated, flexible architecture of Domino and Notes at the core services level is joined to a consistent application model that explains the sustainability of the product today.
The architecture was designed from the start around a core of distributed, operating system-style services:

**Object Store**

The replicated object store is the critical repository for managing rich documents, using a well defined set of container classes. All other services rely on it.

**Security**

Because security features have been entirely integrated into the object store, each platform object (from servers through databases, documents and fields) receives full authentication and access protection. Each user is both the recipient of assigned rights as well as the controller of rights that fall within their scope.

**Replication**

Administrators schedule uni- or bi-directional replication between any object (field to server) on computing devices of any scale (PDA to mainframe). Synchronization provides near real-time assurance of a consistent system image across the enterprise. Locally-connected and mobile users employ the same replication services to manage their own databases and documents.

**Directory Services**

Look-ups to Public and Private Address Books control access, application database and mail traffic flow across Notes networks. Architecturally, the Address Book is itself a standard Notes application. Because it inherits the platform's consistent application model, new Notes applications can plug in their specific configuration and control requirements to the Address Book without disruption to the underlying platform architecture.

**Integrated Messaging and User Applications**

Mail, discussion group, calendar and resource management applications rely on routing services in the kernel to share messages across the platform. The supplied applications are template driven and can be freely extended or re-combined by users to meet task-driven, workflow requirements. Architectural integration yields the clearest user productivity benefit here, since storage, routing, editing, rendering and security management are modeless. Applications inherit the entire set of core Notes services without any need for clumsy integration or user-programmed inter-process communication.

**Integrated Development Environment**

Complete core classes are offered across several cooperating tools and facilities: a formula language for end-users enables manipulation of a session's active object behaviors; LotusScript supports modification of existing system classes and creation of new, inheritance-capable classes; a C-language API gives access to internals. Complete commitment to Java support includes the ability to schedule execution of Java agents on Domino servers as well as the use of Java applets to represent the Notes UI to browser clients. The visual development environment is fully integrated into the platform's form-driven design model.
Because architectural services are exposed by design, a given platform component can be treated logically as a service or as an application. For example, the Public Address Book is, strictly speaking, an application built on top of Notes kernel services. However, third-party applications rely on it as a directory service. Barriers between user interface, applications and core services are highly permeable, enabling developers to make their own decisions about which Notes components will best support their own specialized applications.

As Internet competitors seek to become Notes, Notes continues to adopt open protocols and innovate on a tested, scaleable architecture that was integrated from its inception. It is more difficult to develop integrated services to support Internet-driven applications than to adopt new protocols that map to well tested, robust service layers. By making it possible for knowledge workers as well as programmers to build applications appropriate for team activities, without forcing reliance on hard-core coding, Notes changes the equation between programmers and users, between custom applications and reusable templates.

The rest of this article describes why the guiding design principles and architectural attributes of Notes position it for continued leadership over the coming decade. For definitions of the terms used in this article, please see our glossary sidebar.

**Mapping the architecture**

The Notes architecture defines a set of layers radiating outward from a central kernel (Layers 0 and 1 in the following diagram) to end-user developed applications at the periphery (Layers 2 and 3). A key distinction is that the layers are isolated and opaque to each other.

**Figure 1. Notes architecture diagram**

This layered view underscores the tight integration of core services and applications. It contrasts sharply with other products that were assembled from disparate sources and patched together with ad hoc interfaces.

In this and the sections that follow, we outline a few specific examples of services at different layers of the architecture to illustrate the design.
Layer 0: The Kernel

Layer 0 contains the document-centric object store. Each document, or 'note', is a compound structure of mixed data types arranged in fields that can be arbitrarily modified and extended. A note may contain text, rich text, binary blobs (attachments, ActiveX or Java applets, for instance), encryption keys, doclinks, and so on. Each Notes database contains a collection of notes, and includes meta-organizing structures for display, security, retrieval and access rights to the notes.

Layer 1: Kernel Extensions

Layer 1 is logically distinct from Layer 0, but is tightly bound to the object store. Critical platform services are encapsulated within this layer. These include security, replication, routing, compute (the integral function interpreter), view indexing, and document editing/rendering.

Each Layer 1 module is self-contained, with common interfaces to each other and the object store. As a result, new modules can be introduced into Layer 1 or existing ones swapped out. For instance, the SMTP Message Transfer Agent (the router for Internet mail) was inserted in Release 4 to provide an alternative mail router. The same ability can and will be utilized for search engines, security keyrings (Public Key Infrastructures), and so on, as the needs of customers require the adoption of new or standards-based components.

