Case Study: Unicode in Large Network Systems Management

David Hetherington
Manager TME 10 Internationalization
Tivoli Systems

Abstract:

Tivoli Systems makes the Tivoli TME 10 product set which allows enterprise customers to manage extremely large heterogeneous networks. The TME 10 product set consists of a CORBA framework and a set of associated object oriented applications which run on a wide variety of operating systems platforms including Windows NT, Solaris, HP-UX, AIX, OS/2, NetWare and many others. Tivoli's slogan is "The Power to Manage Anything. Anywhere." In a large international or multinational corporate network, the "Anything. Anywhere" part can involve a daunting array of equipment running incompatible codepages. This presentation will contain an overview of the Tivoli TME product set, the underlying CORBA framework architecture, the codepage related problems encountered, and the Unicode implementation used by Tivoli to overcome the problems.

What is Enterprise Level Heterogeneous System Management?

System Management can loosely be described as any tasks which are handled by a company’s IS staff. However, to better understand the products concerned in this case study, we can start with a few examples.

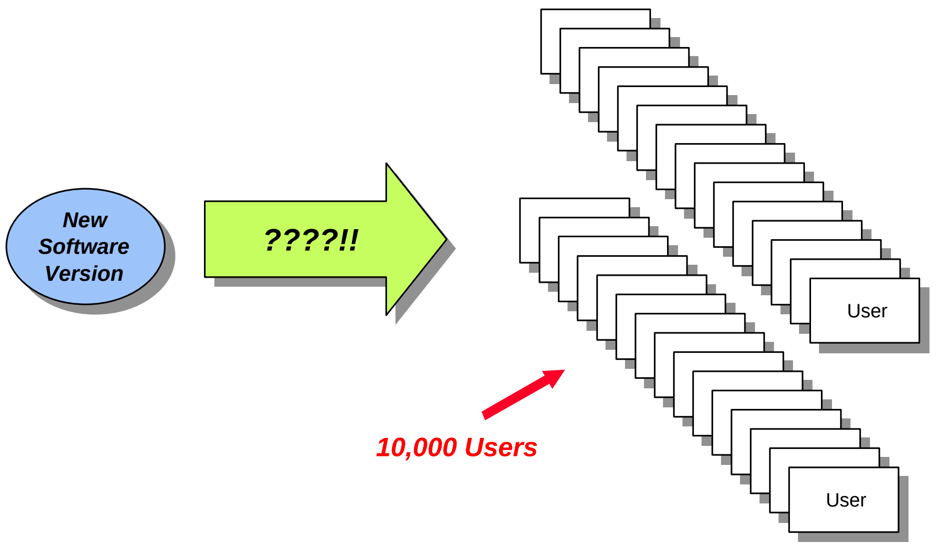


Figure 1 - Example: Software Distribution

Consider the problem facing a large company with 10,000 PC users when a major upgrade appears for everyone’s favorite word processing package. Consider the options facing the IS staff:

1. **Do Nothing**. In this case the users will solve the problem themselves. One by one they will submit requests to the purchasing department to purchase the upgrade. The purchasing department will process 10,000 separate purchase orders. The accounts receivable department will receive 10,000 separate invoices and issue 10,000 separate checks.

2. **Order Centrally.** The IS department orders a site license and walks around installing the software. Several man-years of effort go into software upgrade.

3. **Put it on the LAN Server**. The IS department orders a site license, puts the installation disk images on the LAN server and lets the user community try to install the upgrade themselves. Unfortunately, this is not an engineering company and the user community is not very technically savvy; every fifth user messes up the installation and calls the help desk. The help desk takes 2,000 calls for assistance in unscrambling the incorrect installation.

Obviously, none of these options are very attractive. The answer is to use system management software to automate the distribution and installation of the software. A good software distribution product will not only move the data to all 10,000 endpoints, but also perform any needed preparatory steps, synchronize the activation of the new version if needed and do any number of other things to insure a successful company-wide deployment.

