



Solid Distributed Data Management Platform

A Solid White Paper



Table of Contents

New Data Management Challenges 3

Distributed Data Management..... 4

Solid FlowEngine™ 5

 Embedded Data Management 6

 Distributed Data Management..... 8

 Advanced Bi-Directional Replication 9

FlowEngine Applications..... 10

 Solid FlowEngine in the Network Infrastructure..... 10

 Telematics..... 12

 Example: Buscom 14

 Distributed enterprise 14

Summary 14

New Data Management Challenges

Over the last 20 – 30 years, the most popular technique for managing data in the enterprise has been the central Relational Database Management System (RDBMS). RDBMS like Oracle and IBM DB/2 have been enhanced over hundreds of engineer-years to create data managers that are superbly suited to managing enterprise data for both application packages like Customer Relationship Management (CRM) and Enterprise Resource Planning (ERP), and also custom applications designed to run mission-critical applications within the enterprise.

However, during this 20-30 year period, computing has not stood still. When RDBMS first arrived on the scene, they were applied to departmental-class applications delivered to an end user via a purpose built application written in either a general purpose programming language like C, or a screen-based Fourth Generation Language (4GL). Now applications link the entire value chain and are delivered in HTML to portable devices, the Internet has become the preferred medium for all kinds of business interaction, leading to an explosion of innovation and capital investment in the network infrastructure, and computing is becoming pervasive, with increasingly powerful devices distributed to vehicles, buildings and other remote and mobile machines in order to manage and coordinate their activities.

Through this sea change in computing, a new class of applications has arisen that is poorly served by the central RDBMS. These applications are characterized by running on a large number of computers that each act autonomously, but in a coordinated way, to solve a business problem. The devices are “loosely coupled,” which means that they may not always be connected to the network, but they are some of the time. However, they need to function correctly and efficiently, even when disconnected, so they cannot rely on a remote data source for their working knowledge. Such applications appear in a variety of industries; notable they provide the infrastructure of both the public switched telephone network (PSTN) and the Internet, so they need to run with a very high degree of uptime (often referred to as “five-nines,” meaning 99.999% availability). And yet they run on devices that often go for long periods without human attention, and in configurations where, due to the sheer number of components, something is always failing. This kind of distributed application also runs mission critical processes in high-end cars, that typically incorporate tens of computers and several local area networks (LANs). They are used to manage building environments, monitor security, run factories, manage vehicle fleets. Increasingly, the modern distributed enterprise is also reframing itself in these terms: as a network of loosely connected, cooperating, autonomous entities.

In a sense this change may be seen as the part of the continual evolution of a basic data management design pattern, with a trend to delegating increasing authority to the edge. In the simplest form, data management is part of a monolithic application. This is still appropriate for some small scale, single-user, isolated applications. Client/server pulled the data management out of the application and delegated it to a specialized database management system.

N-tier, Web-based computing elaborated on the client/server design pattern by adding a connection and interface management layer between the database server and the client. Pervasive computing deals with massively distributed applications, where connectivity to the shared data manager may not always be available or economical. In this model, data management is distributed throughout the nodes of the application, and synchronization takes care of maintaining the global consistency of the data.

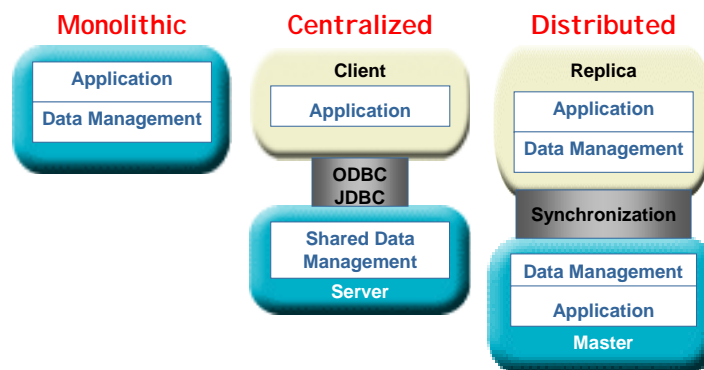


Figure 1: Evolving data management design patterns

Distributed Data Management

Highly distributed applications have data management requirements very different to those of the centralized corporate applications that drove the development of the central RDBMS.