Layer 2: System and Core Applications

Layer 2 includes core collaborative applications that are coordinated by supporting system applications. Core applications (mail, Calendar and Scheduling, discussions, document management, Web Navigator, and so on, are the first level of services directly visible to end-users. As standard Notes applications, they can be modified and extended by Notes developers, but usually have a dependency on kernel routines. System applications control and monitor these and all other applications. They include the LDAP-compliant directory service (Public Address Book), logging, administration, billing, free time and others.

Layer 3: Layered End-User Applications

Layer 3 applications are not distinguishable technically from Layer 2 applications. Layer 3 applications also rely on the kernel and the Public Address Book. However, most of them further extend core mail, Calendar and Scheduling, or discussion features into higher level, task-specific applications. The growing set of Domino.Applications exemplifies the way that complex function evolves from lower architectural layers, while adding design and user interface elements of their own. Domino.Action, for instance, draws on security, workflow, document management and discussion threading to enable end-users to specify and generate Web sites. Thousands of existing third-party Notes applications also fit into this layer.

OS Optimizations and Programmability

Two orthogonal "slices" traverse all architectural layers. The first slice reflects the optimizations required to connect the platform-independent architecture to specific operating systems. These include file system, memory management and process/thread tuning (most notably for OS/2,
Windows 95/NT and UNIX), support for numerous network protocols (TCP/IP, SPX, NetBeui, and others) and customization of user interfaces (Windows/NT, Macintosh, OS/2, X/Windows).

The second slice offers a suite of programming tools giving low- to high-level access to the entire platform. This ranges from access to API internals (excluding those essential to maintaining platform integrity); a scripting language with visibility into kernel classes and support for user-defined classes (LotusScript); and a macro formula language for quick, high-level customization. Important third-party products and tools, both for application development (Visual Basic), Web design and site management (NetObjects' Fusion), and system development (Java and JavaBeans) are also supported.

An Architecture for Application Development

The architecture balances rigorous design controls, implementation discipline and service modularity against the objective to make that design visible to servers, applications, programmers and end-users. The combination of integration and exposure means that a particular element (for instance, mail) can be viewed from one perspective as a self-contained application; from another, as a service upon which still more sophisticated applications can be layered. Competent non-programmers can copy, clone or extend database templates to evolve their own applications.

Because services and applications are inter-woven, Notes has been variously described as a distributed database, a document manager, a bulletin board system, a groupware enabler and more. It can be more precisely defined as an integrated architecture exposing programmable objects from which arbitrarily complex applications can be developed.

This is programming by exception and inspection, not by default. While low-level coding is enabled, object classes are provided at a high level of abstraction. Usually, most of the required application function is already available. Design elements needed (forms, views, scripts, and agents) can be layered on or integrated into the existing application for immediate execution without compilation.

Layer 0: The Object Store

The essence of Notes is its Notes Storage Facility (NSF), a centralized object store. NSF is a database engine, designed from its inception to support distributed and asynchronous collaborative work. Several insights guided its development:

- Knowledge workers output rich-text documents replete with contextual links (versions, references, and so on) to other similar documents.
- Knowledge work is highly distributed geographically.
- Collaboration is most productive when users can choose their own timing (that is, work asynchronously).

These insights drove a set of inter-related requirements:

- Large numbers of documents must be managed.
- Versioning and hierarchical relationships between documents must be system-maintained.
- Document synchronization is critical.
• Document properties must be self-contained and extensible.

NSF is critically distinguished by an assumption that has been amply validated by Notes usage: human collaboration does not require, and indeed, in many cases, is fundamentally incompatible with a consistent, distributed data dictionary. A loose coupling of document versions is necessary for enabling distributed work, providing that multiple versions and version conflicts can be flagged and resolved. This original insight allowed for the development of a high-performance object database with integral replication capabilities.

Relational databases were carefully considered as a possible implementation solution, but a custom design was judged more suitable. The RDBMSs are tuned for high performance atomic transactions and multi-table join operations. While they may be leveraged for Notes in the future, they are not typically targeted for collaboration where persistent compound documents with private behaviors need to communicate through defined and flexible interfaces. Notes can be linked conveniently to relational and object-relational database systems to support transaction-driven application behavior.

**Compound Document Model**

Compound documents ('notes') are the base objects in NSF, consisting of an extensible set of structured data organized in a fielded containment hierarchy. All structures are exposed for programmatic control through object-oriented languages such as LotusScript, Java or C++.