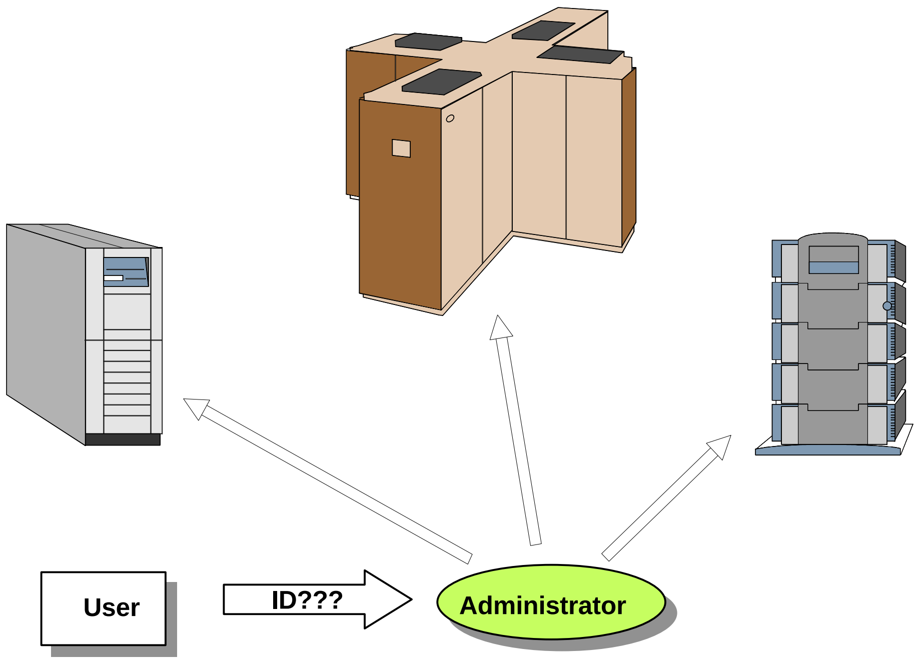


Figure 2 - Example: User Administration

In a modern company, it is not unusual for every single employee to need user accounts on multiple systems. For example, the users might routinely need accounts and passwords for Windows NT, a mainframe and a database server.

Administering all of these separate accounts is a time consuming headache for the IS staff. It is also difficult to keep administrative policies consistent when the accounts are spread across multiple incompatible systems. A good system management software package will allow the administrator to create and administer the multiple accounts for each user as though they were a single account.

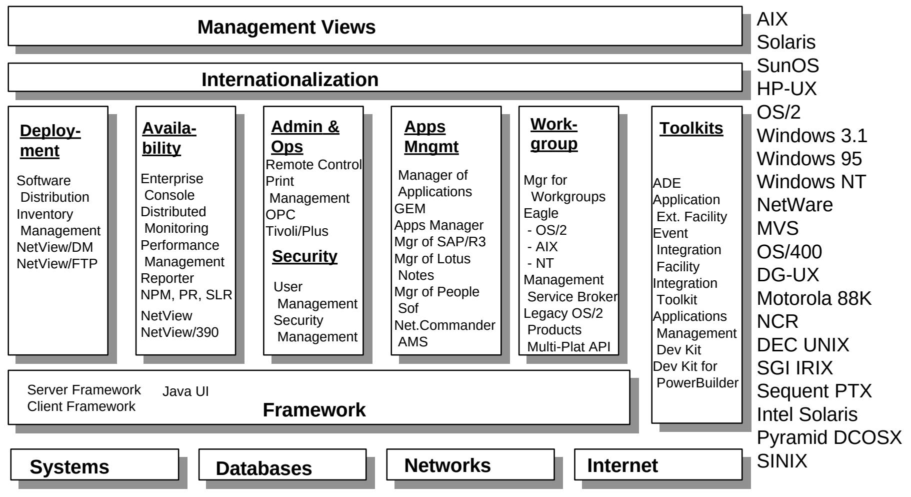


Figure 3 - Overview of Tivoli's TME 10 Product Line

Tivoli’s TME 10 product line includes these two system management applications and many others as well. These applications run on a wide variety of operating system platforms. Tivoli has customers all over the world.

Tivoli’s CORBA Framework Implementation of System Management

Most of the applications shown in Figure 3 above are based on a Tivoli implementation of a CORBA-compliant object framework.