The central RDBMS was designed for a very tightly controlled environment. It provides large numbers of tuning parameters so that it can be continually adjusted to maintain optimum throughput of the entire application suite as workloads shift. A cadre of Database Administrators (DBAs) has arisen to take charge of the day-to-day management of the central database, and have come to contribute significantly to the total cost of ownership (TCO). Because it is suited to everything from high-speed on-line transaction processing to complex data mining, the central RDBMS has become enormously complex, and demands significant resource to function properly. These data managers are available only on the enterprise operating systems: mainframes, enterprise Unix, and Windows.

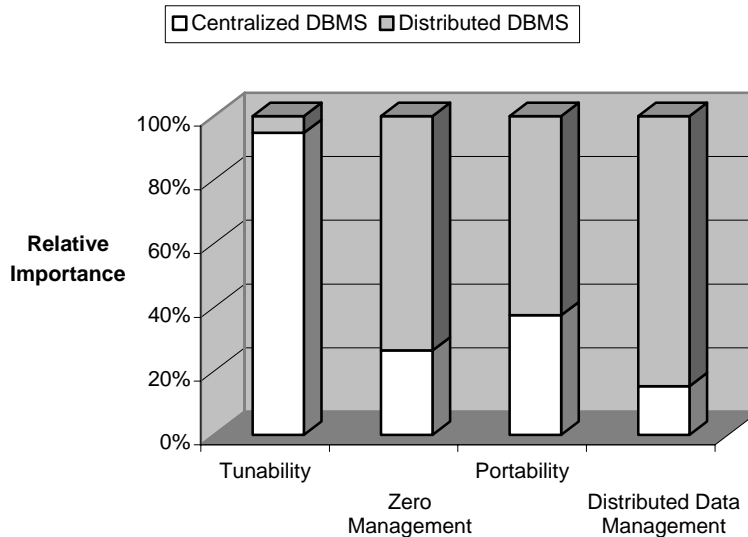


Figure 2: Different design parameters

The distributed data management environment is very different. A Distributed Data Manager must run for long periods unattended, and even when completely isolated from the network. It may run on a device with severe resource constraints – some devices may have no persistent storage at all. While distributed applications often demand a high transaction rate, their other data processing demands are modest: a handful of connections, relatively simple SQL, a few Gigabytes of data or less – sometimes much less. On the other hand, the operational demands are stringent. The DBMS must be able to embed invisibly into applications so that there is no separate installation or management. It must run on the real-time operating systems (RTOS) of the device. It must not need human intervention for routine tasks. It may have to cooperate with tens or thousands of other databases running on the other devices with which it interacts. Just as the central RDBMS provides an environment in which data integrity is guaranteed, so must a Distributed Data Manager guarantee transactional consistency across a number of loosely connected devices. It must also make it possible for applications to intelligently upgrade themselves, and make it possible for them to work around network disconnections and other failure states.

Solid FlowEngine™

Solid Information Technology has developed a Distributed Data Manager explicitly for the needs of distributed applications. The Solid FlowEngine™ is made up of a high-performance SQL-92 RDBMS tightly integrated with bi-directional synchronization that guarantees data consistency across a loosely connected heterogeneous system. These features are combined into a package that requires few computing resources, runs on Real Time Operating Systems (RTOS), and requires no DBA attention, even to recover from a variety of system and network failures.

Embedded Data Management

In 1976, the pioneering computer researcher Niklaus Wirth wrote a book that he entitled “Algorithms + Data Structures = Programs” drawing attention to the basic fact that what computer programs do is manipulate data. Every non-trivial program contains code to manage its data. In writing a new application, especially for a non-standard device, the temptation is to write all of the data management as part of the program. Many years of experience has shown that this is a short-sighted decision. The data management task increases in complexity over time, and often the data has to be shared with other applications, requiring a shared transaction management system. A competent development team can build a data manager that will handle most initial requirements, though they may have trouble evolving it to fulfill new demands. However, this effort distracts developers from their primary task, which is to provide value to their internal or external customers.