The hierarchy includes a note ID (a compact, unique identifier with date/time and replication information) and a series of fields. The fields are typed and can be text, numeric, date/time, keyword, names, reader, author or rich text. Rich text fields are segmented by text and formatting records (including provision for binary runs of attachments or external objects such as ActiveX documents). Tagged HTML/SGML text is incorporated as a rich-text class and can easily be extended.

Control information is maintained by each note within meta-data fields. Persistent security attributes (document signatures and encryption keys), access control information (authors and roles), replication history, and so on, are kept on a per-document basis. A fast, flexible *On Disk Structure* (ODS), a normalized representation of the data, provides a consistent, cross-platform method for storing documents. This relieves developers of otherwise burdensome requirements to normalize and convert rich, collaborative content.

Notes' object identification schema is not based on relative disk location, hash algorithms or attribute-based identifiers but, instead, on the reasonable assumption that there may be multiple replicants of a database that are only occasionally connected to or between servers. The identifier travels with its containing object, bringing its own history with it.

**Notes Application Model**

The database is the second level of containment. It instantiates the collection of documents and control information as a file. The database contains meta-data similar to that maintained for individual notes, but scoped across an entire document collection. It also contains forms, views
and other special objects (subforms, design elements, agents, shared script libraries, help and About documents, shared fields, graphical navigators, and so on).

The Notes application model, relying on the database, utilizes templates that contain all design elements, security information and computational logic. They are, in essence, the application without the data. Notes developers utilize the standard sample templates to start building their applications, and frequently cut and paste design elements (forms, views, fields, scripts, and so on) to assemble the finished application.

A Notes form (and the extension, subform) provides an arbitrary, flexible method of representing a Notes document. Many different forms can be applied to a document and the form can be stored with the document, if desired. The form or subform representation may be defined conditionally, based on field values.

For example, a purchasing request form for the end-user may be very simple, with fields provided only for the requested item, quantity, totals, approvals, and so on. When the same document is viewed by the purchasing clerk, it may have fields displayed for vendors, cost center, management approval, and so on. Once approved, the form may be converted to a purchase order and then submitted to the vendor. These different renderings of the same document are achieved by conditionally displaying fields or replacing the form itself based on the viewer's role in the application. This is a fairly simple technique in Notes application development.

This built-in mechanism gives great freedom to application designers. Web site designers, in particular, gain great flexibility for creating dynamic documents easily. This unique "late-binding" model of presenting information is supported on the Notes client and on Web browsers through the Domino HTTP service.

A Notes view is a list of documents within (or across multiple) Notes databases. Like the result set of a traditional database query, the list utilizes selection formulae to select the set. Views provide a simple universal method for selecting, searching, categorizing, sorting and, finally, presenting documents. The view collection, utilizing summary meta-data of each document, is optimized for page-oriented transmission and display. Presentation and manipulation are flexible, allowing efficient sort-on-the-fly and ad-hoc view query. Full-text search indexing and result presentation have been integrated into the view mechanisms, providing relevance ranked results. Applications can constrain searches to single databases, multiple databases or across entire sites, reducing the incidence of irrelevant results.

Unlike a typical retrieved query, a view is actionable. It not only provides a means to open documents but to perform flexible actions upon their properties such as dragging and dropping into folders, using the view as input to any arbitrary script, importing and exporting, and so on. Views have also been extended to support new UI metaphors, including calendar views (where the layout can be by day, week or month) and folders (a view subset to facilitate drag-and-drop document categorization).

Notes views represent an excellent demonstration of the power of architecture integration. You can easily search, select and copy rich text documents from one application context to another (say,
from a discussion database to a group's project calendar). This is very difficult or impossible when users are forced to cobble together the connections from disparate applications.

Layer 1: Security

When Notes was first designed, it became clear that collaborative work dictated security far beyond simple password access schemes. While RSA public/private-key cryptography was already defined, it was of interest only to security professionals in the military and financial sectors.

An early RSA adopter, Notes exerted leadership in document and access security just recently understood as a result of widespread exploitation of the public Internet. Data encryption with a large cryptographic key length is the only mechanism capable of protecting sensitive data now and in the immediate future.

Notes' security support falls into four general categories: user authentication, channel encryption, document encryption/signature and execution control.

User Authentication

The first element of a secure system, and the most critical, is the guarantee that a user is who he or she says they are. Notes uses the RSA Public/Private Key cryptosystem to challenge user and server authenticity before access is allowed. Each user has a unique ID file that is password protected and contains the private key, a collection of hierarchical identification certificates and a collection of encryption keys. The login mechanism is protected from brute force dictionary attacks and has been integrated with other similar OS login systems (Windows NT, for example) for user convenience. The result of this authentication protocol is assured identity on both sides.