Most programmers are aware of object oriented programming in objected oriented languages like C++. Within a C++ program, a programmer can define objects which inherit properties from other objects. Actually, however, once the program is compiled and linked, there is generally nothing objected oriented about it. That is, a program coded, compiled and linked in C++ does not allow another program to “inherit” anything any more than a program coded in assembly language. Also, while C++ does a great job of defining objects in memory, keeping those objects around between executions of the program tends to be a major headache.

CORBA approaches the problem of object orientation from an entirely different angle. The object request broker or “ORB” runs as a service on each machine. Users can interact with the ORB (literally by typing commands on the command line) to define classes, assign scripts and programs to act as the methods for those classes, and allocate persistent storage on disk for the objects. New classes can be defined at runtime without stopping the execution of the existing application.

CORBA achieves object orientation independent of the features of the language which the object methods are written in. Object methods can be C programs, PERL scripts and so on.

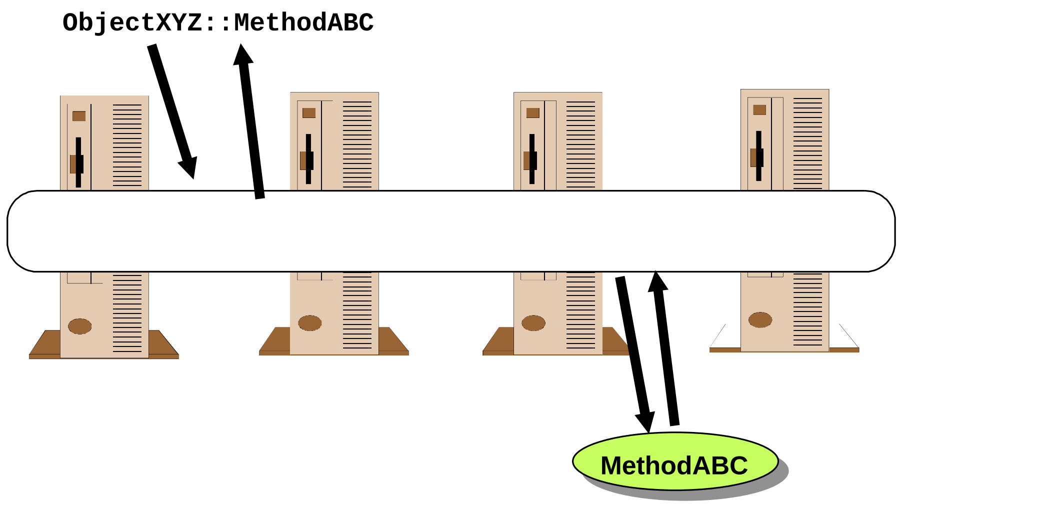


Figure 4 - The CORBA Framework

The CORBA framework in a network application is actually the sum of the ORBs on each machine. One of the nice features of CORBA is location and platform independence. When a object invokes a method of another object, the calling method need not know where the called object actually resides. The ORBs working together resolve the actual location of the called method, takes care of moving any parameter data to the method and takes care of moving results back to the caller.

