Caché Technology Guide

Innovations by **InterSystems**

Table of Contents

INTRODUCTION 3

CHAPTER ONE

DATA MODELING – RELATIONAL OR OBJECT ACCESS 6

RELATIONAL TECHNOLOGY 7
OBJECT TECHNOLOGY AND OBJECT DATABASES 8
OBJECT VS. RELATIONAL ACCESS 8
OVERVIEW OF THE CACHÉ OBJECT DATA MODEL AND OBJECT PROGRAMMING 8
KEY OBJECT CONCEPTS 10
WHY CHOOSE OBJECTS FOR YOUR DATA MODEL? 11
OBJECT DATA STORAGE ... PLUS RELATIONAL ACCESS 11

CHAPTER TWO

CACHÉ'S MULTIDIMENSIONAL DATA SERVER 12

INTEGRATED DATABASE ACCESS 12
MULTIDIMENSIONAL DATA MODEL 13
SQL ACCESS 16
CACHÉ OBJECTS 18
WORD-AWARE TEXT SEARCHING 20
TRANSACTIONAL BIT-MAP INDEXING 22
ENTERPRISE CACHE PROTOCOL FOR DISTRIBUTED SYSTEMS 24
FAULT TOLERANCE 26
SECURITY MODEL 29

CHAPTER THREE

CACHÉ'S APPLICATION SERVER 32

THE CACHÉ VIRTUAL MACHINE AND SCRIPTING LANGUAGES 32
CACHÉ OBJECTSCRIPT 34
BASIC 40
MV BASIC 42
C++ 42
JAVA 42
CACHÉ AND JALAPEÑO 44
CACHÉ AND .NET 46
CACHÉ AND XML 47
CACHÉ AND WEB SERVICES 48
CACHÉ AND MULTIVALUE 49

CHAPTER FOUR

BUILDING FAST WEB APPS FAST WITH CACHÉ SERVER PAGES 52

THE CACHÉ SERVER PAGE MODEL 54
CLASS ARCHITECTURE OF WEB PAGES 56
MULTIPLE DEVELOPMENT STRATEGIES 56
CSP FILES 57
HYPER-EVENTS 58

Introduction

THE COMPUTING WORLD HAS ENTERED THE “POST-RELATIONAL” ERA

Thirty years ago, relational databases were hailed as a great innovation. Instead of monolithic legacy databases, each with its unique data schema, data would be stored in a tabular format, and be accessible to anyone who knew SQL. Relational databases were highly successful, and SQL became a common standard for database access. However, as is common with older technologies, relational databases have limitations that reduce their applicability to today's world – primarily in the realms of performance/scalability, ease of use, and fit with today's development technologies.

The usage and complexity of computer applications are exploding, and today's systems increasingly have processing requirements that outstrip the capabilities of relational technology. Many key applications that demanded high performance and scalability never made the transition to relational databases, and today even simple applications are beginning to approach the limits of traditional relational technology.

“Impedance mismatch” between relational databases and today's development technologies has become a serious problem – making development more complex and the chances of failure greater. While the simplicity of tabular structures supports an elegant query language (SQL), it is difficult to decompose real-world data structures into such simplistic rows and columns. The result is a huge number of tables whose relationships are hard to remember and hard to use – rows and columns are simple, but the pervasive need to program left outerjoins, stored procedures, and triggers is not.

Modern applications are usually written using object technology, which enables a faster and more intuitive way of describing and using information. Development is faster, and reliability increases. Unfortunately, objects are not natively compatible with relational databases. The advantages of object technology get blunted when the resulting database objects have to be forced into the two-dimensional relational model.

Today's transaction processing applications have requirements that outstrip the capabilities of relational technology – they must span large networks, service thousands of clients, but still provide superb performance, Web compatibility, and simple operations at low cost. And they must be developed quickly!

Introducing Caché

InterSystems Caché® is a new generation of ultra-high-performance database technology. It combines an object database, high-performance SQL, and powerful multidimensional data access – all of which can simultaneously access the same data. Data is only described once in a single integrated data dictionary and is instantly available using all access methods. Caché provides levels of performance, scalability, rapid programming, and ease of use unattainable by relational technology.

But Caché is much more than a pure database technology. Caché includes an Application Server with advanced object programming capabilities, the ability to easily integrate with a wide variety of technologies, and an extremely high-performance runtime environment with unique data caching technology.

Caché comes with several built-in scripting languages: Caché ObjectScript, a powerful yet easy-to-learn object-oriented programming language; Caché Basic, a superset of the widespread Basic programming language including extensions for powerful data access and object technology; and Caché MVBasic, a variant of Basic used by MultiValue applications (sometimes referred to as Pick applications). Other languages, such as Java, C#, and C++, are supported through direct call-in or through other interfaces, including ODBC, JDBC, .NET, and a Caché-provided object interface that allows accessing the Caché database and other Caché facilities as properties and methods.

Caché also goes beyond traditional databases by incorporating a rich environment for developing sophisticated browser-based (Web) applications. Caché Server Pages (CSP) technology allows the rapid development and execution of dynamically generated Web pages. Thousands of simultaneous Web users can access database applications, even on low-cost hardware.

For non-browser based applications, the user interface is typically programmed in one of the popular client user interface technologies, such as Java, .NET, Delphi, C#, or C++. Best results (fastest programming, greatest performance, and lowest maintenance) are usually obtained by performing all of the rest of the development within Caché. However, Caché also provides extremely high levels of interoperability with other technologies and supports all of the most commonly used development tools, so a wide range of development methodologies are available.

Chapter One: Data Modeling – Relational or Object Access

Early in the process of designing a new application, developers must decide upon their approach towards data modeling. For most, this comes down to a choice between the traditional modeling of data as relational tables and the newer approach of modeling as objects. Faced with the need to handle complex data, many developers believe that modeling with objects is a more effective approach.

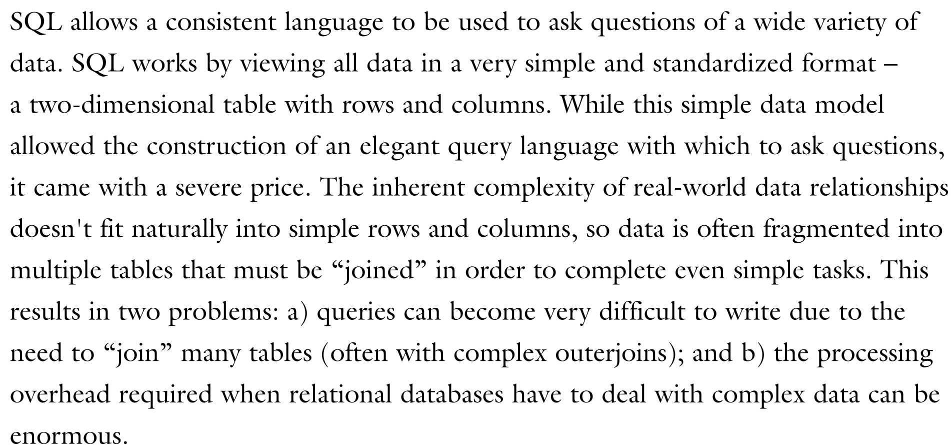
Of course, when moving an existing application to Caché, the first step is to migrate the existing data model. There are easy ways to import data models from various relational or object representations so that the result is a standard Caché data definition. Once migrated to Caché, data can be simultaneously accessed as objects, relational tables, and multidimensional arrays.

Caché supports both SQL and object data access, and at times each is appropriate. To understand the uses of each and why data modeling with objects is generally preferred by modern-day developers, it is useful to understand how and why each has developed.

RELATIONAL TECHNOLOGY

In the early days of computing, information processing was done on huge mainframe systems and data access was, for the most part, limited to IT professionals. Databases tended to be home grown, and retrieving data effectively required a thorough knowledge of the database. If users wanted a special report, they usually had to ask an overworked central staff to write it, and it usually wasn't available in time to influence decisions.

Although relational technology was originally developed in the 1970s on the mainframe, it remained largely a research project until it began to appear in the 1980s on mini-computers. With the advent of PCs, the world entered a more “user-centric” era of computing with more user-friendly report writers based on SQL – the query language introduced by relational technology. Users could now produce their own reports and ad hoc queries of the database, and relational usage exploded.



SQL allows a consistent language to be used to ask questions of a wide variety of data. SQL works by viewing all data in a very simple and standardized format – a two-dimensional table with rows and columns. While this simple data model allowed the construction of an elegant query language with which to ask questions, it came with a severe price. The inherent complexity of real-world data relationships doesn't fit naturally into simple rows and columns, so data is often fragmented into multiple tables that must be “joined” in order to complete even simple tasks. This results in two problems: a) queries can become very difficult to write due to the need to “join” many tables (often with complex outerjoins); and b) the processing overhead required when relational databases have to deal with complex data can be enormous.

SQL has become a standard for database interoperability and reporting tools. However, it is important to understand that while SQL grew out of relational databases, it need not be constrained by them. Caché supports standard SQL as a query and update language, using a much stronger multidimensional database technology, and it is extended to include object capabilities.

OBJECT TECHNOLOGY AND OBJECT DATABASES

Object programming and object databases are a practical result of work to simulate complex activities of the brain. It was observed that the brain is able to store very complex and different types of data and yet still manipulate such seemingly different information in common ways. To support that simulation, very complex behavior needed to be implemented in programs while hiding that complexity – supporting simpler, more generalized and understandable logic with adaptable, reusable functionality. Clearly, these characteristics are also true of today's leading-edge applications, and a technology that lets developers work in a natural manner that is more similar to how humans think is a huge advantage.

OBJECT VS. RELATIONAL ACCESS

In object technology, the complexity of the data is contained within the object, and the data is accessed by a simple consistent interface. In contrast, relational technology also provides a simple consistent interface, but because it does nothing to manage real-world data complexity – the data is scattered among multiple tables – the user or programmer is responsible for constantly dealing with that complexity.

Because objects can model complex data simply, object programming is the best choice for programming complex applications. Similarly, object access of the database is the best choice for inserting and updating the database (i.e., for transaction processing).

Caché complements object access with an object-extended SQL query language. SQL is a powerful language for searching a database and is widely used by reporting tools. However, we believe SQL is best suited for that purpose – queries and reports – rather than for transaction processing (for which it is cumbersome and often inefficient). Caché SQL's object extensions eliminate much of the cumbersome join syntax, making SQL even easier to use.

OVERVIEW OF THE CACHÉ OBJECT DATA MODEL AND OBJECT PROGRAMMING

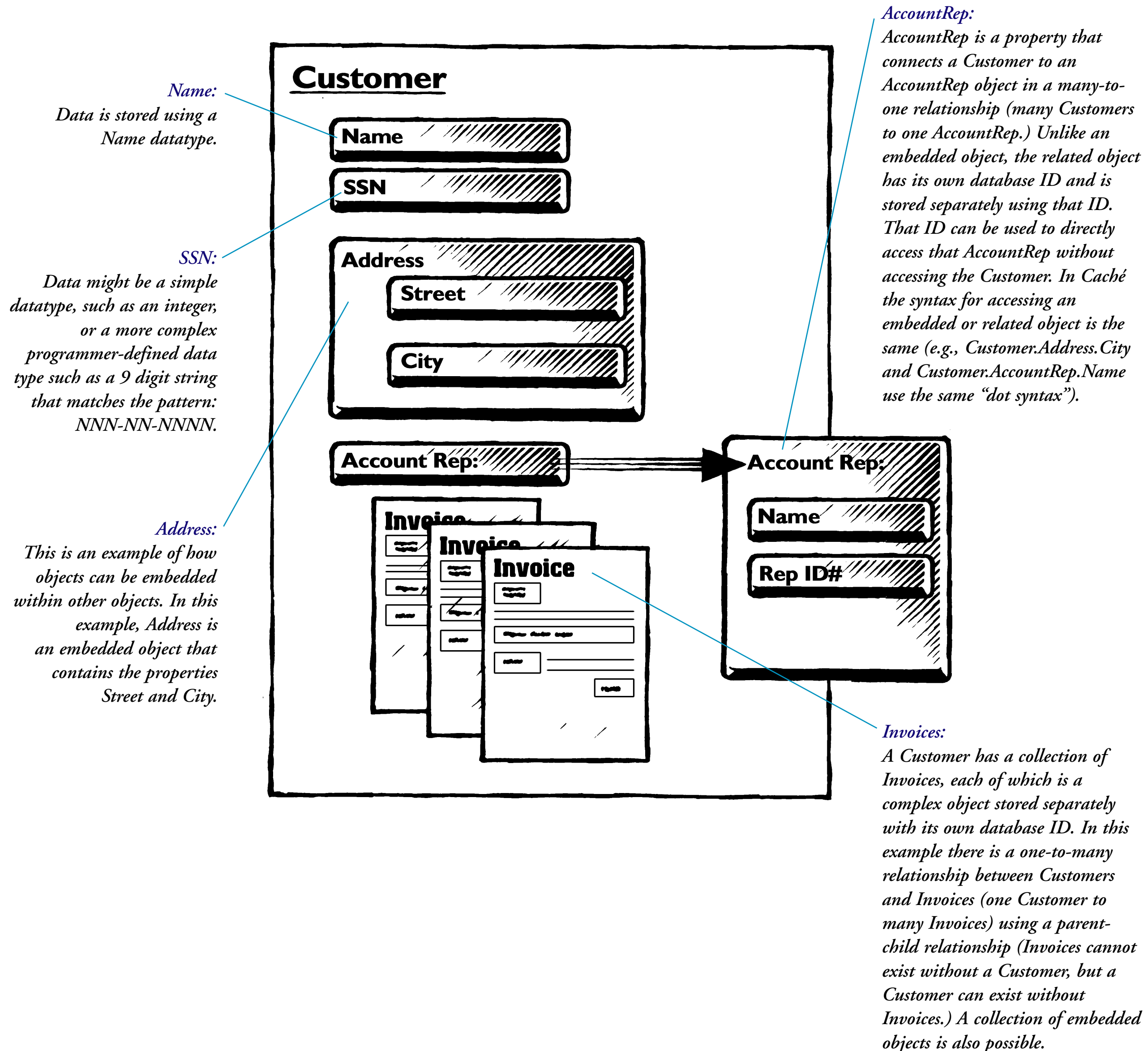
The Caché object model is based upon the ODMG (Object Database Management Group) standard and supports many advanced features, including multiple inheritance.