When Notes usage expanded across entire enterprises, managing name spaces became more complex, both for security and directories (the Address Book). Guaranteeing name uniqueness as well as trust led to development of a hierarchical naming system, based on X.500, to support certificate and key exchange between servers. A typical fully qualified name is in the format of common name/user organization/organization (for example, John Doe/Cambridge/Lotus). Notes incorporates the ability to generate user, server and certifier ID files, and to generate a cross-certificate capable of bridging trusted elements.

As the Internet X.509 certificate implementation has become standardized and interoperable, the Notes schema treats it as an integrated certificate in the Notes ID for Web access. In the future, this will become the standard format for Notes ID certificates.

Channel Encryption

Since Notes was designed as a distributed client-server system, it was important to implement protection against a network-based "replay attack" (where an intruder captures network data packets and re-sends them to gain system-wide access). Notes incorporates both a proprietary packet encryption scheme and the standard SSL packet encryption (for connection to non-Notes servers) to secure the communications channel between client and server. This encryption is another layered service that works across all communication channel types, not just TCP/IP.
One interesting extension to this security allows users to pass-through a server to access another server. The pass-through path traverses many servers and different protocols. At each pass-through stop, the user is authenticated and server security restrictions are checked before the session is established. Consequently, Domino servers may serve as secure application-level firewalls to the Internet, in addition to providing a flexible and efficient method of centralizing connections for dial-up clustering or security monitoring.

For example, an organization can implement a Notes server on the Internet with one network adapter using TCP/IP network protocols and another adapter connected to the internal (secure) network using SPX protocols. From the internal network, Notes users can pass-through this server to get to Internet-based resources (other Notes servers, Web sites, and so on). Hackers have a difficult task breaking into the internal network. TCP/IP utility hacks will not work on SPX, while breaking through Notes security (and RSA authentication) is prohibitively resource intensive.

**Document Encryption and Signature Verification**

Public-key encryption can be applied to any database or document. Public/Private key pairs are generated by Notes when registering new users. The public key is stored in the Public Address Book and is used to encrypt sensitive material. A private key is also issued, but is kept secret in the user's ID file. When someone wants to send a secure message, they encrypt it with the public key they received from the intended recipient of the message. Then, once the message has been sent, the recipient uses their private key to decrypt the transmission.

Beyond mail encryption, any document can be further encrypted with a private (but shared) document encryption key, stored in the user's ID file. Once generated and distributed by Notes, the usage of document keys are applied transparently.

Notes documents can also be digitally signed, assuring the recipient that the document has not been tampered with during transmission. Multiple signatures can be stored within a single document. Digital signatures can be applied to sections of documents as well. Citing one instance, a workflow application, such as a purchasing requisition, can bear an electronic signature and date stamp that is irrefutable. In Notes, the process is as simple as activating a check box and saving/sending the document. With most intranet applications, you typically need to run the message through an external encryption or signature program (for example, PGP) with required management of an external directory of certificates and keys.

**Execution Control**

Today's dynamic documents make it necessary to control the execution of macro formulas, embedded scripts or active objects more precisely. By contrast with the coarse control available through Java virtual machines, Notes supports fine-grained control of executable content using all the naming, access control and certificate verification elements discussed above. This allows the user to precisely determine what action a signed (or unsigned) document can perform on their local machine, such as write to the file system, manipulate in-memory structures, read/write to other Notes documents, and so on. Notes goes well beyond the simple "Do you trust the creator of this application?" to "Should this mail message be permitted to execute a program and write to your hard disk?"
Access Control

Notes (or any other product's) security capabilities are moot without a structured system for deploying them. Application developers can design a flexible, multi-layered security system. Access rights determine who may view a database, document, or field and what they are entitled to do with it. At the system level, the server's Deny Access and Allow Access lists (an Address Book application service) are the primary gatekeepers to the server itself.

Each database sourced or replicated on a server also maintains its own access control lists (ACL). This further defines both who may access a database and what level of access is permitted (for example, a Manager role for a database or a Reader role for a document). Documents or views contained within a database also define access rights for granularity of control at the document or view level. In addition to access rights, users can be assigned roles that allow still finer-grained access to field-level elements. Like rights, roles can be easily implemented by group lists.

Usability and Interoperability

Notes provides unparalleled levels of security and usability due to its architectural integration. Today's requirements for unobtrusive access to Web-based material are met through (controlled) anonymous access options for both Web and native Notes clients.