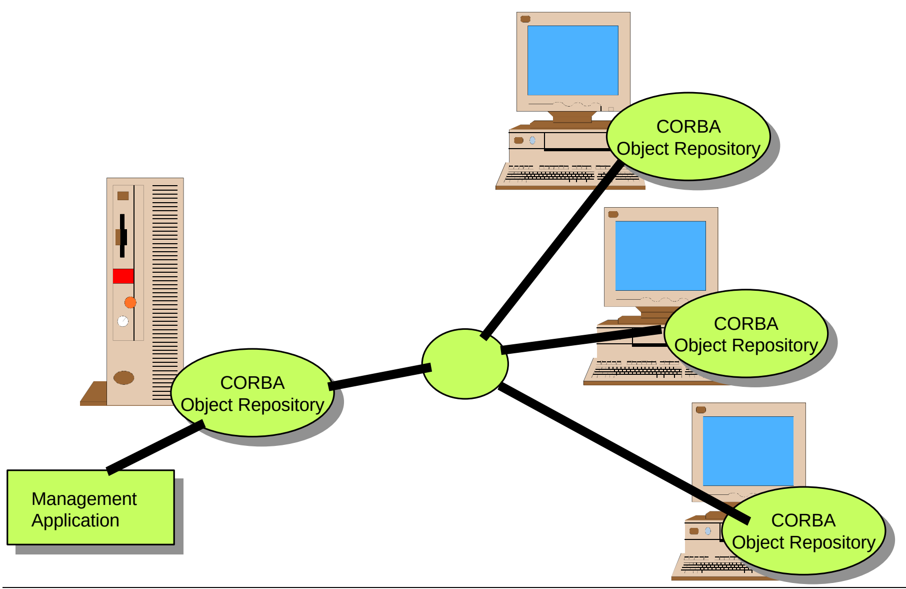


Figure 5 - Tivoli System Management Application Using CORBA Framework

The network distributed nature of the CORBA framework make it ideal for implementing enterprise system management applications. Most of Tivoli’s system management applications are implemented as collections of CORBA objects using Tivoli’s framework.

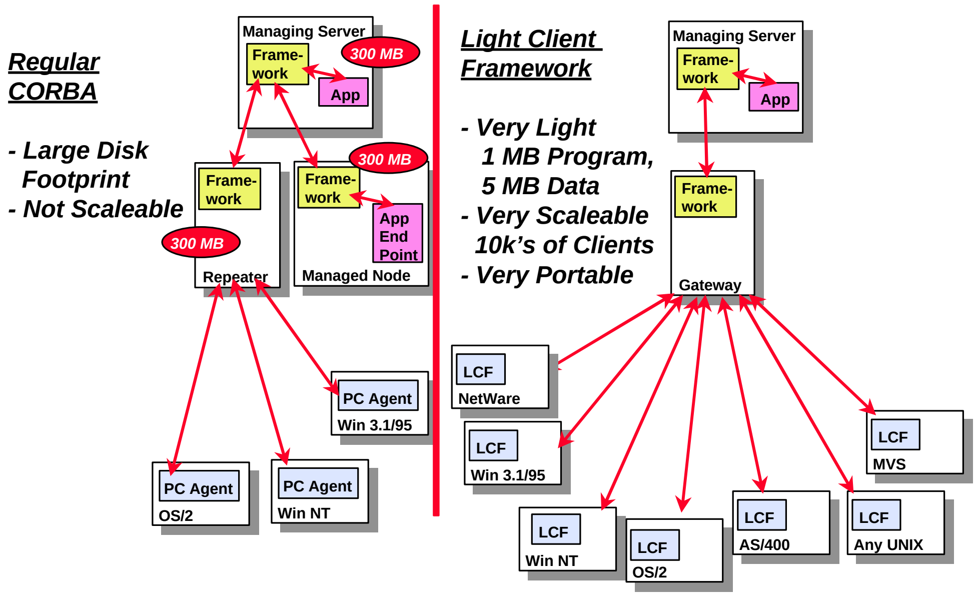


Figure 6 - The Light Client Framework

There are, however, some points at which the reality diverges from the theory. Use of the CORBA framework model has allowed Tivoli to implement industry leading systems management applications rapidly. However, there have been some limitations to the approach. The key limitations are:

1. The ensuing object set is too large to reasonably install on personal computers.
2. The pure CORBA approach encounters performance problems when trying to manage networks of thousands of nodes.
3. The pure CORBA approach has some difficulty accommodating computers whose attachment to the network is intermittent.

To overcome these limitations, Tivoli has developed a modified CORBA approach called the Light Client Framework. In this approach, the application is still written logically as an objected-oriented CORBA application. However, the endpoints are not actually full CORBA systems. The endpoints run a very small, tight kernel which communicates with a gateway. The gateway downloads object methods to the endpoints on demand. The endpoints cache frequently used methods on local disk. The tradeoff of download performance versus cache disk consumption can be finely tuned to match customer requirements. The resulting system is much leaner at the endpoint and much more scaleable than a pure CORBA implementation.

What sort of Problems are Caused by Character Sets and Codepages?