Object technology attempts to mirror the way that humans actually think about and use information. Unlike relational tables, objects bundle together both data and code. For example, an Invoice object might have data, such as an invoice number and a total amount, and code, such as Print().

Conceptually, an object is a package that includes that object's data values (“properties”) and a copy of all of its code (“methods”). An object's methods send messages to communicate with other methods. To reduce storage, it is common for objects of the same class to share the same copy of code (e.g., it would be unrealistic for each Invoice object to have its own private copy of code). Also, in Caché, method calls typically result in efficient function calls rather than enduring the overhead of passing messages. However, these implementation techniques are hidden from the programmer; it is always accurate to think in terms of objects passing messages.

What is the difference between an object and a class? A class is the definitional structure and code provided by the programmer. It includes a description of the nature of the data (its "type") and how it is stored as well as all of the code, but it does not contain any data. An object is a particular "instance" of a class. For example, invoice #123456 is an object of the Invoice class.

Object technology also promotes a natural view of data by not restricting properties to simple, computer-centric data types. Objects may contain other objects, or references to other objects, which makes it easy to build useful and meaningful data models. Here's a simple example of a Customer object:



Name:
Data is stored using a Name datatype.

SSN:
Data might be a simple datatype, such as an integer, or a more complex programmer-defined data type such as a 9 digit string that matches the pattern: NNN-NN-NNNN.

Address:
This is an example of how objects can be embedded within other objects. In this example, Address is an embedded object that contains the properties Street and City.

AccountRep:
AccountRep is a property that connects a Customer to an AccountRep object in a many-to-one relationship (many Customers to one AccountRep.) Unlike an embedded object, the related object has its own database ID and is stored separately using that ID. That ID can be used to directly access that AccountRep without accessing the Customer. In Caché the syntax for accessing an embedded or related object is the same (e.g., Customer.Address.City and Customer.AccountRep.Name use the same "dot syntax").

Invoices:
A Customer has a collection of Invoices, each of which is a complex object stored separately with its own database ID. In this example there is a one-to-many relationship between Customers and Invoices (one Customer to many Invoices) using a parent-child relationship (Invoices cannot exist without a Customer, but a Customer can exist without Invoices.) A collection of embedded objects is also possible.

KEY OBJECT CONCEPTS

Inheritance is the ability to derive one class of objects from another. The new class (a subclass) contains all of the properties and methods of its superclass, as well as additional properties and methods unique to it. Objects of the subclass can be thought of as having an "is a" relationship to its superclass. For example, a dog "is a" mammal, so it makes sense for the Dog class to inherit all the properties and methods of the Mammal class plus have additional properties and methods such as a DogTagNumber. A subclass may also override an inherited definition (e.g., the Print() method for a subclass of the Invoice class may be different from the Print() method of Invoice). Inheritance promotes reusability of code and makes it easier to introduce major improvements.

Multiple inheritance means a subclass can be derived from more than one superclass. For example, a dog "is a" mammal and "is a" pet, so the object class "Dog" can inherit the properties and methods of both the "Mammal" class and the "Pet" class.

Encapsulation means that objects can be viewed as a sort of "black box". Public properties and methods can be accessed by methods of any class, whereas private properties and methods can only be accessed by methods of the same class. Thus, the application doesn't need to know the internal workings of an object – it deals only with the public properties and methods. The power of encapsulation is that programmers can improve the inner workings of a class without affecting the rest of the application.

Polymorphism refers to the fact that methods used in multiple classes can share a common interface, even if the underlying implementation is different. For example, suppose the classes Letter, Mailing Label, and ID Badge all contain a method called Print( ). To print, an application doesn't need to know which type of object it is accessing – it merely calls the object's Print( ) method.

THE CACHÉ ADVANTAGE

Caché is fully object-enabled, providing all the power of object technology to developers of high-performance transaction processing applications.

Intuitive Data Modeling: Object technology lets developers think about and use information – even extremely complex information – in simple and realistic ways, thus speeding the application development process.

Rapid Application Development: The object concepts of encapsulation, inheritance, and polymorphism allow classes to be reused, re-purposed, and shared between applications, enabling developers to leverage their work over many projects

WHY CHOOSE OBJECTS FOR YOUR DATA MODEL?

For new database applications, most developers choose to use object technology because they can develop complex applications more rapidly and more easily modify them later. Object technology provides many benefits:

- Objects support a richer data structure that more naturally describes real-world data.
- Programming is simpler – it is easier to keep track of what you are doing and what you are manipulating.
- Customized versions of classes can easily replace standard ones, making it easier to customize an application.
- The black box approach of encapsulation means programmers can improve the internal workings of objects without affecting the rest of the application.
- Objects provide a simple way to connect different technologies and different applications.
- Object technology is a natural match with graphical user interfaces.
- Many new tools assume object technology.
- Objects provide a good insulation between the user interface and the rest of the application. Thus, when it becomes necessary to adopt a new user interface technology (perhaps some currently unforeseen future technology), you can reuse most of your code.

OBJECT DATA STORAGE ...

Unfortunately, although many applications are now being written with object programming languages, they often try to force object data into flat relational tables. This significantly impairs the advantages of object technology.

Caché provides a multidimensional data structure that naturally stores rich object data. The result is faster data access and faster programming.

... PLUS RELATIONAL ACCESS

Of course, many tools (such as report writers) use SQL, not object technology, for accessing data.

A unique feature of Caché is that whenever a database object class is defined, Caché automatically provides full SQL access to that data. Thus, with no additional work, SQL-based tools will immediately work with Caché data, and even they will experience the high-performance advantage of the Caché Multidimensional Data Server.

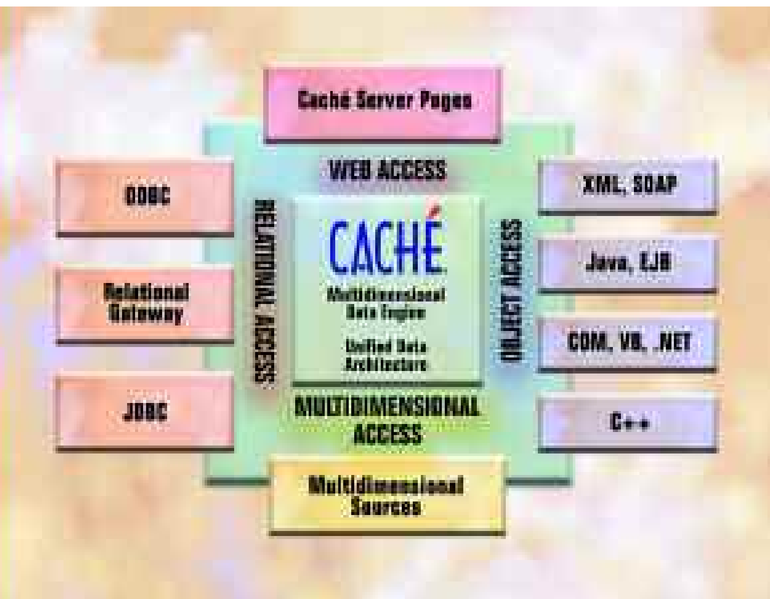
The reverse is also true. When a DDL definition of a relational database is imported, Caché automatically generates an object description of the data, enabling immediate access as objects, as well as through SQL.

The Caché Unified Data Architecture keeps these access paths synchronized; there is only one data description to edit.

Chapter Two: Caché's Multidimensional Data Server

Caché's high-performance database uses a multidimensional data engine that allows efficient and compact storage of data in a rich data structure. Objects and SQL are implemented by specifying a unified data dictionary that defines the classes and tables and provides a mapping to the multidimensional structures – a mapping that can be automatically generated.

Multidimensional Access



INTEGRATED DATABASE ACCESS

Caché gives programmers the freedom to store and access data through objects, SQL, or direct access to multidimensional structures. Regardless of the access method, all data in Caché's database is stored in Caché's multidimensional arrays.

Once the data is stored, all three access methods can be simultaneously used on the same data with full concurrency.

A unique feature of Caché is its Unified Data Architecture. Whenever a database object class is defined, Caché automatically generates a SQL-ready relational description of that data. Similarly, if a DDL description of a relational database is imported into the Data Dictionary, Caché automatically generates both a relational and an object description of the data, enabling immediate access as objects. Caché keeps these descriptions coordinated; there is only one data definition to edit. The programmer can edit and view the dictionary both from an object and a relational table perspective.

Caché automatically creates a mapping for how the objects and tables are stored in the multidimensional structures, or the programmer can explicitly control the mapping.

THE CACHÉ ADVANTAGE

Flexibility: Caché's data access modes – Object, SQL, and multidimensional – can be used concurrently on the same data. This flexibility gives programmers the freedom to think about data in the way that makes most sense and to use the access method that best fits each program's needs.

Less Work: Caché's Unified Data Architecture automatically describes data as both objects and tables with a single definition. There is no need to code transformations, so applications can be developed and maintained more easily.

Leverage Existing Skills and Applications: Programmers can leverage existing relational skills and introduce object capabilities gradually into existing applications as they evolve.

MULTIDIMENSIONAL DATA MODEL

At its core, the Caché database is powered by an extremely efficient multidimensional data engine. The built-in Caché scripting languages support direct access to the multidimensional structures – providing the highest performance and greatest range of storage possibilities – and many applications are implemented entirely using this data engine directly. Direct "global access" is particularly common when there are unusual or very specialized structures and no need to provide object or SQL access to them, or where the highest possible performance is required.

There is no data dictionary, and thus no data definitions, for the multidimensional data engine.

Rich Multidimensional Data Structure

Caché's multidimensional arrays are called "globals". Data can be stored in a global with any number of subscripts. What's more, subscripts are typeless and can hold any sort of data. One subscript might be an integer, such as 34, while another could be a meaningful name, like "LineItems" – even at the same subscript level.

For example, a stock inventory application that provides information about item, size, color, and pattern might have a structure like this:

```
^Stock(item,size,color,pattern) = quantity
```

Here's some sample data:

```
^Stock("slip dress",4,"blue","floral")=3
```

With this structure, it is very easy to determine if there are any size 4 blue slip dresses with a floral pattern – simply by accessing that data node. If a customer wants a size 4 slip dress and is uncertain about color and pattern, it is easy to display a list of all of those by cycling through all of the data nodes below:

```
^Stock("slip dress",4).
```

In this example, all of the data nodes were of a similar nature (they stored a quantity), and they were all stored at the same subscript level (four subscripts) with similar subscripts (the third subscript was always text representing a color). However, they don't have to be. Data nodes may have a different number or type of subscripts, and they may contain different types of data.

Here's an example of a more complex global with invoice data that has different types of data stored at different subscript levels:

```
^Invoice(invoice #,"Customer") = Customer information
^Invoice(invoice #,"Date") = Invoice date
^Invoice(invoice #,"Items") = # of Items in the invoice
^Invoice(invoice #,"Items",1,"Part") = part number of 1st Item
^Invoice(invoice #,"Items",1,"Quantity") = quantity of 1st Item
^Invoice(invoice #,"Items",1,"Price") = price of 1st Item
^Invoice(invoice #,"Items",2,"Part") = part number of 2nd Item
etc.
```

Multiple Data Elements Per Node

Often only a single data element is stored in a data node, such as a date or quantity, but sometimes it is useful to store multiple data elements together in a single data node. This is particularly useful when there is a set of related data that is often accessed together. It can also improve performance by requiring fewer accesses of the database.

For example, in the above invoice, each item included a part number, quantity, and price all stored as separate nodes, but they could be stored as a list of elements in a single node:

```
^Invoice(invoice #,"LineItems",item #).
```

To make this simple, Caché supports a function called $list(), which can assemble multiple data elements into a length delimited byte string and later disassemble them. Elements can in turn contain sub-elements, etc.

Logical Locking Promotes High Concurrency

In systems with thousands of users, reducing conflicts between competing processes is critical to providing high throughput. One of the biggest conflicts is between transactions wishing to access the same data.