As Internet security protocols become accepted and widely implemented, Notes will incorporate them. Secure Socket Layer (SSL), LDAP directory (for public key distribution) and Digital IDs (X.509 formatted certificates and key pairs) are already incorporated. More are sure to come, and Notes will continue to deliver a superset of this functionality.

User convenience has historically been factored into any secure application, and Notes has the flexibility to meet the needs of application designers. Notes security may range from none specified to simple username/password schemes to Internet-level SSL security (with or without Digital IDs) to the complete suite of Notes security mechanisms (user authentication combined with channel encryption, document encryption/signatures and complete auditing logs).

Layer 1: Enterprise-scale replication

Secure, efficient synchronization of Notes databases is a critical architectural service. The object store was designed for peerless, masterless replication as opposed to disk-based replication. Documents "know" that others have a copy of them and can exchange information on an as-replicated, cooperative basis.

By contrast, log-based replication must implement clumsy top-down replication chains that are inflexible and scale poorly. The use of a single, consistent replication model across the platform, rather than multiple, semi-compatible implementations across components, provides the needed leverage for sizing replication to entire enterprises.

Replication allows the mobile user to "take his/her office with them." It allows an enterprise developer to distribute application updates globally to all users overnight. It propagates employee "moves-and-changes" information (with the critical security information) almost instantly.
Replication makes it possible and convenient for a corporation to smoothly scale their Web site performance and reliability by using Domino servers in a cluster configuration.

**Defining Replication**

Replication synchronizes Notes elements (databases, design elements, documents or fields) between servers or between client and server. This is further refined by one or more conditions governing the synchronization event (scheduling, authentication, access rights, roles, scripted events and more). Each replicable object maintains its own replication history, enabling efficient incremental replication (through system analysis of the history) as well as performance checks and troubleshooting by administrators.

While databases or documents can be copied, a copy operation decouples them into two, discrete objects with their own subsequent, separate, transaction history. In replication, an object shares the same ID with its source object. The sharable ID becomes the basis for tracking and applying changes made to the object, as well as with "n" number of other progeny (its replicas) across the enterprise. Replicas can be synchronized with any other replica that shares the same ID.

Layer 1 security services can fully protect the integrity of replicated data. Beyond this, ACL rules already specified for design and use of a given object are applied automatically to each object replicated. Like other architectural services, replication is exposed programmatically. Notes replication is uniquely granular to the field level, permitting efficient synchronization of large directories and workflow databases (characterized by large numbers of small changes). Notes views can be utilized to select a data subset to replicate (for instance, restricting e-mail replication to documents within the Inbox).

**Managing Replication In-The-Large**

For administrators, replication is an integral component of network and server management. Varied network topologies (hub-and-spoke, end-to-end, ring or mesh) can be used independently or in combination to suit replication routes and priorities. Replication can be uni- or bi-directional. Each replication connection can be assigned a schedule or can be executed on programmatic demand.

A special variation of database replication is supported within a Notes server cluster. Real-time replication occurs whenever changes to an object are detected. Server clustering enhances availability, provides failover services and passes replication changes automatically to servers within the cluster group. If a cluster server fails, a database request is redirected to another server within the cluster that contains the replicated database. Administrators can define the trade-off between the number of databases required within a cluster to ensure availability against the overhead of the replication process.

Because simultaneous document editing is common, documents are not automatically locked. A schema for managing save or replication conflicts (where a document has been changed by more than one user between two replication events) is managed by a response document model. Replication conflicts can be programmatically resolved along a set of established rules or criteria. These simple, user-definable rules determine which becomes the parent, and which the sibling
documents. Document chains can be synchronized later by administrators, users or through application scripting.

**Selective Replication On The Ground**

 Appropriately, local replication settings cannot over-ride replication rules at the server level. However, local and mobile users own their personal databases and can tune their replication logic with the same tools available for the enterprise.

Selective replication is particularly useful. Users can conveniently pick a pre-composed database folder or view (that is, a selection of documents) and assign that for replication. Formulas can restrict field-level replication when a view is unavailable. The ability to replicate partial documents (summaries) as well as to replicate documents by date further filters databases into the objects of immediate interest.

For programmers, selective replication enables very convenient transmission of controlled design elements. When a database template changes, database instances are changed automatically, regardless of their location. No rigid master schema is required nor must multiple versions of binaries be managed. If a business rule represented in a master copy of a database is changed, all copies will be changed automatically during a replication event.

Since available bandwidth varies according to traffic peaks, application loads and evolving topologies, the ability to combine replication options makes for highly sensitive tuning to the available bandwidth between servers and/or between servers and clients.