The figure below shows the reduction in the amount of functionality that needs be developed once an embedded RDBMS is available. An independent study of Solid customers conducted by Enterprise Applications Consulting showed development time savings of up to 50% that translated directly into reduced time to market and improved application functionality. (The full study is available on the Solid web site at <http://www.solidtech.com/library/whitepapers.html>.)

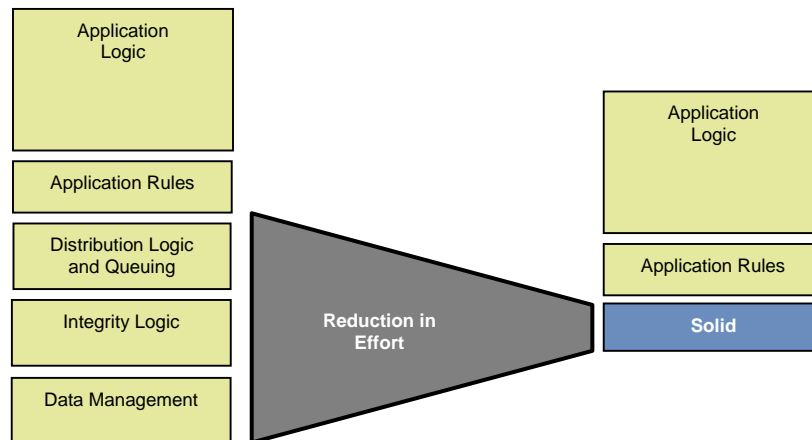


Figure 3. Solid frees developers to focus on customer value

The Solid FlowEngine is a full SQL RDBMS that complies with the SQL-92 standard. Applications communicate with it via standard ODBC or JDBC calls; though a higher-performance C-level interface is also provided. Application developers familiar with any other DBMS will be able to use Solid FlowEngine straight away, using their familiar tools. Solid enables them to take distributed data management for granted, and focus on delivering customer value.

Solid FlowEngine is a multi-user, multi-threaded data manager that provides full ACID transaction semantics. ACID transactions are Atomic, in that they happen completely or not at all. They are Consistent, meaning that every transaction, successful or not, leaves the database in a consistent state. They are Isolated, meaning that multiple transactions can be taking place simultaneously, but they do so in isolation, invisible to other transactions until they commit or roll back. They are Durable, meaning that a committed transaction is guaranteed not to be lost. These full transaction semantics simplify the development of applications that must share the same data, or modify several pieces of data at once. Using Solid removes the need for applications to handle file I/O directly and guarantees full data consistency, even in after application error or system crash situations.

The flexible SQL foundation of FlowEngine makes it easy to change the data architecture of an application, or to add to the database to support new applications. The standard SQL access controls enable fine-grained control over the kinds of access to the data allowed to applications and users.

For nodes without persistent storage, Solid FlowEngine is available in the Solid Diskless Option™. In this configuration, the FlowEngine initializes itself from a device fitted with persistent storage. It can then continue to run, offline from the rest of the world, or as we shall see in the next section, it may periodically synchronize itself with another FlowEngine database to provide persistence for its state, and to receive new data.

To support applications that run under very stringent performance requirements, or where the presence of a separate RDBMS is not desired, Solid FlowEngine may be linked into a user application, using the Solid Accelerator Option™. In the normal case, when the Solid FlowEngine runs as a separate process from the user application, it communicates with the user application via TCP/IP. If they are both on the same machine, of course, these calls do not go as far as the wire, but they do introduce TCP/IP code into the application code path that adds little or no value. Once FlowEngine is embedded in an application, the entire TCP/IP stack is discarded, shortening the code path, and enhancing performance.

Because it is often used within the network infrastructure, where reliability is paramount, Solid provides the Solid Availability Option™, in which a primary FlowEngine database is mirrored to a secondary copy. Transactions commit to both databases before returning to the application. All data modifications are directed to the primary, but the secondary, being an exact transactional copy of the primary at all times, is available for reading, thus spreading the workload. If the primary fails, applications can rapidly switch their updates to the connection they maintain with the secondary database, which becomes the new primary. When the old primary recovers, or is replaced, it starts as a new secondary, and synchronizes itself automatically. Solid runs on High Availability hardware, like the HP MC/Service Guard and the Sun Cluster, where it reduces Mean Time To Repair (MTTR), because the FlowEngine is already running on the backup hardware. There is no need to start it, or for it to perform recovery, so software switch-over can be virtually instantaneous.