Caché processes don't lock entire pages of data while performing updates. Instead, because transactions require frequent access or changes to small quantities of data, database locking in Caché is done at a logical level. Database conflicts are further reduced by using atomic addition and subtraction operations, which don't require locking. (These operations are particularly useful in incrementing counters used to allocate ID numbers and for modifying statistics counters.)

With Caché, individual transactions run faster, and more transactions can run concurrently.

Variable Length Data in Sparse Arrays

Because Caché data is inherently variable length and is stored in sparse arrays, Caché often requires less than half of the space needed by a relational database. In addition to reducing disk requirements, compact data storage enhances performance because more data can be read or written with a single I/O operation and data can be cached more efficiently.

Declarations and Definitions Aren't Required

Caché multidimensional arrays are inherently typeless, both in their data and subscripts. No declarations, definitions, or allocations of storage are required. Global data simply pops into existence as data is inserted.

Namespaces

In Caché, data and code are stored in disk files with the name CACHE.DAT (only one per directory). Each such file contains numerous “globals” (multidimensional arrays). Within a file, each global name must be unique, but different files may contain the same global name. These files may be loosely thought of as databases.

Rather than specifying which database file to use, each Caché process uses a “namespace” to access data. A namespace is a logical map that maps the names of multidimensional global arrays and code to databases. If a database is moved from one disk drive or computer to another, only the namespace map needs to be updated. The application itself is unchanged.

Usually, other than some system information, all data for a namespace is stored in a single database. However, namespaces provide a flexible structure that allows arbitrary mapping, and it is not unusual for a namespace to map the contents of several databases, including some on other computers.

THE CACHÉ ADVANTAGE

Performance: By using an efficient multidimensional data model with sparse storage techniques instead of a cumbersome maze of two-dimensional tables, data access and updates are accomplished with less disk I/O. Reduced I/O means that applications will run faster.

Scalability: The transactional multidimensional data model allows Caché-based applications to be scaled to many thousands of clients without sacrificing high performance. That’s because data access in a multidimensional model is not significantly affected by the size or complexity of the database in comparison to relational models. Transactions can access the data they need without performing complicated joins or bouncing from table to table.

Caché's use of logical locking for updates instead of locking physical pages is another important contributor to concurrency, as is its sophisticated data caching across networks.

Rapid Development: With Caché, development occurs much faster because the data structure provides natural, easily understood storage of complex data and doesn’t require extensive or complicated declarations and definitions. Direct access to globals is very simple, allowing the same language syntax as accessing local arrays.

Cost-Effectiveness: Compared to similarly sized relational applications, Caché-based applications require significantly less hardware and no database administrators. System management and operations are simple.

SQL ACCESS

SQL is the query language for Caché, and it is supported by a full set of relational database capabilities – including DDL, transactions, referential integrity, triggers, stored procedures, and more. Caché supports access through ODBC and JDBC (using a pure Java-based driver). SQL commands and queries can also be embedded in Caché ObjectScript and within object methods.

SQL accesses data viewed as tables with rows and columns. Because Caché data is actually stored in efficient multidimensional structures, applications that use SQL achieve better performance with Caché than with traditional relational databases.

Caché supports, in addition to the standard SQL syntax, many of the commonly used extensions in other databases so that many SQL-based applications can run on Caché without change – especially those written with database independent tools. However, vendor-specific stored procedures will require some work, and InterSystems has translators to help with that work.

Caché SQL includes object enhancements that make SQL code simpler and more intuitive to read and write.

TRADITIONAL SQL

```
SC.FullName, SM.Descr, MS.Value,
SI.InvDate, SI.InvNumber
FROM
MainSales MS, SalesItem SI,
SalesProduct SP, SalesCustomer SC,
SalesMarket SM
WHERE
SI.SalesItemID *= MS.SalesItem
AND SP.SalesProductID *= MS.Product
AND SC.SalesCustomerID *= MS.Customer
AND SM.SalesMarketID *= SC.SalesMarket
AND SP.Descr = 'Hammer'
```

OBJECT EXTENDED SQL

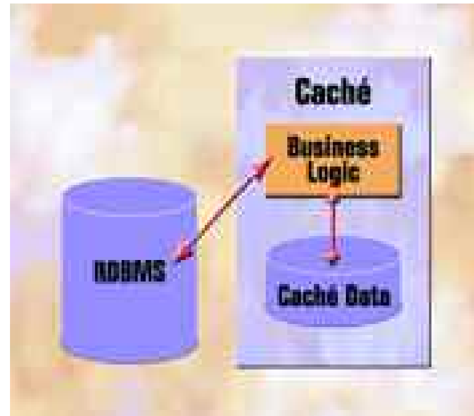
```
SELECT
Customer->FullName,
Customer->SalesMarket->Descr, Value,
SalesItem->InvDate, SalesItem-
>InvNumber
FROM MainSales
WHERE Product->Descr = 'Hammer'
```

Accessing Relational Databases with Caché Relational Gateway

The Caché Relational Gateway enables a SQL request that originates in Caché to be sent to other (relational) databases for processing. Using the Gateway, a Caché application can retrieve and update data stored in most relational databases.

Additionally, if Caché database classes are compiled using the CachéSQLStorage option, the Gateway allows Caché applications to transparently use relational databases. However, applications will run faster and be more scalable if they access Caché's post-relational database.

Relational Gateway



THE CACHÉ ADVANTAGE

Faster SQL: Relational applications can enjoy significantly enhanced performance by using Caché SQL to tap into Caché's efficient post-relational database.

Faster Development: In Caché, SQL queries can be written more intuitively, using fewer lines of code.

Compatibility with Relational Applications and Report Writers: Caché's native ODBC and JDBC drivers provide high-performance access to the Caché database for relational applications and reporting tools. The Caché Relational Gateway enables Caché applications to use SQL to access other (relational) databases.

CACHÉ OBJECTS

Caché's object model is based upon the ODMG standard. Caché supports a full array of object programming concepts, including encapsulation, embedded objects, multiple inheritance, polymorphism, and collections.

The built-in Caché scripting languages directly manipulate these objects, and Caché also exposes Caché classes as Java, EJB, COM, .NET, and C++ classes. Caché classes can also be automatically enabled for XML and SOAP support by simply clicking a button in the Studio IDE. As a result, Caché objects are readily available to every commonly used object technology.

There are several ways for a program outside of the Caché Application Server to access Caché classes:

1. Any Caché class can be projected as a class in the native language. When a Java, C++, C#, or other program accesses a Caché object, it calls a template of the class in the native language. That template class (which is automatically generated by Caché) communicates with the Caché Application Server to invoke methods on the Caché server and to access or modify properties. State for the Caché objects is maintained in the Caché Application Server. To speed execution and reduce messaging, Caché caches a copy of the object's data on the client and piggybacks updates with other messages when possible.

2. A "lighter-weight" projection can be used for database classes in which the native language template class directly accesses the database – bypassing the Application Server. The object's state is not kept on the Application Server; the in-memory properties are only maintained in the client. This approach provides significantly higher throughput but less functionality, since server-side instance methods of the class (i.e., methods that need access to the in-memory properties) cannot be invoked.

3. InterSystems Jalapeño technology allows Java developers to first create Java database classes just like any other POJO (plain old Java object) class in their IDE of choice and then have Caché automatically generate a database schema and corresponding Caché class. Using this approach, the Java class is unchanged, and the application continues to access its properties and methods. Caché provides a library class ("ObjectManager") with an API that is used to store and retrieve database objects and issue queries.

With each of these three approaches, the object appears to be local to the user program. Caché transparently handles all communications, using either call-in or TCP.

The Java template and supporting library is completely Java-based, so it can be used across the Web or on specialized Java devices.

Method Generators

Caché includes a number of unique advanced object technologies – one of which is method generators. A method generator is a method that executes at compile time, generating code that can run when the program is executed. A method generator has access to class definitions, including property and method definitions and parameters, to allow it to generate a method that is customized for the class. Method generators are particularly powerful in combination with multiple inheritance – functionality can be defined in a multiply inherited class that customizes itself to the subclass.

THE CACHÉ ADVANTAGE

Caché is fully object-enabled, providing all the power of object technology to developers of high-performance transaction processing applications.

Rapid Application Development: Object technology is a powerful tool for increasing programmer productivity. Developers can think about and use objects – even extremely complex objects – in simple and realistic ways, thus speeding the application development process. Also, the innate modularity and interoperability of objects simplifies application maintenance, and lets programmers leverage their work over many projects.

Natural Development: Database objects appear as objects native to the language being used by the developer. There is no need to write tedious code to decompose objects into rows and columns and later re-assemble them.

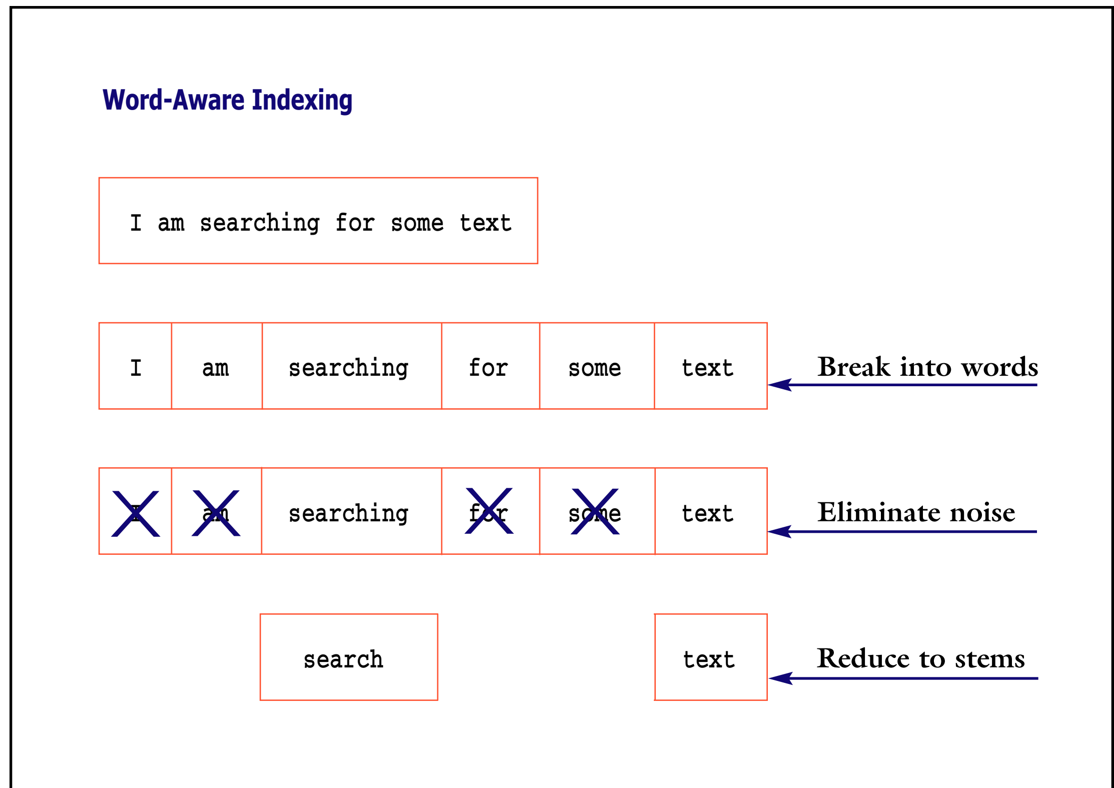
WORD-AWARE TEXT SEARCHING

Caché supports free text searching in which queries can search for text containing words of interest, even though the actual words in the text may be variants of the search words.

To utilize Word-Aware searching, the text field must be Word-Aware indexed, which occurs in the following steps:

1. Discrete words in the text field are first identified.
2. Words that are so common as to provide little search value are removed (e.g., words such as "the" or "for" are removed).
3. The remaining words are reduced to their stem words (e.g., "searching" becomes "search" and "flowers" becomes "flower").
4. The resulting words are indexed.

In Word-Aware searching, the search text is usually first processed in a similar manner, and then the index is used to produce matches.



Word-Aware indexes are maintained by object and SQL updates. Searching is most commonly done through SQL queries, although procedural code can use the indexes directly. Such queries may include AND/OR logic for more sophisticated searches.

Word-Aware algorithms are specific to the natural language being used. Word-Aware searching is available for a wide range of natural languages, including English, French, German, Italian, Japanese, Portuguese, and Spanish. Others are being added.

Word-Aware Searching

WHERE Description %Contains ('search')	Caché finds "search", "searched", "searching", …
WHERE Description %Contains ('close')	Caché finds "close", "closed", … But not "closet" or "disclose".

THE CACHÉ ADVANTAGE

Powerful Unstructured Text Searches: Unstructured text, such as physician's notes or documents, can be easily searched for keywords and related words.

Extremely Rapid Searches: Coupling Word-Aware with Caché bit-map technology, searching of massive quantities of text can be performed in a fraction of a second.

TRANSACTIONAL BIT-MAP INDEXING

Caché uniquely provides Transactional Bit-Map Indexing, which can radically increase performance of complex queries giving fast data warehouse query performance on live data.

Traditional Index on Property: "Type of Pet"

Database performance is critically dependent on having indexes on properties that are frequently used in searching the database. Most databases use indexes that, for each possible value of the column or property, maintain a list of the IDs for the rows/objects that have that value.