**Layer 2: The Notes directory**

The Notes directory (defined as the Public Address Book or "PAB" for short) occupies a unique place within the architecture. Each named domain has a replicated PAB database on each server. As a centralized directory service, it controls the name space (users, groups, mail domains), public encryption keys, Connection documents (mail routing and replication scheduling), certificates for the named domain, configuration and restrictions on servers, and server extensions (such as MTAs and gateways). Each Notes client also has a private PAB subset with name, cross-certificate, connection and location information.

The PAB concept and structure was modeled on the original X.500 and X.400 directory/mail requirements, with a hierarchical and extensible schema. It is also, however, just another Notes database, which places it above the core of Notes (layers 0 and 1) and alongside the system applications on Layer 2.

**Administering Address Books**

A Master Address Book can share Public Address books across domains, using rules for resolving name conflicts and providing a cascading lookup capability across domain name spaces. Public Address books are generally hosted on hub servers for frequent, rapid replications to server spokes. Optimizing Address Book replication enhances system responsiveness, since other applications access the PAB continually across the entire enterprise.
As the gatekeeper application, Address Book administration ties tightly to security management. Certifier documents control the certification chain, using hierarchical naming support. Additional database creation and modification roles have been introduced so that Address Book documents and fields can be managed collaboratively between administrators with varied authority and task responsibilities.

**Making The Personal Connection**

The Personal replica contains a subset of the Public Address Book’s views and documents describing co-workers and connections to servers appropriate to their work. From this, users customize their Personal Address Book to include companies and additional persons. This is not restricted to Notes users, making the Address Book one component of a personal information manager (PIM).

A default location set is also defined (for instance, office, Internet, home, and so on). These location documents specify ports, protocols, communication devices and phone numbers for reaching a given server. Server connection documents describe a chosen server. The combination of person, location and server connection documents creates a mapping between a Personal Address Book and the specific Public Address Book that can resolve (that is, find) the person or machine resource. This may include referencing the appropriate gateway for resolving an Internet address.

**The Platform's World-View**

The PAB serves as a good model of Notes extensibility and adaptability. Consider the Lightweight Directory Access Protocol (LDAP) support that was recently added to the Domino server. LDAP is a subset of the original X.500. Its requirements for a rigid, secure schema and open extensibility were easily handled by the PAB. Adding LDAP makes it possible for sites to conveniently support Notes-only and mixed Notes-Internet environments using a single Notes toolset.

The Address Book illustrates both how robust the original Notes architecture has been at supporting unexpected demands for enhancement as well as the advantages of a consistent application model. New applications can add their own system document types to the Address Book database. Although customization is best reserved for special situations by knowledgeable users, the Address Book itself can be modified and extended.

**Layer 2: Messaging as an application**

Electronic mail in Notes (Notes Mail) has driven the rapid expansion of Notes deployment worldwide. Technically, however, mail is just another Notes application, although it has produced some core extensions and, for this reason, fits within Layer 2 of the architectural schema. We outline below mail extensions that illustrate the robustness of the base architecture.

These extensions were driven by the requirement for a router function to expedite the efficient transfer of messages between servers. Notes has implemented a multi-threaded/multi-process
router with the intelligence needed to select least-cost routing for both the original dial-up X.25 and IP Internet connectivity. The router can be configured to accompany replication traffic or can bypass it based on a preset threshold or message urgency.

Shared mail services (single-copy message store) store identical server copies of a multi-recipient mail document. Mail headers are shipped to users, with a link maintained to the shared server file. A user can delete a mail message (actually, the header), without deleting the shared copy. Special security rules arbitrate content sharing.

To incorporate the user interface of the popular Lotus cc:Mail program into Notes, folders (a variant of Notes views) are included in the core product. Folders enable direct drag-and-drop manipulation of documents into storage hierarchies. As discussed previously, calendar views were also added to the product to support user schedules. In addition, a "free time" system was introduced to support the lookup of another user's calendar or a resource (a conference room, projector, and so on).

And finally, Notes mail services have also been extended to include Message Transfer Agents (MTAs) for SMTP and X.400 mail systems. These MTAs reside alongside the Notes Router function and deliver messages using alternative protocols. You could, for instance, utilize SMTP for all server-to-server routing in a Notes Mail system.

These extensions not only made Notes Mail a world-class messaging system, but, due to the integrated nature of the platform, also propagated the same benefits to all Notes applications. Calendar views can be used to monitor and schedule group events or product schedules. Message routing can be used in a purchasing workflow application to remind managers of pending approval requests. Free-time lookups can enhance project management applications. The Web Navigator application uses messaging to deliver the user's personal news summary.