In taking the TME 10 product set international, Tivoli was facing a number of specific technical problems related to character sets and code pages.

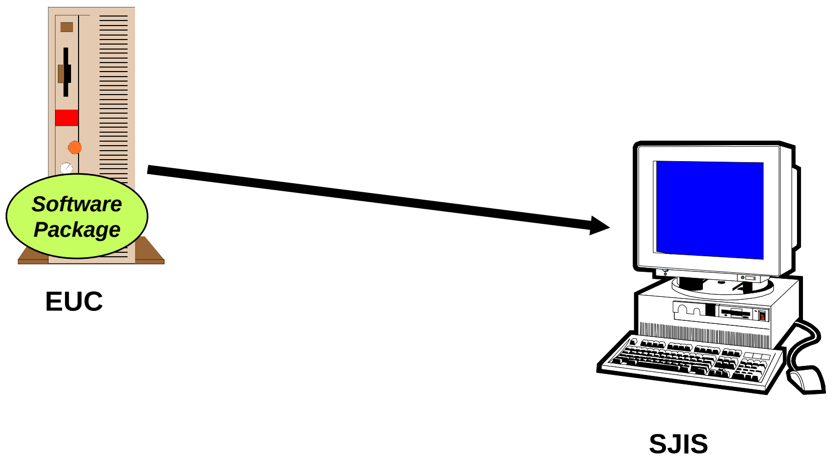


Figure 7 - Problem: Pathnames in Software Distribution

One of the first problems to be encountered was in the area of software distribution. The administrator would prepare a package of files to be distributed to a large number of personal computers. The administrator might be working on a UNIX platform. The users would typically be using personal computers. The administrator would enter specific file and path names into the software package. When the package arrived at the PC endpoints, the pathnames would be incorrect due to codepage incompatibilities between the UNIX workstation and the PCs . The distribution would fail as a result.

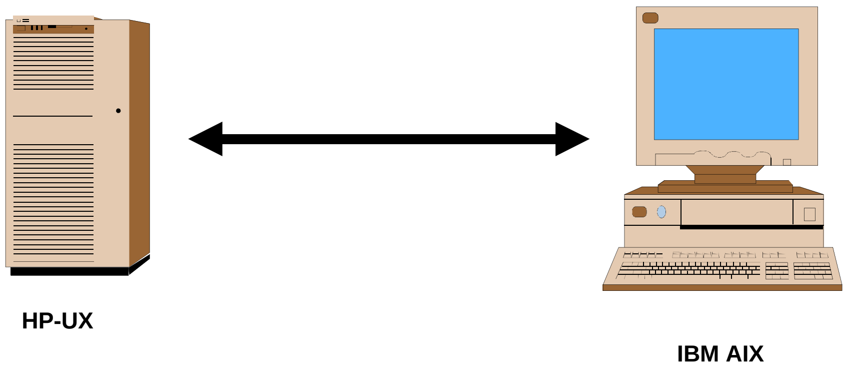


Figure 8 - Problem: French Usernames

Tivoli's TME 10 User Administration product greatly simplifies the life of the administrator by allowing a user's entire profile of data to be entered once and distributed to a variety of heterogeneous systems. The

French codepages used by Hewlett Packard and IBM for their UNIX workstations are not compatible. User information would become garbled when such a distribution was attempted.