Bit-Map Index on Property: "Hair Color"

A bit-map index is another type of index. Bit-map indexes contain a separate bit-map for each possible value of a column/property, with one bit for each row/object that is stored. A 1 bit means that the row/object has that value for the column/property.

The advantage of bit-map indexes is that complex queries can be processed by performing Boolean operations (AND, OR) on the indexes – efficiently determining exactly which instances (rows) fit the query conditions, without searching through the entire database. Bit-map indexes can often boost response times for queries that search large volumes of data by a factor of 100 or more.

Bit-maps traditionally suffer from two problems: a) they can be painfully slow to update in relational databases, and b) they can take up far too much storage. Thus, with relational databases, they are rarely used for transaction processing applications.

Caché has introduced a new technology – Transactional Bit-Map Indexing – that leverages multidimensional data structures to eliminate these two problems. Updating these bit-maps is often faster than traditional indexes, and they utilize sophisticated compression techniques to radically reduce storage. Caché also supports sophisticated "bit-slicing" techniques. The result is ultra fast bit-maps that can often be used to search millions of records in a fraction of a second on an online transaction-processing database. Business intelligence and data warehousing applications can work with "live" data.

Caché offers both traditional and transactional bit-map indexes. Caché also supports multi-column indexes. For example, an index on State and CarModel can quickly identify everyone who has a car of a particular type that is registered in a particular state.

THE CACHÉ ADVANTAGE

Radically Faster Queries:
By using transactional bit-map techniques, users can get blazing fast searches of large databases – often millions of records can be searched in a fraction of a second – on a system that is primarily used for transaction processing.

Real-Time Data Analytics: Caché's Transactional Bit-Map Indexing allows real-time data analytics on up-to-the-minute data.

Lower Cost: There is no need for a second computer dedicated to data warehouse and decision support. Nor is there any need for daily operations to transfer data to such a second system or database administrators to support it.

Scalability: The speed of transactional bit-maps enhances the ability to build systems with enormous amounts of data that need to be maintained and periodically searched.

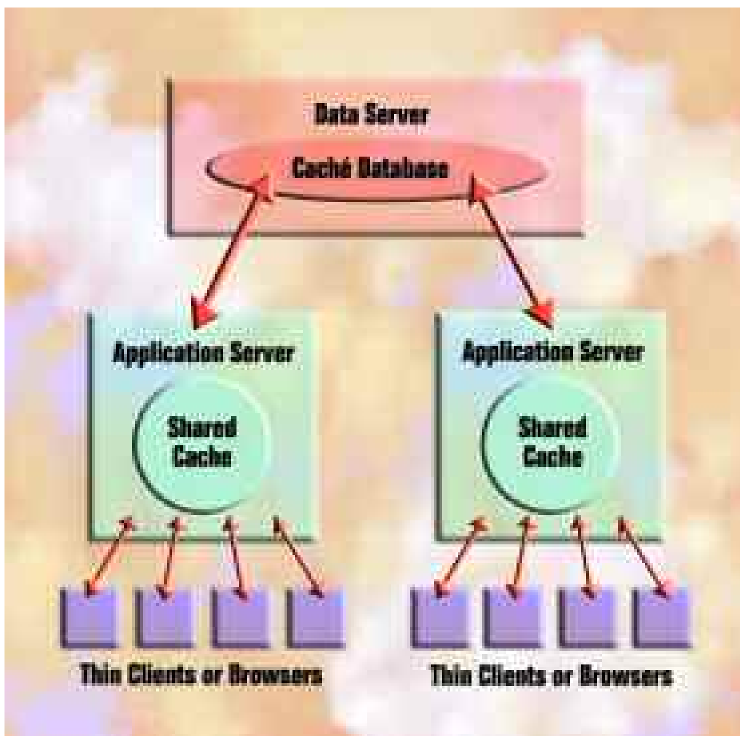
ENTERPRISE CACHE PROTOCOL FOR DISTRIBUTED SYSTEMS

Scalable Performance in Distributed Systems

InterSystems' Enterprise Cache Protocol (ECP) is an extremely high-performance and scalable technology that enables computers in a distributed system to use each other's databases. The use of ECP requires no application changes – applications simply treat the database as if it was local.

Here's how ECP works: Each Caché Application Server includes its own Caché Data Server, which can operate on data that resides in its own disk systems or on blocks that were transferred to it from another Caché Data Server by ECP. When a client makes a request for information that is maintained on a remote Data Server, the Application Server will attempt to satisfy the request from its local cache. If it cannot, it will request the necessary data from the remote Data Server. The reply includes the database block(s) where that data was stored. These blocks are cached on the Application Server, where they are available to all applications running on that server. ECP automatically takes care of managing cache consistency across the network and propagating changes back to Data Servers.

Enterprise Cache Protocol



The performance and scalability benefits of ECP are dramatic. Clients enjoy fast responses because they frequently use locally cached data. And caching greatly reduces network traffic between the database and application servers, so any given network can support many more servers and clients. However, while most applications benefit from ECP, there are some whose architecture does not readily support such scaling. Benchmarking is recommended, and often a few simple changes will speed performance.

Easy to Use – No Application Changes

The use of ECP is transparent to applications. Applications written to run on a single server run in a multi-server environment without change. To use ECP, the system manager simply identifies one or more Data Servers to an Application Server and then uses Namespace Mapping to indicate that references to some or all global structures (or portions of global structures) refer to that remote Data Server.

Configuration Flexibility

Every Caché system can function both as an Application Server and as a Data Server for other systems. ECP supports any combination of Application Servers and Data Servers and any point-to-point topology of up to 255 systems.

THE CACHÉ ADVANTAGE

Massive Scalability: Caché's Enterprise Cache Protocol allows the addition of application servers as usage grows, each of which uses the database as if it was a local database. If disk throughput becomes a bottleneck, more Data Servers can be added and the database becomes logically partitioned.

Higher Availability: Because users are spread across multiple computers, the failure of an Application Server affects a smaller population. Should a Data Server "crash" and be rebooted, or there is a temporary network outage, the Application Servers can continue processing with no observable affects other than a slight pause. Configuring Data Servers as a fail-over hardware cluster with backup data servers can significantly enhance availability.

Lower Costs: Large numbers of low cost computers can be combined into an extremely powerful system supporting massive processing – "grid computing".

Transparent Usage: Applications don't need to be written specifically for ECP – Caché applications can automatically take advantage of ECP without change.

FAULT TOLERANCE

Even in the most rigorous environments unexpected events can occur – hardware failure, power loss, or something as severe as a flood or other natural disaster – yet hospitals, telecommunications, and other critical operations cannot afford to be "down". To meet such exacting standards, Caché is designed to recover gracefully from outages and offers a variety of fail-over and other options to reduce or eliminate the impact on users.

Caché Write-Image Journaling and other integrity features assure database integrity for most types of hardware failures – including power outages – allowing rapid recovery while minimizing the impact on users.

Caché also provides advanced high-availability configuration options to further reduce or eliminate user impact, including:

- Fail-over Clusters
- Shadow Servers
- Distributed ECP

Fail-over Clusters

Using fail-over clustered hardware, data servers share access to the same disks, but only one is actively running Caché at a time. If the active server fails, Caché is automatically started on another server that takes over the processing responsibilities. The users can immediately sign back on to the new server.

Shadow Servers

Caché Shadow Servers are backup servers that are "loosely connected" through TCP. The primary server is constantly sending a logical record of database updates to the Shadow Server so that the Shadow Server always has an "almost-up-to-date" copy of the database. Switching to the Shadow is less automated than with fail-over clusters, but survivability is improved because the hardware is not physically connected – the Shadow Server may even be at a different location.

A Shadow Server can be mixed with a Fail-over Cluster, further enhancing fault tolerance.

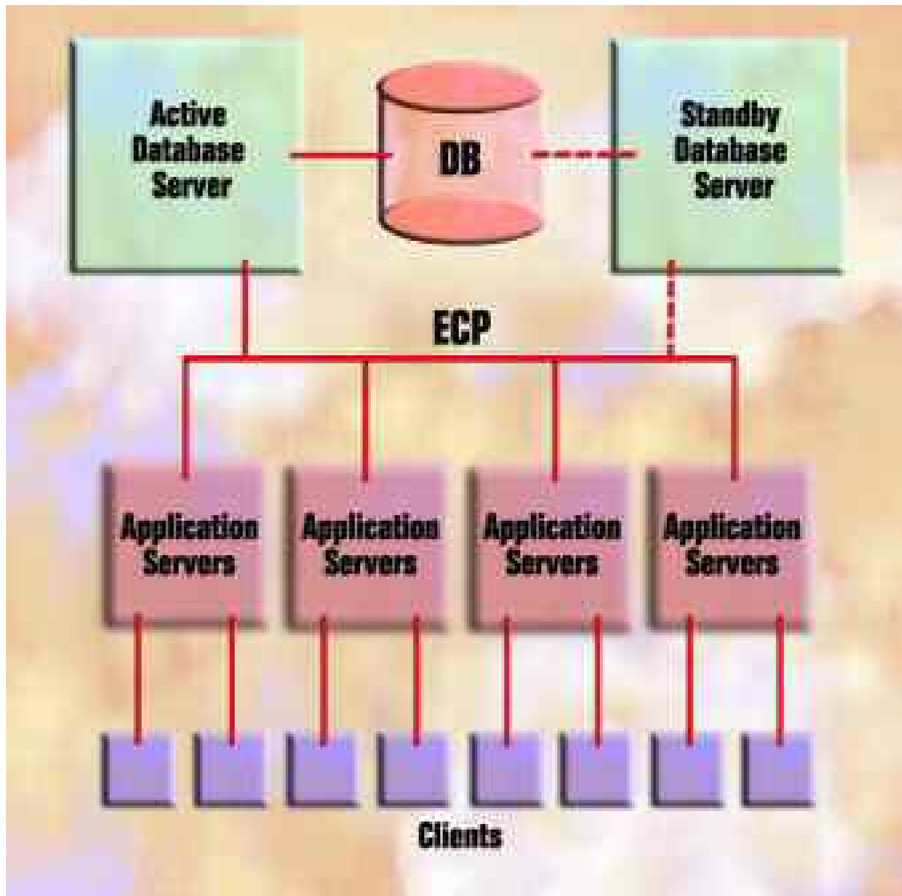
Distributed ECP

For distributed systems using ECP, upon a temporary network outage or a Data Server crash and reboot, the Application Servers attempt to reconnect. If a successful reconnect occurs within a specified time period, the Application Servers resend any uncompleted requests and operations continue with no observable effect to users other than a slight pause.

If an ECP Application Server fails, only the users on the failed Application Server are affected. They can then sign on to another Application Server to continue working.

An ECP Data Server is frequently configured as a Fail-over Cluster. If the primary Data Server crashes, the backup Data Server takes over for the failed Data Server, allowing uninterrupted operation with users experiencing only a slight pause.

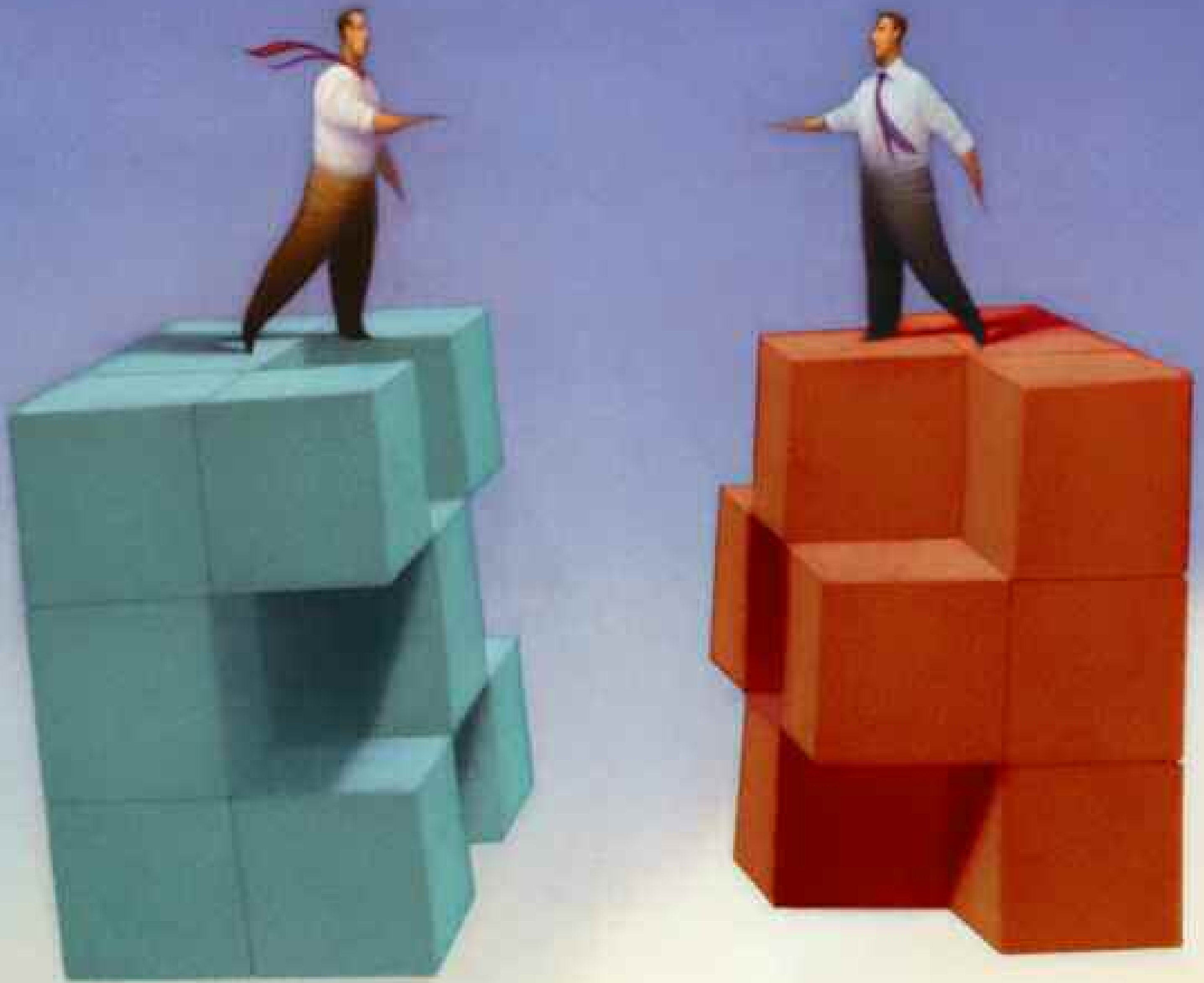
An ECP Fail-over Cluster



THE CACHÉ ADVANTAGE

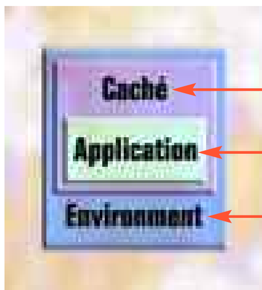
Bullet-Proof Database: Caché Write-Image Journaling and other integrity features assure database integrity for most types of hardware failures, including power outages.