These extensions offer a few examples of leveraging the messaging infrastructure. The reverse opportunity also exists. As a standard Notes application, Notes Mail can be (and has been) extended by the user or corporate developer to contain custom applications. For more information on the possibilities, see the "Collaboration in Daily Work" sidebar.

An open toolkit

Learning from Lotus 1-2-3, the developers of Notes knew they were creating a tool as much as an application. A strong commitment has been kept to make every action and event scriptable with @Command-style macros. As Notes matured, a C/C++ API and the LotusScript language were integrated. An end-user agent builder interface was added to give a convenient method for executing menu or schedule-driven actions. Today, Java classes are being incorporated to give Notes the broadest possible range of programming interfaces.

API Access

Core internal routines can be accessed directly through an object-based, C++ language API. Services, including the object store, security, replication, routing, indexing, editing, and rendering can be called directly to support standalone application development. The API interface is most
typically used to develop back-end server applications and is, by nature, platform specific. An Extension Manager routine traps internal back-end calls and redirects them to the add-in control.

Two Internet-oriented developments are extending these existing APIs. The first consists of a series of Java-based back-end classes that open Java applet access to Domino server functions over HTTP. The second is a standard CORBA/IIOP implementation affording direct back-end access to all server objects from any IIOP client (for instance, a browser).

Script-Level Access

Lotus Development designed LotusScript as a cross-product, cross-platform object-oriented scripting language. LotusScript is a Visual Basic clone interpreter immediately familiar to experienced VB programmers.

Notes distinguishes two primary class categories: back-end classes and user interface (UI) classes. LotusScript Extension (LSX) classes form a third category (for example, extension classes offer ODBC support). UI classes make the session properties and the currently active object visible to an application. Back-end classes expose kernel service objects (sessions, databases, forms, views, documents, fields, ACLs, and so on). While many programmers need only manipulate the methods or properties of system-defined classes, programmer-defined, single-inheritance path classes are also supported.

Most actions that can be taken by administrators or users through the user interface (server, routing and replication management; database, form, view and document management) can also be scripted. In addition, numerous properties not normally visible to the UI are also exposed (for example, a script can specify an action for triggering failover after server loss). The @Command and LotusScript interfaces come together in an Integrated Development Environment (IDE) complete with a debugger, a class browser, a real time variable display and other facilities. JavaScript is supported for Web-oriented applications.

Formulas

The integrated formula language was Notes’ first programming tool. It gives simple scripting access to most menu commands and back-end routines. Formulas are typically used to build simple (but powerful) multi-step actions for repetitive or scheduled back-end functions (for instance, change all documents with value ‘X’ in the subject field to value ‘Y’, or file all incoming mail from user ‘B’ in folder ‘B’ in my mail file).

The simple paradigm of document forms and views within a database container in Notes provides the structure needed for a rapid application development environment. A wide array of application services are available programatically, including security, naming, object store, replication, rich-text rendering, full-text indexing, view filtering and lookup, editing, and so on. More generalized memory management, file caching and distribution services are also exposed.

One of the more powerful advantages of Notes is the use of the object store to distribute, control and maintain applications. The Notes database contains all the design and logical elements of an application, providing the ability to prototype an application locally and replicate it globally when
complete. Additionally, with the exception of API programs, which are by nature external and therefore OS-dependent, all Notes applications run on all supported Notes platforms, ranging from Windows 3.1 to high-end Unix MP servers.

**Exposing Development Options To The Programmer**

The effect of these overlapping but distinguishable toolsets is to expose most internal Notes functions as application development options for programmers. Moreover, the consistent application model ensures that the platform’s own onboard applications (Address Book, mail, discussion, Calendar and Scheduling, and others) are no different in kind than end-user applications, minimizing technology transfer and training costs.

Once assigned Designer access, programmers can modify these onboard applications as freely as their own. Or, they may cut-and-paste code from them into their own scripts, but use them for entirely different application types.

While many sophisticated applications have been developed in Notes, developers retain ample opportunity to push the functional envelope in elastic and, as yet, unforeseen directions. The architecture remains an enabling, rather than limiting, factor. One of the most promising indicators of the architecture’s future sustainability is the still unexplored limits of the toolkit.

**Adapting to the Internet**

No vendor explicitly anticipated the explosive growth of the Internet. However, the characteristics of the Internet that make it ideal for collaborative work (networks, platform independence, ubiquitous mail, document-centered presentation) have long been familiar to Notes users.