Distributed Data Management

Solid FlowEngine solves the data management needs of distributed applications by linking together a set of loosely coupled, cooperative databases that share data with one another under strict integrity and security rules. Given a long enough quiet period, the databases would all converge on a single, consistent global state. Some databases, *masters*, have authority over others, *replicas*. A master database publishes subsets of its data, in a way that allows replicas to subscribe to them. A publication is very like a database VIEW, and can contain as little as a single data element, and as much as the entire database. When a replica subscribes to a view of the data, it receives a copy of that publication, and thereafter, it can synchronize with the master at any time to be refreshed with the changes that have taken place since its last visit.

A master can manage many replicas, limited only by the processing power of the master computing platform and the bandwidth between the master and replicas. Once a system expands beyond the ability of a single master to handle the load, developers can easily create a cascading hierarchy of masters, each synchronizing with replicas that are themselves masters for their own replicas.

A server may manage replica data from multiple master databases. This is of use, for example, in setting up an N+1 failover scenario, in which N masters replicate themselves to a single replica, which can take over for any one of them if they fail. N+M reliability is also possible, where each of the N masters synchronizes with the same M replicas. Any of the replicas can take over for any of the masters. This provides a higher degree of redundancy, making the application robust in the face of more than one simultaneous failure.

Masters can synchronize with one another as peers. This, too, provides a robust architecture, in which the surviving masters can share the additional workload if any master fails.

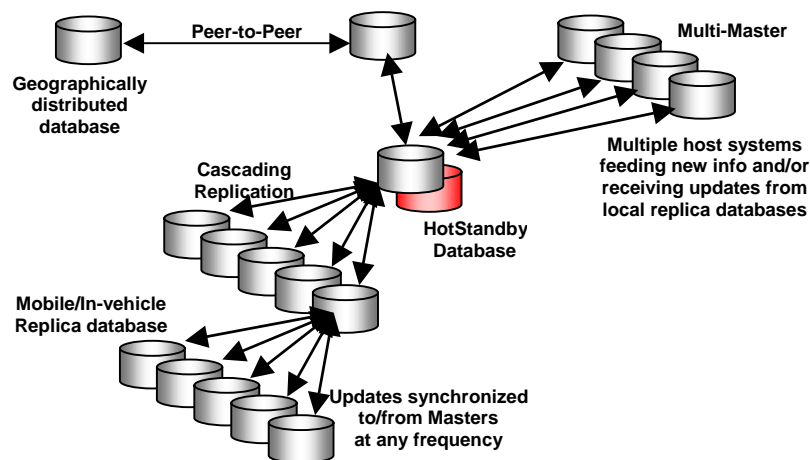


Figure 4: Possible replication architectures with Solid FlowEngine

Advanced Bi-Directional Replication

Many applications are designed to use only read-only replicas. However, in some applications, replicas must be active, writable databases. For such applications, Solid provides advanced bi-directional replication facilities that deal with some of the special issues in such systems.

In any loosely connected system, conflicts may occur. A familiar example is two people trying to make a date by voicemail. The two parties may each leave voicemail for the other, proposing different meeting times. This is a conflict that must be resolved, by converging on an acceptable meeting time in a series of voicemails. The possibility of conflict arises not because of any flaw in the voicemail system, but rather because of its loosely connected, asynchronous nature. Any loosely connected, asynchronous system is liable to conflicts. Careful application design can reduce conflict, and even eliminate it in cases where the data can be completely partitioned. Where conflict may be frequent, the usability of a system revolves around its conflict resolution mechanism.

In older enterprise replications systems, if replicas make changes to their data, those changed data rows are sent to be merged into the master, where conflict resolution takes place. Unfortunately, the raw data rows themselves lack the business context needed for the master to make a sound business decision, and decision making often degenerates to "first wins" or "last wins."