High-Availability Fault-Tolerant Configurations: The use of Caché Shadow Servers, ECP, and/or Fail-over Clusters allow rapid recovery from outages while minimizing, or in some cases eliminating, their impact on users.



SECURITY MODEL

Caché has a modern security model, designed to support application development in three ways:



By securing the Caché environment itself.

By making it easy for developers to build security features into their applications.

By ensuring that Caché works effectively with – and does not compromise – the security technologies of the operating environment.

Caché provides these security capabilities while minimizing the burden on application performance.

Users, Roles, Resources, and Privileges

There are a variety of resources (such as databases, applications, and system services) and users must be granted permission (such as READ, WRITE, or USE) to use them by the Security Administrator. In addition to the system-defined resources, the Security Administrator can create application-specific resources and use the same mechanisms for granting and checking permissions.

For simplicity, users are usually assigned one or more "roles" (e.g., "LabTech", or "Payroll"), and the Security Administrator then grants privileges for a particular resource to those roles rather than to individual users. The user inherits all of the privileges granted to the roles it is assigned.

Every process has an associated username, even if it is only "UnknownUser". The username is established during "authentication". A simple example of authentication is when a user enters a username and password and the system checks to see that the correct password was entered. Following authentication, the username is assigned to the process and the permissions associated with that username are granted. (A "user" is not necessarily a human being. It could, for example, be a measurement device generating data or an application running on another system that is connected to Caché.) If a user does not go through authentication, it has a username of "UnknownUser", which only entitles that process to the permissions granted to everyone.

Connection to Caché is controlled by a set of Services. Each Service specifies whether it is Public – which means anyone can use it – or whether it requires authentication and, following authentication tests, whether the user has the required access privilege. Services can also be individually disabled, so that access is denied to everyone.

The assignation and management of privileges is normally performed through the Caché Management Portal.

Application-Assigned Roles

It is often useful for a user to temporarily gain additional privileges rather than have them permanently assigned. For example, rather than the Security Administrator granting a broad set of privileges to a user (such as the ability to access and modify the payroll database), the user can instead be given just the privilege to access the payroll application, and that application can then elevate the user's privileges while that application is being used.

To accomplish this elevation, roles can be assigned to applications. When that application is accessed, the user temporarily acquires additional roles. The additional roles may be simply a list that everyone authorized to use the application acquires, or the additional roles may be more customized, based on the roles the user already has.

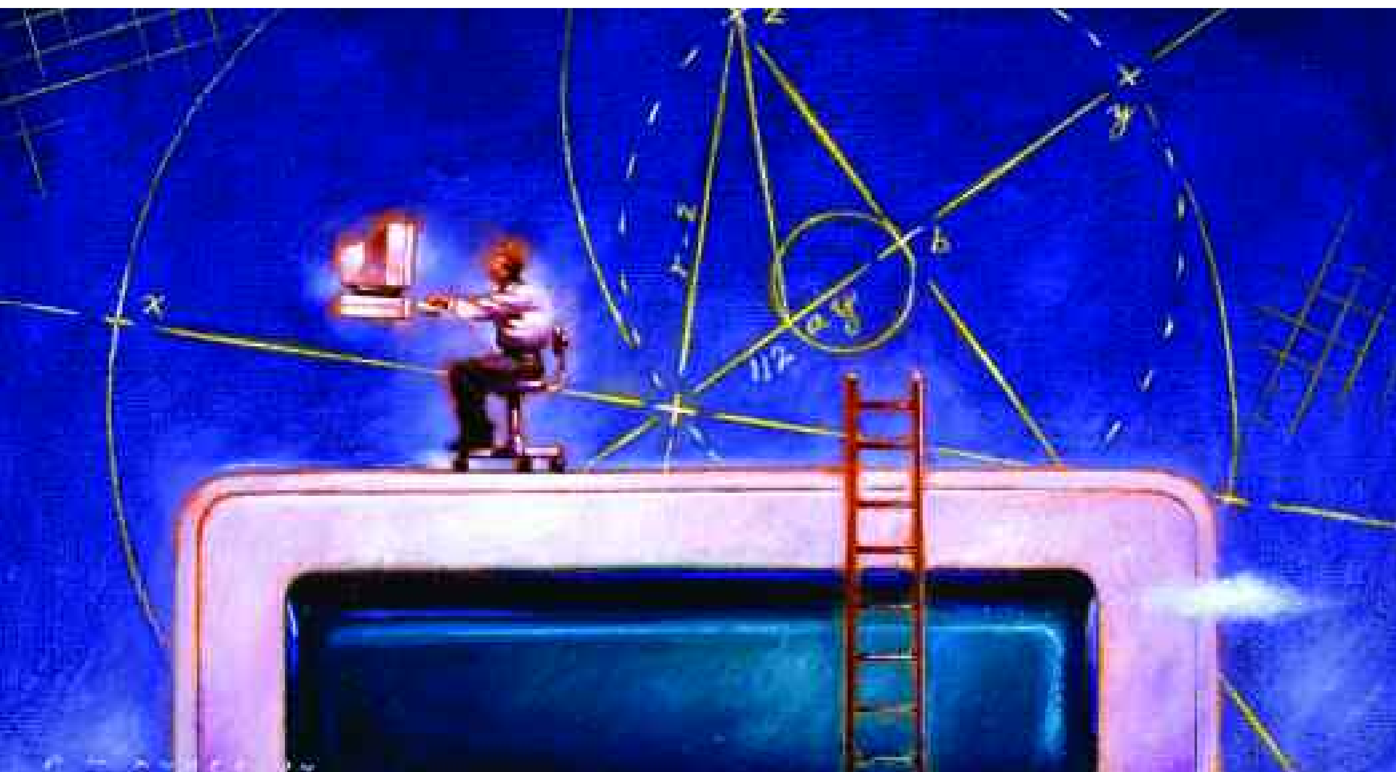
This feature is particularly useful for browser-based applications using CSP (Caché Server Pages). With CSP, a portion of every URL specifies an application name. Following authentication and a determination that the user is authorized to use that CSP application, the user temporarily gains the additional roles assigned to that application for the duration of that page request.

The Security Administrator can also designate specific routines as capable of performing role elevation to gain the additional roles of specified applications, after passing user specified security tests. This facility is tightly controlled, and it is the mechanism by which non-CSP applications perform role elevation.

Authentication

Caché supports various levels of authentication, ranging from none, to the use of passwords, to the use of the Kerberos protocol to authenticate the identity of users. Kerberos provides very strong authentication and has the advantages of being fast, scalable, and easy to use. With Kerberos, passwords are never transmitted over the network, which provides an extra measure of security.

Caché supports the implementation of a single sign-on.



Database Encryption

Caché supports two forms of database encryption:

- The Security Administrator can designate one or more CACHE.DAT files (databases) to be encrypted on disk. Everything in those files is then encrypted.
- Developers can use system functions to encrypt/decrypt data, which then may be stored in the database or transmitted. This feature can be used to encrypt sensitive data to protect it from other users that have read access to the database but not the key.

By default, Caché encrypts data with an implementation of the Advanced Encryption Standard (AES), a symmetric algorithm that supports keys of 128, 192, or 256 bits. Encryption keys are stored in a protected memory location. Caché provides full capabilities for key management.

The journal can also be encrypted.

Auditing

Many applications, especially those that must comply with government regulations like HIPAA or Sarbanes-Oxley, need to provide secure auditing. In Caché, all system and application events are recorded in an append-only log, which is compatible with any query or reporting tool that uses SQL.

Chapter Three: Caché's Application Server

The Caché Application Server offers advanced object programming capabilities, provides sophisticated data caching, and integrates easy access to a variety of technologies. The Caché Application Server makes it possible to develop sophisticated database applications rapidly, operate them with high performance, and support them easily.

More specifically, the Caché Application Server provides:

- Access to Caché Multidimensional Data Servers on the same and other computers with transparent routing.
- Connectivity software with client-side caching to permit rapid access to Caché Objects from all commonly used technologies, including Java, C++, C#, COM, .NET, and Delphi. Caché automatically performs the networking between the client and Application Server.
- Compatibility with SOAP and XML.
- SQL access using ODBC and JDBC, including sophisticated caching at the client and application server for high performance.
- Access to relational databases.
- Caché Server Pages for high-performance, easy-to-program Web applications.
- Caché Studio – an IDE to rapidly develop and debug applications with Caché.
- Code for the Scripting Languages is stored in the database and can be changed online, with changes automatically propagating to all application servers.

THE CACHÉ VIRTUAL MACHINE AND SCRIPTING LANGUAGES

The core of the Caché Application Server is the extremely fast Caché Virtual Machine, which supports Caché's scripting languages.

- Caché ObjectScript is a powerful and easy-to-learn object-oriented language with extremely flexible data structures.
- Caché Basic provides an easy way for Visual Basic programmers to start using Caché. Similar to VBScript, Caché Basic supports objects and is extended to have direct access to the Caché Multidimensional Arrays.
- Caché MVBasic is the variant of the Basic programming language used in MultiValue (Pick) applications. MVBasic has been extended to support objects and have direct access to the Caché Multidimensional Arrays.

Database access within the Caché Virtual Machine is highly optimized. Each user process in the Caché Virtual Machine has direct access to the multidimensional data structures by making calls to shared memory that access a shared database cache. All other technologies (Java, C++, ODBC, JDBC, etc.) connect through the Caché Virtual Machine to access the database.

Complete Interoperability

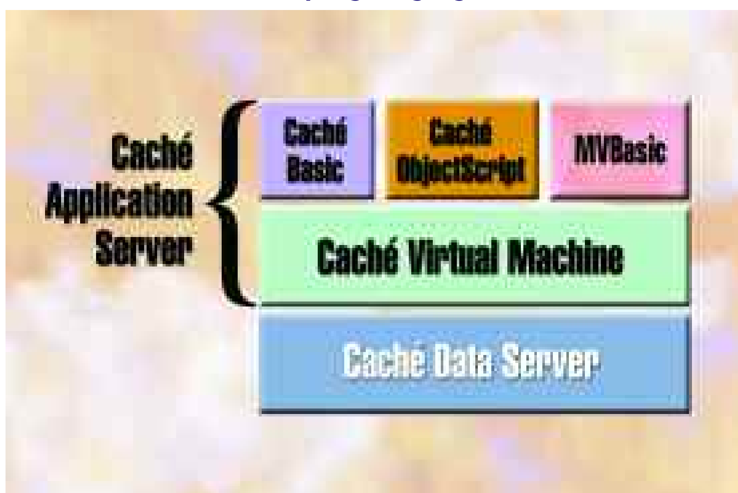
Since Caché ObjectScript, Basic, and MVBasic are all implemented on the same Caché Virtual Machine, they are completely interoperable:

- Any object method can be written in any language – the same class can use all three languages.
- Each language's function calls can access code written in the other languages.
- They share variables, arrays, and objects.

Faster Development/Flexible Deployment

In almost all cases, programmers can develop applications faster, and those applications will run significantly faster with greater scalability, by writing as much code as possible in these scripting languages so that they run in the Caché Virtual Machine. Plus, such code requires no changes to switch hardware or operating systems. Caché automatically handles all differences in operating system and hardware.

Scripting Languages



THE CACHÉ ADVANTAGE

Rapid Application Development: Development of complex database applications with Caché ObjectScript is radically faster than any other major language – often 10 to 100 times faster. Faster also means the project has a better chance of succeeding – with fewer programmers – and being able to adjust rapidly as application needs change.