In a real sense, the Internet has validated the original Notes vision. Notes was developed before the emergence (or acceptance, in the case of TCP/IP) of many of the industry standards we assume today. Such critical Internet components as HTML, HTTP, IIOP, Java, POP3, and so on, had not been invented when Notes first shipped. Other protocols like NNTP, TCP/IP, DNS, SMTP, were not in widespread corporate use.

**Service Integration**

Much of what users expect the Internet to do eventually (for instance, security and asynchronous replication) is available within Notes-centric networks today. At the same time, it is important for Notes to interoperate natively with other platforms and software systems. This is especially critical in cases of widespread consensus on shared data or transport protocols.

In 1994, Lotus began development and integration of Internet protocols into Notes. This started with Web and news protocols, moving from there to messaging and the base data formats themselves. The integration process began with the addition of gateway servers and then embedded the service itself with Notes.

The evolution of the Web Navigator demonstrates platform flexibility. The Navigator was first delivered as a server add-on to provide a workgroup with controlled access to the Web (and to centralize the limited corporate Internet bandwidth). The next release integrated the functionality
into the Notes client, but kept the collaborative features of the server. The latest release embeds Microsoft's OLE (aka ActiveX) browser component within the Notes UI to make Internet Explorer's advanced rendering available. At the same time, powerful collaborative Notes features, such as forwarding a Web page via Notes Mail with all rendering and links intact, are retained.

HTML data formats demonstrate another relevant integration case. Starting in Notes 4.0, an HTML datatype was introduced to provide "raw", unaltered storage of HTML documents. In Notes 4.5, a mechanism was introduced to store and reference graphics individually, with progressive rendering of graphics in Web and Notes documents. In Notes 5.0, the HTML document model is fully assimilated. Notes supports native HTML rendering and URL addressing of each document element. This flexibility gives customers the option of using either document model (contained or referenced) and provides for greater interoperability and mobility of data.

Adaptability for the User

While the demonstrated adaptability of the architecture to the Internet is business-critical, adaptability must always have real users in view. Since the Internet is still in a state of flux (a condition likely to persist) and some of its capabilities are relatively primitive (notably security), the years invested in delivering enterprise-strength capabilities for Notes customers remain applicable.

In situations where "proprietary" means "better" and "standards" mean "lowest common denominator functionality" (or standards are not in place), Notes users benefit. Meanwhile, Notes implements each emerging Internet standard, usually in forms complementary to current Notes usage (as with security). The evolution toward platform-wide Java support shows commitment to an important technology, while support for LotusScript and other programming tools also evolves.

As always, the Notes philosophy is, "provide the tools and let the users decide."

Notes: A framework for interoperable, collaborative applications

Collaboration and teamwork are a strategic objective reached as the result of many specific solutions that work together smoothly: mail, calendar, document creation and management, and others. The components and applications shipped with Notes (Address Book, Notes Mail Discussion Threads, Calendar and Scheduling, ToDo) offer functionally complete solutions to the narrow problem domains that make up the actual bricks and mortar of collaborative work.

Communication and collaboration revolve around the creation, management and exchange of unstructured data at various points of time and space. So, on the one hand, the Notes architecture refused to make limiting assumptions about time, space and platform. On the other hand, precisely in order to make human knowledge management possible, a system must maintain as much semantically retrievable data as possible.

Well-designed software balances open assumptions with logical rigor, responding to a user's requirements unobtrusively on an as-desired basis. This principle of "self-disclosure" is especially germane to intranet and Internet applications.
The compound document meta-model of Domino and Notes structures applications, forms, views, documents and links and, more pertinently, tracks their state. Consequently, platform-wide messaging and other services are not bolted-on, but are integral. While similar document models are migrating slowly to the Web, current and near-term Web servers "see" links as flat, while the Domino Web server sees a link as an element in a richly structured set of applications and documents.

While the sustainability of the Notes architecture is key to mission-critical corporate investments, it is the way that the architecture is implemented that secures the allegiance of developers and users.

For developers, Notes delivers a reusable, extensible framework for building fully secure, networked and scaleable applications at any desired level of complexity. Using Notes, task-driven collaborative applications can be designed that are greater than the sum of the parts that compose them. It is the applications themselves that make Notes more useful today than ever before.

The combination of a mature architecture with a rich feature set that implements platform-independence and interoperability based on full compliance with industry standards is demonstrably unique. Notes is the ideal candidate for your mission-critical collaborative computing infrastructure.

**Copyright**

1997 Iris Associates, Inc. All rights reserved.
Related topics

- Collaboration in Daily Work sidebar
- glossary sidebar
- Notes software architecture

© Copyright IBM Corporation 1997
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)