Solid has patented a replication technique, Intelligent Transactions™, that does two important things differently: firstly, it encapsulates business context information with the change message; and secondly, it passes transactions, not changed data, from replica to master. In a Solid distributed application, a replica is free to modify its copy of the data, but the change is not permanent until the master reruns the transaction and checks it for validity and integrity. At the master, the transaction code may be more elaborate, since the master often has access to additional resources, and has a special responsibility for the integrity of the data. The results of the master transaction propagate to the replica, where they bring local data back into global synchronization. If necessary, the local application can bring conflict resolution changes to the attention of the user.

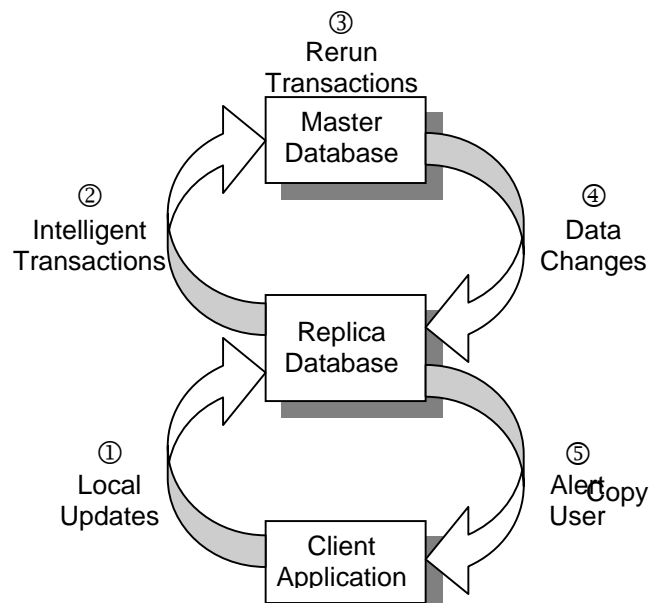


Figure 5: Intelligent transactions and conflict resolution

This centralized, intelligent conflict resolution enables the easy development of distributed applications that behave deterministically. In a sense, Solid provides a new kind of long-lived transaction, in which applications can continue to use locally consistent data, but are periodically resynchronized with a master, thus maintaining global data consistency and integrity. In this way, code at the replicas may be simplified, and the master retains centralized control over the loosely connected replicas. All communication between masters and replicas is transactional and recoverable, ensuring that no database ever violates data integrity rules, and no committed transaction is ever lost. Because requirements vary from application to application, Solid provides a fully programmable API for controlling which changes are synchronized and when.

FlowEngine Applications

Solid FlowEngine in the Network Infrastructure

After a period of heavy capital expenditure in the 1990's, carriers and service providers are looking to monetize their investment by delivering value-add services on top of their existing offerings. In the IP networks, this is understood to mean the ability to provision, assure, and bill for quality of service (QoS) service level agreements (SLAs) over and above the existing "best efforts" delivery of "digital dialtone." In the PSTN, carriers are looking to emerging services like SMS to provide additional revenue. In both cases, service providers believe they can increase the average revenue per user (ARPU) by offering attractive new services for which they can bill.

Let us consider what is required to create a QoS layer in the Internet. (The PSTN operates in a similar enough manner that the same arguments carry over.) A message between, say, two wireless devices over the Internet, engages the actions of wireless access points, edge routers and core routers that may be owned and managed by different entities, and were probably built by different manufacturers, may run different operating systems, and rely on different chips. These devices have to interact with one another cooperatively in order to deliver value to the end user in a way that ensures payment for the fraction of the service that each provides. It is no surprise that the infrastructure of the Internet is a paradigmatic distributed application, hierarchically controlled, at least within segments, but with a wide range of autonomous action both within and between segments.

The Solid FlowEngine is embedded in a variety of network elements and the Element Management Systems (EMS) that control them. Network controllers are typically constructed in racks, with a rack controller caring for a number of shelf controllers, each of which manages its shelf of line cards.