Shorter Learning Curve: Basic is perhaps the world's best-known computer language. Developers who know Visual Basic can instantly start writing code in Basic, and the Caché object model is easily learned.

Faster and More Scalable: The Caché Virtual Machine with its direct access to the database provides faster applications that can scale to tens of thousands of users using low-cost hardware.

Flexibility: Code that runs in the Caché Virtual Machine can run on other hardware and operating systems without change. Code is stored in the database and automatically propagated to Application Servers.

CACHÉ OBJECTSCRIPT

Caché ObjectScript is a powerful, object-oriented programming language designed for rapid development of database applications. Here are some of the key characteristics of the language.

Overall Structure

Caché ObjectScript is command-oriented; hence it has syntax such as:

```
set x=a+b
do rotate(a,3)
if (x>3)
```

There is a set of built-in system functions that are particularly powerful for text manipulation. Their names all start with the single "$" character to distinguish them from variable and array names. For example:

```
$extract(string,from,to)    // get a set of characters from a string
$length(string)             // determine the length of a string
```

Expressions use left-to-right operator precedence, just like most hand-held calculators, except when parentheses alter the order of evaluation.

Flexible Data Storage

One of the most unique characteristics of Caché ObjectScript is its highly flexible and dynamic data storage. Data may be stored in:

- Object properties.
- Variables.
- Sparse, multidimensional arrays that permit any type of data for subscripts.
- Database files ("globals") which are sparse multidimensional arrays.

With rare exceptions, any place in the language where a variable can be used, an array, object property, or global reference could also be used.

In most computer languages, datatypes are an extension of hardware storage concepts (integer, float, char, etc.). However, Caché ObjectScript has the philosophy that humans don't think using such storage types, and that these "computer-centric" datatypes simply impede rapid application development. Requiring declarations and dimension statements introduces far more errors than they help prevent (e.g.: errors such as a 2-byte integer overflow, or when a string overflows its allocation of memory and corrupts other storage). However, object typing, such as Person, Invoice, or Animal, is viewed as highly valuable and consistent with the way humans think.

Thus, in Caché ObjectScript, object properties are strongly typed, but the other three types of storage (variables, arrays, and global nodes) are fully polymorphic, typeless entities that need not be declared or defined. They simply pop into existence as they are used and mold themselves to the data needs of what they are storing and how they are being used in an expression. Even arrays do not need any specification of size or dimension or type of subscripts or data. For example, a developer might create an array called Person by simply setting:

```
set Person("Smith","John")="I'm a good person"
```

In this example, data was stored in a two-dimensional array using string data for subscripts. Other data nodes in this array might have a different number of dimensions and might intermix strings, integers, or other types of data for subscripts. For example, one might store data in:

```
abc(3)
abc(3,-45.6,"Yes")
abc("Count")
```

all in the same array.

Direct Access to the Database

A direct reference to the database (a "global reference") is essentially a multidimensional array reference preceded by the carat character "^". That character indicates this is a reference to data stored in the database rather than to temporary process private data. Each such database array is called a "global".

As with multidimensional arrays and variables, no declarations or definitions or reservations of storage are required to access or store data in the database; global data simply pops into existence as data is stored. For example, to store information in the database one might write:

```
set ^Person("Smith","John")="I'm a very good person"
```

and later might retrieve it by code such as:

```
set x=^Person("Smith","John")
```

The programmer has complete flexibility in how to structure these global data arrays. (See the Multidimensional Data Model.)

Object References

Caché Objects implement the ODMG data model, with powerful extensions.

In Caché ObjectScript, an "oref" is used to access an object. (An "oref" is typically a variable whose value specifies which in-memory object is being referenced). The oref is followed by a dot and then by the name of a property or method. Object references can be used wherever an expression can be used. For example:

```
set name=person.Name      // "person" is a variable whose value is an oref
                          // the person's name is put into the variable "name"
if (person.Age>x)         // see if the person's age is greater than "x"
set money=invoice.Total() // "Total()" is a method that calculates the sum of
                          // all of the invoice's line items
```

Methods can also be executed with a DO command when no return value is needed. For example:

```
do part.Increment()       // "Increment()" is a method whose return value,
                          // if any, is not of interest
```

The oref is not the same as a database object ID. The object ID is a value that is permanently associated with a database object; it is used to retrieve and store a database object. Once an object is in memory, it is assigned a reusable oref value that is then used to access the object's data. The next time that same database object is brought into memory it will probably be assigned a different oref value.

HTML and SQL Access

HTML for Web applications and SQL can be embedded in Caché ObjectScript code.

Calling Code

In some object languages, all code has to be part of some method. Caché ObjectScript doesn't have that restriction – code may be directly called or called through object syntax.

Code is often called using the DO command.

```
do rotate(a,3)
```

Code that returns a value can also be called as a function. For example,

```
set x=a+$$insert(3,y)
```

calls the programmer-written procedure or subroutine "insert".

Code can also be invoked as an object method.

```
set money=invoice.Total()    // Total() returns the invoice total amountdo
part.Increment()             // "Increment()" is a method whose return value,
                             // if any, is not of interest
```

Both call by value and call by reference is supported for parameters.

Routines

Caché ObjectScript code is fundamentally organized into a set of "routines". Each routine (typically up to 32KB in size) is atomic in the sense that it can be independently edited, stored, and compiled. Routines are linked dynamically at run time; there is no separate linking step for the programmer. Routine code is stored in the database; thus, routines can be dynamically paged across the network rather than having to be installed on each computer.

Within a routine, code is organized as a set of procedures and/or subroutines. (An object method is a procedure, but it is accessed by a different syntax.)

When calling code that is within the same routine, only the procedure or subroutine name is needed. Otherwise, the routine name must be appended to it.

```
do transfer()              // calls "transfer" in the same routine
do total^invoice()         // calls "total" in the routine "invoice"
```

A procedure or subroutine that has a return value of interest should be called using the "$$" function syntax.

```
set x=$$total^invoice()    // calls the same "total" procedure but uses the
                           return value
```

Routines can be edited and compiled through the Caché Studio.

Object Methods

Class definitions and their method code are stored in global data files, and the Class Compiler compiles each class into one or more routines. Each method is simply a procedure in a routine, although it can only be invoked by object syntax. For instance, if the Patient class defines an Admit method and the Pat variable identifies a specific Patient object, then we call the Admit method for that object with the following syntax:

```
do Pat.Admit()          // Call the admit method for Patient
set x = Pat.Admit()     // Calls the same method but uses the return value
```

Procedures and Public/Private Variables

A procedure is a block of code within a routine that is similar to a function in other languages. A procedure consists of a name, a formal parameter list, a list of public variables, and a block of code delimited by "{}". For example:

```
Admit(x,y)[name,recnum] { ...code goes here
                        }
```

In Caché ObjectScript, some variables are public (common) and others are private to a particular procedure. Every variable that is used within a procedure is considered private to that procedure unless it is listed in the public list. In the above example, "name" and "recnum" access the public variables by those names, whereas all other variables exist only for this invocation of this procedure. Variable names that start with a "%" character are always implicitly public.

Procedures cannot be nested, although a procedure can contain subroutines.

Subroutines

Routines may also contain subroutines, which are lighter weight than procedures. A subroutine may contain a parameter list and it may return a value, but it does not have a public list or formal block structure. Subroutines may be embedded within procedures or be at the same level as a procedure in a routine.

Subroutines permit the calling of code using the same public/private set of variables as the caller, and they can be called quicker. A subroutine embedded within a procedure uses the same variable scope as the procedure and may only be called from within that procedure. Variable references for a subroutine that is not part of a procedure are all to public variables.

BASIC

Basic is perhaps the world's best-known application programming language. In Caché, Basic has been extended to support direct access of the Data Server's core data structures – multidimensional arrays – as well as other Caché Application Server features. It directly supports the Caché ObjectModel using Visual Basic syntax, and runs in the Caché Virtual Machine.

Basic can be used either as methods of classes or as Caché routines (see the Caché ObjectScript description of routines). Basic can call Caché ObjectScript, and vice versa, with both languages accessing the same variables, arrays, and objects in process memory.

Arrays have been extended to be far more powerful:

- The presence of a "^" character preceding the array name indicates a reference to a database multidimensional array – persistent arrays that are shared with other processes.
- Subscripts can be of any datatype – strings, integers, decimal numbers, etc.
- Data can be stored at multiple subscript levels in the same array – for example, data could be stored at A("colors") and A("colors",3).
- Arrays do not have to be declared and they are always sparse – Caché only reserves space as nodes are inserted.
- A Traverse function allows identification of the next (or previous) subscript at a given subscript level.

Other extensions include:

- Transaction processing commands to Start, Commit, and Rollback a transaction.
- An atomic increment function that can be used on the database.
- Extensions that provide better integration with the Caché Application Server capabilities.

Object Access with Basic

In Caché, classes are organized into packages, and class names include the package name followed by a period. For example, Payroll.Person is a Person class of the Payroll package. The Basic New command is used to create an object:

```
person = New Payroll.Person()      // creates a new Person object
```

Basic has been extended with an OpenID command to access an existing object:

```
person = OpenID Payroll.Person(54) // opens the Person object with OID 54
```

Here are some examples of code that access the person's properties:

```
person.Name = "Smith, John"        // sets the person's name
person.Home.City                   // references the person's home city
person.Employer.Name               // brings the person's employer object into
                                   // memory and accesses the employer's name
```

Database classes can be saved to disk with the Save method. For example:

```
person.Save()
```

will save the person, creating an object ID if it is the first time the object was stored. If related objects (such as the Employer) were also modified, they are automatically saved as well.

MVBASIC

MVBasic is another scripting language offered in Caché that is a variant of Basic. However, it is intended to execute applications written for MultiValue (Pick) systems and therefore supports additional characteristics, including capabilities to access and manipulate MultiValue files.

MVBasic can be used either as methods of classes or as Caché routines (see the Caché ObjectScript description of routines). MVBasic can call Caché ObjectScript or Basic, and vice versa, with all three languages accessing the same variables, arrays, and objects in process memory.

Caché MVBasic has the same extensions as Caché Basic, including object access. However, because of possible ambiguity, the two character sequence "->" is used instead of using a period separator "." in object references.

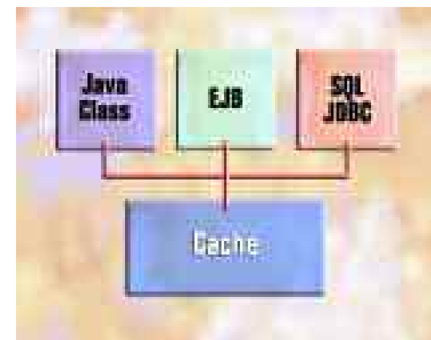
C++

Every Caché class can be projected as a C++ class, with methods corresponding to each property and method of the Caché class. To C++ programs, these classes look just like any other local C++ classes, and Caché automatically handles all communications between the client and server. Properties of the class are cached on the client, and C++ method calls invoke corresponding server side methods, including methods to store an object in the database and later retrieve it.

JAVA

Java is a popular programming language, but connecting Java applications to most databases can be challenging. Connecting to a relational database requires extensive coding of SQL – which is very time-consuming and blunts many of the advantages of Java's object technology. Caché's approach of directly storing objects without the developer worrying about how the data will be persisted and using object syntax to access the database is much simpler – and usually preferred.

Java Supported Several Ways



Some developers want to work exclusively with "plain old Java objects" (POJOs), while others prefer Enterprise Java Beans (EJB). Also, some developers prefer to first define the database schema and then automatically generate a corresponding Java class for each database class, while others prefer to first create Java classes and have Caché automatically generate a database schema. Caché supports all of these approaches:

- Any Caché class can be projected as a Java class so properties and methods can be accessed as Java objects.
- Caché classes can also be projected as Enterprise Java Beans.
- JDBC provides high-performance SQL access using a completely Java-based (Type 4) driver.
- InterSystems Jalapeño technology creates Caché classes from POJO class descriptions.

Object Access Through Projected Classes

Every Caché class can be projected as a Java (or EJB) class, with methods corresponding to every property and method of the Caché class. To Java programs, these classes look just like any other local Java classes. The generated Java class uses a Java library provided by InterSystems to handle all communications between the client and server.

State for every Caché object is maintained in the Caché Application Server, although properties of the class are also cached in the client to optimize performance. Java method calls invoke corresponding methods on the Caché Application Server – including methods to store an object in the database and later retrieve it. It is transparent to the client as to which Caché Data Server contains the data, or even if the object data is stored in a relational database accessed through the Caché Application Server.

Caché Methods Written in Java

Methods of Caché classes can be written in Java using Caché Studio. However, unlike Caché ObjectScript and Basic, Java methods are not executed by the Caché Virtual Machine. Instead, they are included in the generated Java class and executed in any Java Virtual Machine. Such code is not accessible from non-Java methods.

Providing Persistence for J2EE Applications

Developers of J2EE applications, which use Enterprise Java Beans (EJB), work primarily with objects until such time as they need to access the database. Then, they are usually forced to revert to using SQL. Through its JDBC interface, Caché can provide extremely fast SQL response to such applications. However, SQL access is not generally the preferred approach.