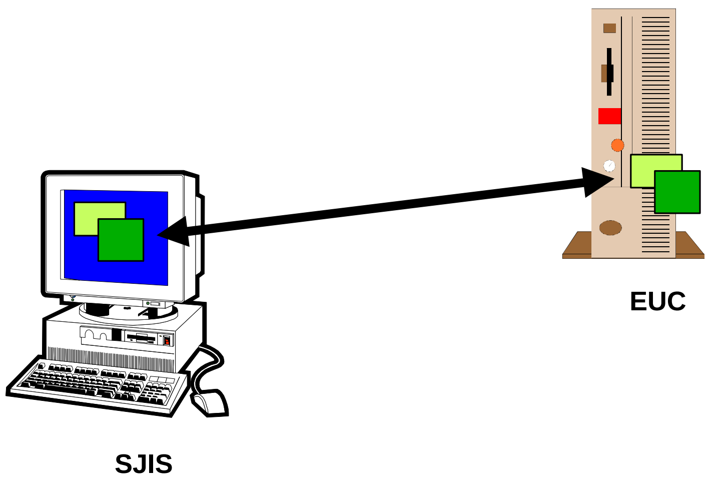


Figure 9 - Problem: Remote Graphical User Interface

One of the nice things about having a product based on a CORBA object implementation is that the user interface is simply another collection of objects. As such, the user interface can run anywhere in the network. It does not need to run specifically on the system management application management server. This convenience naturally leads to the situation in which the system management application is running on a heavy duty UNIX server for performance and reliability while the administrator is running the user interface on a personal computer in his office. Data being delivered from the UNIX server to the personal computer for display are garbled because of codepage incompatibility.

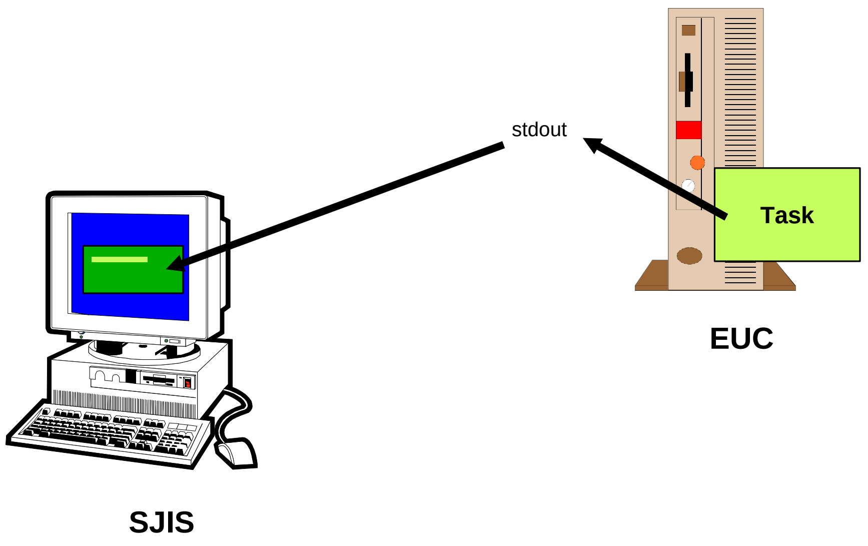


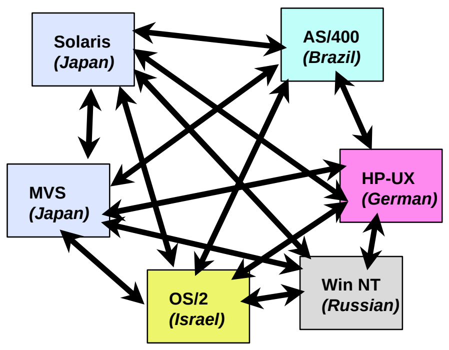
Figure 10 - Problem Console Output from Remote Tasks

As described above, CORBA can freely associate PERL and bash scripts with classes to serve as the methods for the object instances of those classes. These scripts generally return output simply by piping text to stdout. This text output is returned by the CORBA framework to the management application and in some cases to the administrator's display. Codepage incompatibilities cause this information to be garbled in transmission.

Tivoli's UTF-8 Unicode Implementation

To attack these problems, it was obvious that some sort of codepage conversion would be needed.

- Any-to-Any Codepage Conversion is a mess with 30 different platform types.
Solution => Use UNICODE



Any-to-Any Codepage Conversion - What a Mess!!

Figure 11 - The Brute Force Approach

The first approach considered was to have systems understand where data was coming from and going to and do a direct conversion of between the codepages as needed. IBM's work on CDRA, a codepage registry architecture was reviewed. It was found that there were over 100 codepage variations just for Japanese on mainframe computers alone. IBM provided the codepage conversion tables needed for to cover the IBM platforms on two CD-ROMS. These two CD-ROMS did not cover the non-IBM platforms. This was not a promising approach.