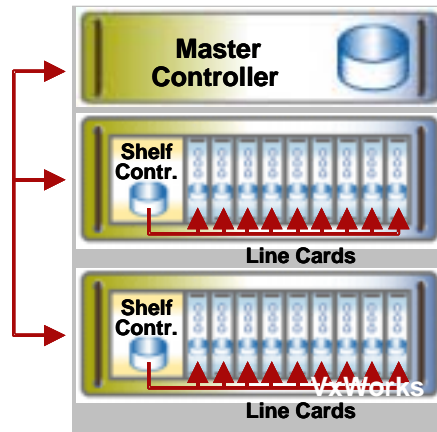


Figure 6: Internet controller architecture

This hierarchical organization perfectly matches the Solid architecture. Solid can be used in diskless line cards that initialize themselves from a shelf controller, periodically persisting their state with the shelf controller in order to speed recovery after a board fails. One board may be set up as an idle standby that acts as a replica to all the line cards on a shelf, and can be activated by the shelf master upon the failure of any line board, reproducing the state of the failed board. Because the failure of a rack controller would mean the failure of the entire rack, the rack controller may be protected with a Solid hot standby, and possibly a peer-to-peer replica at another location.

Example: Celox Networks

The new high-performance edge aggregation and IP services switch from Celox Networks, the Celox SCx192, is used by carrier-class network service providers (NSPs) to deliver QoS-based classes of service to individual users. The switch can achieve up to 80Gbps I/O throughput, which is sufficient bandwidth to deliver value-added services to millions of subscribers.

Celox sells to the NSPs and understands their needs:

- Speed time to revenue for themselves and their clients;
- Easily and quickly provision, bill, and evaluate the performance of services against SLAs;
- Scale and expand their systems with the needs of business;
- Offer five nines or better availability.

Early in design, Celox realized that it is the management of an efficient and timely distribution of information throughout the network that is the key to delivering true customer value. This free flow of information supports, and is driven by, a sophisticated element management system, the Celox SCms. With intelligence distributed throughout the network elements and coordinated in SCms, Celox enables carriers to delegate network management from the NOC to segment owners, and even to end-users, so that they can provision, manage and bill for value-add services for their subscribers. To meet these critical requirements, Celox knew it needed to incorporate a fully distributed, massively scalable, real-time data management platform.

Celox chose the Solid FlowEngine™ as its distributed data management platform. “We already knew what else was out there in the pre-packaged relational database market, and we felt that we needed an RDBMS, but also the ability to synchronize system data in real time. FlowEngine was most aligned with our requirements in that it not only had inherent capabilities to enable real-time synchronization, but we could also embed it into our platform,” says Celox Networks Senior Product Manager Joe Frandi. The ability to distribute network management from the Network Operation Center (NOC) to segment owners is a powerful differentiator for Celox, and the Solid FlowEngine is the key enabling technology.

Using FlowEngine, Celox Networks customers can store, manage, and enable real-time synchronization of system data and application code in all network elements under the control of the SCms. Subscriber information and user profiles, events and faults, topology and configuration, accounting, and other critical system data is not only viewable through a reporting interface, it is also distributed, so it is immediately available at the operational points where it is needed most.

Telematics

Telematics covers the use of embedded devices in everyday machines, such as cars, trucks, buildings, factories. Particular attention is turning now to devices that are fully autonomous in normal use, but periodically connect to the network to send and receive data, and in some cases, to receive application upgrades. Examples include public transportation, in-car systems, fleet management, and building environment and security management systems.

Telematics applications are distributed by definition, and often made up of a highly complex assemblage of many nodes of varying ability and authority. Serious concerns for those building telematics applications are, firstly, the very long life of the telematics device compared with the turnover rate of general purpose computers; and secondly, a means of coordinating the actions of loosely coupled, possibly roaming agents.

The average life of a car could be said to be 10 years, compared with the average computer life of 18 months. In-vehicle systems often deal with dynamic environments like maps of streets and location-based services, and yet they have a lifetime of a decade. Over a decade, according to Moore's law, computer systems undergo an order of magnitude improvement in most measurements. In 1990, there was effectively no World Wide Web. By 2000 it had become part of the life of hundreds of millions of people, touching almost everyone in the developed world and many outside it. The in-vehicle telematics designer has to deal with the knowledge that computing will change by this much, but in an unpredictable direction, during the life of many in-vehicle devices. Clearly the keys to successful design are flexibility and upgradeability. While systems may initialize themselves from CD-ROMs, they will require periodical upgrades of both data and logic while the application is in the field.