Object databases represent a more natural access technique to EJB programmers. Caché projects Caché classes as EJBs, automatically generating high-performance persistence methods for Bean-Managed Persistence (BMP). This avoids the overhead of SQL and object/relational mapping – the result is higher scalability for J2EE applications.

InterSystems Jalapeño technology can also be used in J2EE applications with the same advantages.

THE CACHÉ ADVANTAGE

Flexibility: Java developers have choices when it comes to accessing Caché objects – they can use SQL and JDBC or more naturally project objects as Java classes or Enterprise Java Beans. With Jalapeño, developers have the option of working entirely within their favorite Java development environment and letting Caché automatically provide methods to store and retrieve objects from the database without touching the developer's classes.

High Performance: All Java applications, regardless of how they are connected to Caché, benefit from Caché's superior performance and scalability.

Native Compatibility with J2EE Means More Rapid Development: Caché classes can easily be projected as EJBs, giving J2EE developers a simple way to connect to Caché's post-relational database. When a Caché class is projected using bean-managed persistence, Caché automatically generates the method used by the EJB to access the Caché database. Because developers no longer have to hand code persistence methods, applications can be completed more quickly.

Jalapeño Allows Java-in Development

Instead of starting with Caché classes and projecting them as Java components, InterSystems Jalapeño technology does the opposite. It allows Java developers to define object classes within whatever Java development environment they favor and automatically persist those classes in Caché. The developer's Java class is unchanged – Caché provides a library class with an API that is used to store and retrieve objects and issue queries for the developer's classes.

CACHÉ AND JALAPEÑO

Java developers who wish to create new database applications are faced with several problems today. Typically, they store their data in a standard relational database using SQL. With that approach, developers have to map their Java objects to relational structures and write tedious, and often complex, SQL queries to access that data – all of which can easily consume a high percentage of the total development time. Alternatively, a developer can make use of an object database, which often cannot support SQL and SQL-based reporting tools and may also require schema definitions.

Jalapeño (**JA**va **LA**nguage **PE**rsistence with **NO** mapping) is an InterSystems technology that makes it easy for Java developers to persist their objects within the robust Caché object database using object access while simultaneously providing high-performance SQL access to the same data. Objects are stored in the database as true objects with properties, relationships, etc. (i.e., not by simply storing their serialized state), yet no object-relational mapping is required.

Using Jalapeño, the Java developer creates database classes the same way as any other POJO class, using the developer's Java IDE of choice. The developer then indicates to Jalapeño which classes are database classes – usually by using a plug-in to the developer's IDE provided by the Jalapeño Persistence Library. Jalapeño analyzes the classes, automatically creates a corresponding object (and SQL) database schema, and generates all of the runtime support for saving and retrieving such objects.

The developer's POJO class is not modified, and the developer can continue to change it.

At run time, the application directly accesses properties and methods of the POJO objects in the usual way. To save and retrieve database objects, the application uses the APIs of the "ObjectManager" class provided by Jalapeño. The ObjectManager class also provides methods that establish a connection to the database, support SQL queries, and provide transaction semantics such as start, commit, and rollback.

A simple class description, including a list of properties and their type, is not by itself rich enough to describe everything you might want in a database. At a minimum it is necessary to also specify which property contains the object ID, and usually a developer would also like to specify indices to make queries more efficient. By adding standard Java annotations to their Java source files, developers can supply these and other database specifications.

Jalapeño also supports "schema evolution", in which the developer can continue to modify the class, including adding new properties or changing property definitions, and the schema definition adapts without invalidating any of the already entered data. This results in a natural, iterative development process.

While Jalapeño works best when used with Caché, it can also export its database schema to a corresponding relational schema using standard DDL (Data Definition Language). Therefore, even though the application was built to use object access with Caché, it can also be deployed on a relational database. In this case, the Jalapeño ObjectManager APIs automatically use standard JDBC calls for database connectivity. When connected to the Caché object database, the higher-performance object-based protocol is used.

The Jalapeño library is implemented using standard Java and will run within any Java 1.5 (or higher) JVM or J2EE Application Server environment.

With Jalapeño, the Java developer can concentrate on the UI and business logic of the application, creating database classes the same way as other classes, and let Caché take care of the rest.

In the following example, a Customer object is retrieved, its phone number is changed with a "set" method, the database is updated, and the in-memory object is closed:

```
Customer customer = (Customer) objectManager.openById(Customer.class, customerId);
customer.setPhoneNumber("16176210600");
objectManager.update(customer, true);
objectManager.close();
```

THE CACHÉ ADVANTAGE

Fast, Natural Development with No Object-Relational Mapping: Jalapeño uses introspection of POJO classes to automatically create an object database schema. Data is stored as objects and a standard SQL representation of this data is also created automatically. No object-relational mapping is required, speeding development.

Easy POJO Persistence: Within an application, developers access their database classes just like any other class. Jalapeño generates all of the code to save and retrieve objects from the database using its runtime APIs.

SQL Access: All objects within the database are automatically accessible through SQL using Jalapeño's JDBC API – even though the developer did not perform any object-relational mapping.

Database and Platform Independence: Caché is available on every major platform, and while Jalapeño works best when used with Caché, it can also export its database schema to a corresponding relational schema using standard DDL. Therefore, even though the application was built to use object access with Caché, it can also be deployed on a relational database.

THE CACHÉ ADVANTAGE

Speedy Data Serving: Web applications that use Caché as a data server benefit from the high-performance and massive scalability provided by Caché's multidimensional data engine.

Faster .NET Development: Developers will be more productive when they work with their favorite tools in environments that are familiar to them. Providing both SQL and object data access, Caché supports a wide variety of common development technologies and tools.

CACHÉ AND .NET

Because of its open and flexible data access, Caché works seamlessly with .NET. There are many ways of connecting the two, including objects, SQL, XML, and SOAP. Developers can create applications with the technologies they prefer – all of them will benefit from Caché's superior performance and scalability.

ADO.NET

ADO.NET is a new incarnation of ADO, optimized for use in the .NET framework. It is intended to make .NET applications "database independent", and generally uses SQL to communicate with databases. Through its relational data access, Caché provides native support for ADO.NET. It also supports Microsoft's ODBC.NET and the read-only SOAP connectivity that is built into ADO.NET.

Web Services

There are two ways of using Web services in .NET. One is to send XML documents over HTTP. The other is to use the SOAP protocol to simplify the exchange of XML documents. Because Caché can expose data both ways, it works seamlessly with .NET Web services.

Caché Managed Objects

Caché can automatically generate .NET assemblies (or C# source code) from Caché classes. A plug-in for Visual Studio lets developers who prefer that environment to easily access Caché objects.

CACHÉ AND XML

Just as HTML is an Internet-compatible mark-up language for displaying data in a browser, XML is a mark-up language for exchanging data between applications. The structure of XML data is hierarchical and multidimensional, making it a natural match for Caché's multidimensional data engine.

Exporting XML

All that is required to make a Caché class compatible with XML is to have it inherit from the %XMLAdaptor class that is included in Caché. This provides all the methods needed to:

- Create either a DTD (Document Type Definition) or an XML Schema for the class. Caché will automatically generate DTDs and Schemas, but developers who wish to customize the XML formatting of a class may do so.
- Automatically format an object's data as XML, according to the defined DTD or Schema.

Importing XML

Caché comes with other classes that provide methods allowing developers to:

- Import XML Schemas and automatically create corresponding Caché classes.
- Import the data in XML documents as instances (objects) of Caché classes, via a simple API.
- Parse and validate XML documents via a built-in XML (SAX) parser.

CACHÉ AND WEB SERVICES

Web services enable the sharing of application functionality over the Internet, as well as within an organization or system. Web services have an interface described in WSDL (Web Service Definition Language), and they return an XML document formatted according to the SOAP protocol.

Caché enables any class method, any SQL-stored procedure, and any query to be automatically exposed as a Web service. Caché generates the WSDL descriptor for the service and, when the service is invoked, sends the appropriately formatted XML document. Caché also facilitates rapid development by automatically generating a Web page to test the service, without the need to construct a client application.

THE CACHÉ ADVANTAGE

Easy Connectivity to XML: Caché takes advantage of its capability for multiple inheritance to provide any Caché class a bi-directional interface to XML. The result: Caché classes can easily and quickly be turned into XML documents and schemas. Similarly, XML schemas and documents can be turned into Caché class definitions and objects.

Rapid Development of Faster XML Applications: Because Caché's native multidimensional data structures are a good match to XML documents, there is no need for developers to manually code a “map” that translates between XML and the Caché database. And with less processing overhead, they run faster, too.

Instant Web Services: Any Caché method can be published as a Web service with just a few clicks of a mouse. Caché automatically generates the WSDL descriptor and the SOAP response when the service is invoked.

CACHÉ AND MULTIVALUE

Caché provides all the capabilities needed to develop and run MultiValue applications (sometimes referred to as Pick-based applications), including the MultiValue:

- MVBasic language
- File access
- Query language
- Data dictionary
- "procs"
- Command shell

This MultiValue functionality is provided as an integral part of Caché – not as a separate MultiValue implementation – and it utilizes the rich Caché multidimensional database engine, runtime functionality, and development technologies. This means that MultiValue users can take full advantage of all the Caché capabilities.

MultiValue File Access

MultiValue applications typically treat the database as a set of files accessed through Read and Write record operations and through MultiValue queries. In Caché, each MultiValue file is stored as a multidimensional "global" structure with each record being one global node. This feature relies on Caché's ability to store multiple data elements per global node.

For example, a MultiValue file **that stores invoices might** have the following structure:

```
Invoice #               Item ID
Customer                Attribute 1
InvoiceDate             Attribute 2
Parts                   Attribute 3 (MultiValued)
Quantities              Attribute 4 (MultiValued)
Prices                  Attribute 5 (MultiValued)
  …                       …
```

Caché will internally represent this MultiValue file as the equivalent multidimensional global structure:

```
^Invoice(invoice #) =
Customer ^ InvoiceDate ^ PartNo1 ] PartNo2 ^ Quantity1 ] Quantity2 ^ Price1 ] Price2k
```

where "^" indicates the normal attribute delimiter (ASCII 254), and "]" indicates the subattribute delimiter (ASCII 253).

MultiValue files can be accessed by MultiValue programs through the normal READ/WRITE commands and MultiValue queries. They are also accessible by both MVBasic and other languages through all of the normal Caché mechanisms, including object access, direct multidimensional array access, and SQL.

MultiValue files may be indexed with a variety of index types (including bit-map indexes) and collation sequences.

MultiValue Query Language

The MultiValue Query Language provides both data selection and report formatting functionality for MultiValue files. The query language can be used in MVBasic, the command shell, and "procs". Caché translates MultiValue queries into Caché SQL queries with additional code to support correlatives, report formatting, and various other features. Because these queries use Caché's very high-performance SQL engine, reliability is enhanced, execution is optimized, and a sophisticated set of indexing capabilities can be used. Of course, MultiValue developers can also use Caché SQL directly when desired.

MultiValue Data Dictionary

A MultiValue file may have a corresponding file description in the MultiValue Data Dictionary, which is directly editable through MVBasic code and through the traditional MultiValue "ED" editor. The MultiValue Query Language makes use of this dictionary.

A MultiValue file may also have a corresponding Caché class definition. A class definition is essential if the data is to be made available for object or SQL access (although it is not necessary to use the MultiValue Query Language). A class definition is also required if the file is indexed.

When importing older MultiValue applications, Caché class definitions can be automatically created from the MultiValue Dictionary. However, in most cases the resulting class will need to be edited to make the data more meaningful if SQL or object access is desired. A Studio wizard helps automate class creation from MV Dictionaries and provide further mappings at a later date. By default, these classes are Read-Only (so that the data can be read but not updated through objects and SQL) and edited separately from the MV Dictionary.

When creating a new MultiValue file, it is recommended to first create a Caché class inheriting from the MVAdaptor super-class, which will result in using a MultiValue compatible file format for data storage and will automatically create a file description in the MultiValue Data Dictionary. The file's data will then be accessible and updateable through all of the Caché access paths, including object, SQL, and direct multidimensional global access. Thereafter, a developer would normally edit the class definition rather than the MultiValue Dictionary, as edits to the class are automatically reflected in the MultiValue Dictionary.

MultiValue and Objects

MVBasic has been extended to use objects in the same way as Basic does, except MVBasic uses the "->" syntax to represent object access rather than a period. Any class in the Class Dictionary can be used, regardless of the language used in the methods of the class.