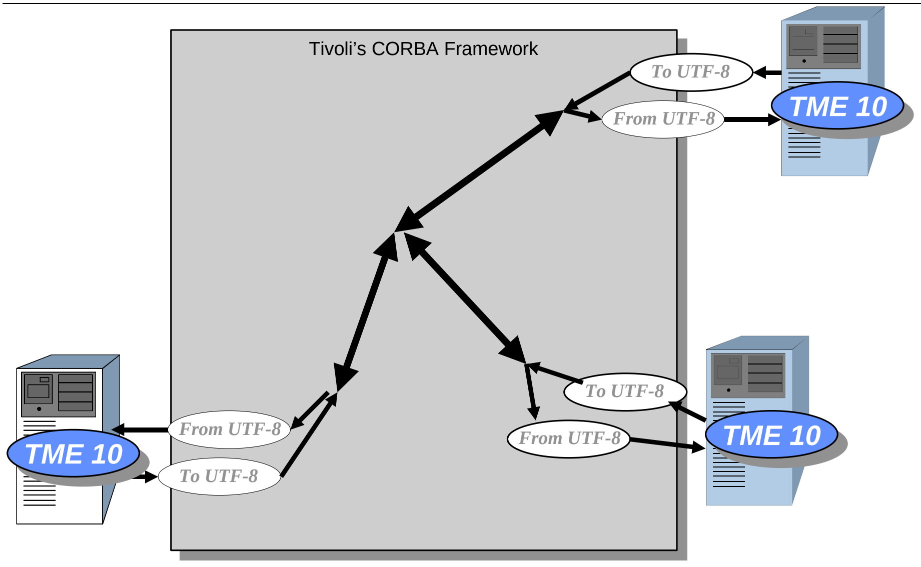


Figure 12 - The Canonical Format Approach

To reduce the complexity entailed in supporting any-to-any codepage conversion in a product covering so many different operating system platforms, Tivoli decided to use Unicode as an intermediate or "canonical" format. All data would be converted to Unicode before transmission to the next system. Upon arrival it would be converted back to the local codepage for processing.

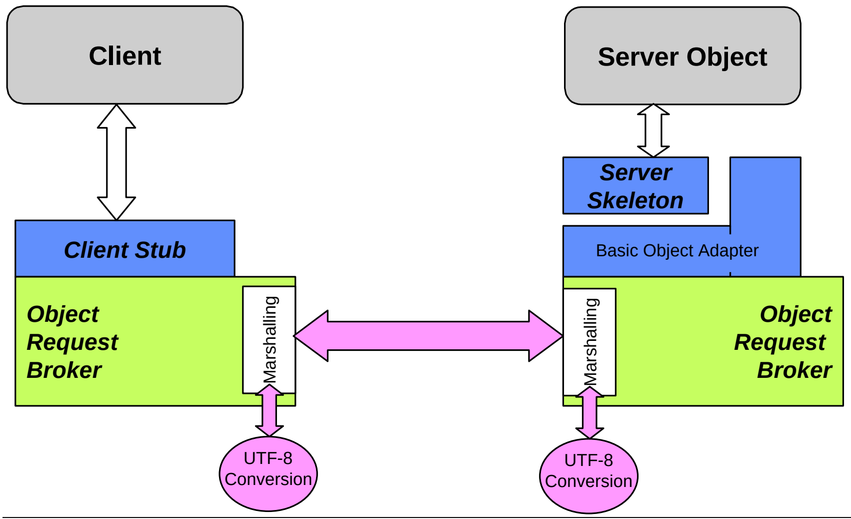
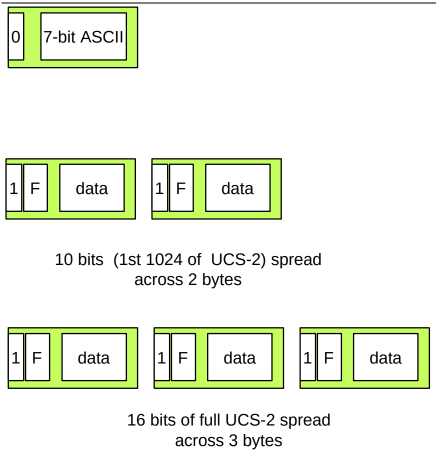


Figure 13 - Conversion in the Marshalling Layer

The next problem was to choose the best point in the software to perform the conversion. This investigation did not take long at all. CORBA already has to contend with problems like integers being represented differently on different machines, the famous "big-endian" versus "little-endian" problem. To overcome these problems, the CORBA framework contains a "marshalling" layer. In this layer, all data is transformed into a vendor neutral format before being transmitted to the next system. Tivoli installed the Unicode conversion mechanism in the marshalling layer.