For data management flexibility, the RDBMS has no competitor. An RDBMS provides a robust, shared data platform that can be re-used by many applications in the same device, and even queried by applications at remote devices using standard interfaces. Solid provides the SQL functionality and the ODBC and JDBC interfaces required to deliver true relational flexibility. In a distributed Solid application, database structure may be enhanced significantly by successive application releases, and yet the whole system can operate in a transactionally consistent way while the upgrade is propagating through the system.

With Solid in the vehicle, the in-vehicle database can subscribe to data from many masters, providing an integration platform for the data and services available from a number of providers. The presence of a general purpose data manager within the vehicle provides a good deal of "future-proofing" by providing a platform on which services that have not yet been created can be deployed in the future. The Solid availability features can provide for a hot standby in the vehicle, if required, and Solid replication can be used to save the vehicle state periodically to base stations, so that data is preserved even in the face of catastrophic destruction of the vehicle.

Solid FlowEngine can replicate a wide variety of data elements to a replica, including Binary Large Objects (BLOBs). A BLOB may contain new application code to upgrade applications running on remote devices. It is trivial to create a self-upgrading system in which application patches and upgrades can be published at a master, and propagated automatically throughout an arbitrarily complex network of heterogeneous devices, with devices reporting back on successful upgrades.

Example: Buscom

Buscom Oy, headquartered in Finland, specializes in traffic management systems for public transportation operators and is one of the largest suppliers of this application in Europe. Buscom Oy has selected Solid FlowEngine™ software as the data management solution for its traffic information management systems, enabling wireless data transfer and synchronization between buses and traffic operation centers. The Solid FlowEngine technology provides the Buscom traffic management system with data management and synchronization capabilities for collecting and reconciling fare transactions in near-real time. The Solid software also allows traffic operation centers to more accurately monitor information such as passenger flow so that they can more efficiently plan route schedules. Buscom's system is used in 70 public transportation installations throughout northern and central Europe. For more information, please refer to the news article "About Buscom - Solid Cooperation" at <http://www.buscom.fi/englanti/references.html>.

Distributed enterprise

If we view a distributed application as a cooperating group of semi-autonomous entities under central control, then we have come close to describing the modern distributed enterprise. And yet most Global 1000 companies continue to base their information architecture on a centralized mode that is inappropriate for many applications. Remote reliance on central data management increases the load on some very expensive parts of the system and increases the consumption of bandwidth.

Another thrust towards distributed processing comes from consideration of the impact of the terrorist attacks on New York in September 2001. Companies started to understand that they had to structure themselves to survive the loss of even their headquarters. This demands an increasing, but controlled, delegation of authority to the edge of the enterprise.

Solid FlowEngine is used on Wall Street to replicate corporate data to remote offices, ensuring that if any failure of computer or connection occurs, edge offices can continue to function autonomously for some time. An additional benefit of this architecture is that it distributes corporate intelligence out to the edge of the enterprise, ensuring maximum leverage of the corporate IQ, by empowering edge workers, while retaining central control over data integrity. Because only data refreshes cross the network, bandwidth demands are significantly reduced.

Summary

In an application environment that is becoming increasingly distributed, developers need to consider adopting a distributed data management platform like Solid that:

- Reduces time to market while enhancing application value,

- Provides an application migration path,
- Requires no local administration,
- Embeds into device-hosted applications,
- Provides a variety of options for delivering the level of reliability required by the application,
- Utilizes familiar data management standards and APIs.

www.solidtech.com

Solid World Headquarters

444 Castro Street, Suite 1010
Mountain View, CA 94041
USA

Tel 1 650 210 9100
Fax 1 650 210 9300

Solid EMEA North

Merimiehenkatu 36D
FIN-00150 Helsinki
Finland

Tel +358 424 8888 1
Fax +358 9 278 2877

Solid Americas West

444 Castro Street, Suite 1010
Mountain View, CA 94041
USA

Tel 1 650 210 9100
Fax 1 650 210 9300

Solid EMEA South

Le Village d'Entreprises Green
Side
400, Avenue Roumanille
B.P 309
06906 SOPHIA - ANTIPOLIS
CEDEX
FRANCE

Tel + 33 4 93 00 12 52

Solid Americas East

35 Corporate Drive, Suite 400
Burlington, MA 01803
USA

Tel 1 781 229 9100
Fax 1 781 685 4831