Here are some examples of code in which "person" is an object reference:

```
person->Name = "Smith, John"    // sets the person's name
person->Home->City              // references the person's home city
person->Employer->Name          // brings the person's employer object into
                                // memory and accesses the employer's name
person->Save()                  // saves the person to disk
```

MultiValue Command Shell

The MultiValue command shell can be run from a terminal environment. In addition to the normal MultiValue command shell capabilities, Caché allows MVBasic commands to be directly executed in the command shell. For example, typing:

```
:; DIM A(34)
:; FOR I = 1 TO 34 ; A(I) = I; NEXT
:; FOR I = 1 TO 34 ; CRT A(I):" ": ; NEXT
```

yields the result:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
```

THE CACHÉ ADVANTAGE

New Life for Old Applications: Caché makes it easy to modernize older MultiValue applications through browser interfaces, object access, robust SQL, and Web services. MultiValue applications can now live on an advanced database that is well accepted in demanding environments and constantly evolving.

Build New Applications Fast: Because MultiValue is implemented as a language and file access in Caché, all of the native Caché capabilities can be used to rapidly build new functionality. MultiValue programmers can begin to take advantage of object programming and easily interact with other applications while continuing to use the MVBasic language.

High-Performance and Scalability with Outstanding Reliability for MultiValue Users: Caché provides dramatically higher performance and scalability for MultiValue users. Plus, Caché is used in critical 24-hour environments, such as hospitals, with no tolerance for downtime. Caché provides sophisticated journaling, transaction processing, "bullet-proof database", and fault-tolerant configurations.

Chapter Four: Building Fast Web Apps Fast With Caché Server Pages (CSP)

"Building fast Web apps fast" uses the word fast two times. That's because it is possible to build sophisticated database-oriented Web applications with CSP more rapidly than with traditional approaches, and also because the built-in Caché database is the world's fastest, capable of running systems with tens of thousands of simultaneous users.

There are many ways to write Web applications with Caché – including all of the traditional ways that use SQL to access the database. In this chapter, we're going to discuss another, more direct approach called Caché Server Pages (CSP).

CSP is a technology that is provided as part of the Caché Application Server. It is the fastest way for Caché applications to interact with the Web, providing:

- An advanced object-oriented development approach.
- Ultra-high performance and scalability at run time.

CSP supports HTML, XML, and other Web-oriented mark-up languages.

CSP is not a Web design tool, although it can be used with them. While Web design tools often concentrate solely on the production of static HTML, CSP goes beyond the appearance of pages to aid in the development of application logic. It also provides the run time environment that enables rapid execution of code within the Caché Application Server.

CSP supports a strong procedural programming environment, so applications can be written with a level of sophistication and exactness that exceeds what is possible with pure application generation technologies. It also supports rapid development through its class architecture, which produces "building blocks" of code that can be combined, and through the use of wizards, which can quickly produce simple versions of customizable code. The result is the ability to quickly develop very sophisticated Web applications.

THE CACHÉ ADVANTAGE

Object-oriented procedural programming, plus Caché Wizards, result in rapid development of sophisticated browser-based database applications.

Some of the characteristics of Caché Server Pages are:

- **Dynamic Server Pages** – Because pages are created dynamically on the application server by application code, rather than having a Web server simply return static HTML, applications can respond rapidly to a variety of different requests and tailor the resulting pages that get sent back to the browser.
- **Session Model** – All of the processing related to pages from a single browser are considered part of a session – from the first browser request until either the application is completed or a programmable timeout occurs.
- **Server State Preservation** – Within a session, application data on the server – and even the entire application context – can be automatically retained across browser requests, making it much easier to develop and run complex applications.
- **Object Architecture** – Because every page corresponds to a class, code and other characteristics common to many pages can be easily incorporated through inheritance. Data is also typically referenced through objects with all of the benefits of object-oriented programming.
- **XML** – XML provides a powerful alternative to HTML for building Web pages. CSP works the same way with XML as it does with HTML. Instead of the programmer supplying HTML in the Page() method, XML is provided.
- **Caché Application Tags for Automatic Generation of Server Application Code** – These extended HTML tags are as easy to use as traditional HTML tags. When added to an HTML document, they generate sophisticated application code providing a variety of functionality, such as opening objects, running queries, and controlling program flow. These tags are extensible – developers can create their own to suit their specific needs.
- **Integration with Popular Web Design Tools** – CSP works with a variety of tools that make it easy to visually lay out a page. With Dreamweaver, CSP goes a step further with the ability to add Caché Application Tags through simple point-and-click interaction. CSP also includes a Wizard that makes it easy to create forms that display or edit data in a Caché database.
- **Server Methods Callable from the Browser** – To facilitate development of more dynamic interactive applications, CSP makes it easy to invoke server-side methods. When an event occurs in the browser – typically because the user took some action – application code on the server can be invoked and a response to the event generated, all without the overhead of transmitting and displaying a whole new page.
- **Encryption** – Caché automatically encrypts data in the URL, to help authenticate requests and prevent tampering. The encryption key is kept only on the server, and it is only good for the life of the single session.

That's a lot of technology – but it does not have to be hard to use. We've focused on making CSP simple to use, with most of these capabilities working automatically for you. Our philosophy is power through simplicity – the complexity should be in our implementation, not in your programming.

THE CACHÉ SERVER PAGE MODEL

Traditional Web Technologies

With traditional Web technology, a request is sent to the Web server, and the Web server retrieves a sequential file of HTML that it sends back to the browser. When applications involve variable data, development gets more complicated, with programmers typically using CGI (with languages such as Perl or tcl) on the Web server and sending SQL queries and stored procedure requests to the database. As a programming environment, this leaves much to be desired, and execution – particularly with a large number of users – can be quite inefficient as the Web server gets heavily overloaded.

With CGI, each browser request typically creates a new process. To avoid this overhead, programmers sometimes link application code directly to the Web server, with the unfortunate side effect that an error in such code can crash the entire Web server.

Dynamic Server Pages

CSP uses a different programming and execution approach: Dynamic Server Page Technology. Content (HTML, XML, style sheets, images, and other types) is generated programmatically at run time on the Caché Application Server, rather than coming from sequential files, allowing much greater flexibility in responding to page requests.

Most of the application code executes on the Caché Application Server, which may or may not be on the same computer as the Web server. Some code – typically JavaScript or Java – may run on the browser, usually to support operations such as data validation, reformatting, or invoking server-side code.

With this approach, processes don't have to be created for each browser request (as they do with the traditional CGI approach), boosting performance. And since the application code isn't linked to the Web server, an application error cannot crash the Web server.

Sessions – The Processing Model

All of the processing related to pages from a single browser are considered part of a session – from the first browser request until either the application is completed or a programmable timeout occurs. When the Web server receives a page request (URL) with the file extension ".csp", that request is sent (by a thin layer of Caché code attached to the Web server) to the appropriate Caché Application Server, which may be on a different computer.

When the Caché Application Server receives the request, it determines whether a session is already in progress for that browser. If not, one is started automatically. Caché then executes application code associated with that particular page – performing the actions requested by the user and programmatically creating HTML, XML, image, or other content that is sent back to the browser.

A session ends when a property is set in the Session object terminating the session. Applications that run stateless may elect to terminate the session on each page.

State Preservation

One of the challenges facing developers is the inherently stateless nature of the Web – there's usually no simple way to retain information on the server from one request to the next. Applications typically send all of the state information they need to retain out to the browser in URLs or hidden form fields. That's not an effective technique for more complex applications, which may resort to saving data temporarily in files or databases. Unfortunately, this imposes significant overhead on the server, and it makes programming considerably more difficult.

Caché's Session model enables Caché to automatically and efficiently preserve data between calls from a browser. CSP provides a Session object that contains general session information plus properties that enable the programmer to control various session characteristics. The application can store its own data in the Session object, which is automatically retained from one request to the next.

The application determines how much state to preserve by setting the Preserve property of the Session object to 0 or 1. (The default is 0, and it can be dynamically modified at run time.)

- **0** – Data stored in the Session object is retained. (Data is simply set into a multidimensional property that accepts data of any type and allows any number of subscripts – including string valued subscripts – without any declarations.)
- **1** – Caché dedicates a process to the session so that all of the state of the process is retained, including all variables (not just those in the Session object), I/O devices, and locks.

A setting of 0 allows a logical partitioning of all preserved data and permits multiple sessions to share a single process, but it preserves less state. A setting of 1 is easier for the programmer and provides a wider range of capabilities, at a cost of increased server resources.

THE CACHÉ ADVANTAGE

By automatically preserving state information on the server, there's less network traffic and less overhead on the server, as the application doesn't have to keep filing and accessing data on every page request. And programming the application is simpler.

The use of Dynamic Server Pages and the Caché Application Server results in greater flexibility to respond to requests, faster execution without the risk of application errors bringing down the Web server, and a richer programming environment.

The Request Object

CSP automatically provides several objects, in addition to the Session object, to help the programmer process the page. One of these is the Request object. When a page is received, the URL is decoded and its contents are placed into the request object. The request object contains all name/value pairs and any form data, along with other useful information. For example, the value of the name "FilmID" could be obtained by the code:

```
%request.Data("FilmID",1)  // the 1 indicates we want the 1st value for this name
                           // in case multiple values are present for this name
```

CLASS ARCHITECTURE OF WEB PAGES

For each Web page, there is a corresponding page class that contains methods (code) to generate the page contents. When a request is received, its URL is used to identify the corresponding page class, and the Page() method of that class is called. Usually page classes are derived from a standard Web page class "%CSP.Page" that provides every page with various built-in capabilities, such as the generation of headers and encryption. Those standard capabilities can be overridden through a variety of means – deriving from another superclass, using multiple inheritance, or simply overriding specific methods.

This class architecture makes it easy to change behavior for an entire application and to enforce a common style. It also brings all of the other programming advantages of object programming to Web development.

The page class contains code to perform the requested action and generate and send a response to the browser, but not all of the application code that is executed is in that page class. In fact, most of the executed code is typically in methods of various database classes and perhaps additional business logic classes. Thus, the development process consists of developing both page classes and database classes (plus perhaps additional business logic classes).

In general, we recommend that page classes contain only user interface logic. Business logic and database logic should be put into different classes, so that there is a clean separation of user interface code from the business and database logic, and it is easier to add additional user interfaces later.

MULTIPLE DEVELOPMENT STRATEGIES

A page class is created for each Web page and contains the code to be executed for that page. There are several ways to build page classes, and most applications use more than one:

- **CSP File** – An HTML file with embedded Caché Application Tags is written using a simple text editor or Web design tool. That sequential file (a "CSP file") is not sent directly to the browser – it is compiled to generate a page class.
- **Direct Programming** – Programmers write the entire page class by coding the appropriate methods.

CSP FILES

CSP files are sequential HTML files with embedded Caché Application Tags that are compiled into page classes – the same sort of page classes that a programmer might write directly. Those page classes are then compiled to generate code that runs on the Caché Application Server in response to browser requests.

Caché Studio includes a Form Wizard that automatically generates a CSP file to edit or view a database class. The user simply clicks on the database class of interest and then clicks on the set of properties to be displayed. The Caché wizard does the rest – adding HTML and Caché Application Tags to the page. Since the wizard outputs HTML, if the result is not exactly what you want, it is easy to edit it.

The CSP file approach is powerful because:

- Web artists can design the visual layout while programmers concentrate on the code.
- Much of the user interface can be programmed non-procedurally in a visual environment and kept isolated from the business and database logic.
- It is often easier to customize an application for a particular user by allowing non-programmers to modify the visual presentation and add simple capabilities.

Since the visual specifications of the application are kept separate from most of the programming logic, it is relatively easy to change the appearance without reprogramming. Simply edit the HTML or XML file and recompile the page.

Although a complete simple application can be created this way, more commonly a programmer supplies additional code. This additional code is provided though application tags that either include the procedural code or invoke code in other classes. However, complex pages with a lot of procedural code are often easier to write using the direct programming approach rather than a CSP file.

Caché also includes an add-in for Dreamweaver, the popular Web page design tool. It provides point-and-click support for adding Caché Application Tags, as well as a Caché Form Wizard that automatically generates the code needed to view or edit a database object.

Caché Application Tags

Caché Application Tags can be added to CSP files. They are used just like standard HTML tags, but they are really instructions to the Caché Web Compiler to generate application code that provides a variety of functionality, such as accessing database objects, running queries, program flow control, and executing code on the Caché Application Server. Caché Application Tags are extensible – developers can create their own to suit their specific needs.

Caché Application Tags are not embedded in HTML that is sent to a browser – they are only in the CSP file read by the Caché Web Compiler. That compiler automatically transforms them into standard HTML, which can be processed by any browser.

HYPER-EVENTS

CSP hyper-events allow events occurring on a browser (such as mouse clicks, field value changes, or timeouts) to invoke server-side methods and to update the current page without repainting it. After taking the appropriate action, that server method can return code – typically JavaScript – for execution by the browser. Using hyper-events, Web applications can become more interactive and responsive.

Within a CSP page, a server side method is invoked simply with the syntax:

```
"#server(...)#"
```

For example, suppose that when the user clicks on a shopping cart image, we want to call a server method called `AddToCart()`. Then the HTML definition for the image might include:

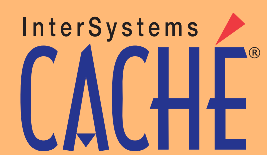
```
onClick="#server(..AddToCart())#"
```

The Web compiler will replace this syntax with JavaScript code that when run on the browser, will invoke the Caché server method.

InterSystems Corporation

World Headquarters
One Memorial Drive
Cambridge, MA 02142-1356
Tel: +1.617.621.0600
Fax: +1.617.494.1631

www.InterSystems.com



InterSystems Caché is a registered trademark of InterSystems Corporation. Other product names are trademarks of their respective vendors. Copyright © 2006 InterSystems Corporation. All rights reserved. 07-06