0
7-bit ASCII
1 F data
1 F data
10 bits (1st 1024 of UCS-2) spread across 2 bytes
1 F data
1 F data
1 F data
16 bits of full UCS-2 spread across 3 bytes

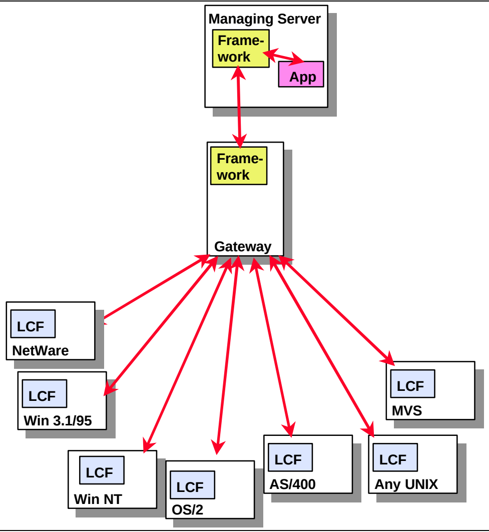
- **Flows very well through legacy code**
- **Allows complex system to be upgraded in stages (English still works)**
- **Only applies size penalty where needed.**

Figure 14 - The UTF-8 Format

The next question was how to represent the data. The normal UCS-2 format was considered but was bypassed in favor of the UTF-8 encoding format. There were two factors which weighed heavily in the choice of UTF-8 over UCS-2:

1. The entire application set consists of many million lines of source code of all types. Even when confining the conversion to the marshaled data, finding all of the locations in the code which would have to be reworked to accommodate 16-bit characters was daunting. Tivoli was on a tight schedule. UTF-8 yields tremendous advantages in that all the characters that code loops normally scan for such as: / \ [ ] { } , ; : ‘ “ ` ~ ! @ # $ % ^ & * ( ) = - | + as well as end-of-line and NULL (for string termination) are the same single byte values as traditional 7-bit U.S. ASCII. Since all other values have the high order bit set, most existing code loops will naturally skip over the multibyte character data, but still find delimiters and string terminations correctly.

2. In large complex installations of many thousands of systems of a dozen or more different types, it is not acceptable to force the customer to “flash upgrade” the entire system all at once. Previously, Tivoli had only ever offered formal support for U.S. English. Since the UTF-8 representation of U.S. English and traditional 7-bit ASCII are the same thing, existing customers would be able to do a phased upgrade of their systems. Converting text to UCS-2 would have made new systems incompatible with older systems, even for U.S. English.



- **Full set of conversion tables too bulky for light endpoints.**
- **Full set is stored on gateway.**
- **Gateway downloads correct conversion table at the time the light endpoint connects.**

Figure 15 - Downloading Conversion Tables from Gateway to Endpoints

There was one other consideration which was the size of the conversion tables. The sum of the size of all the necessary conversion tables was several megabytes. This size was much better than two CD-ROMS for the any-to-any solution. Nevertheless, several megabytes was still unacceptable in terms of Tivoli’s design goals for lightweight support on PC endpoints. Fortunately, Tivoli’s lightweight design explicitly supports downloading files from the gateway to the endpoint on demand. The design was enhanced to make the gateway query the endpoint and download the correct conversion files at the time the endpoint first connects to the gateway.

Conclusions

- Must Use Unicode as Canonical Format (Point to Point Conversion is Hopeless)
- UTF-8 Much More Useful than UCS-2
- Unicode Maximum Benefit is on the Wire
- UTF-8 on-the-wire Implementation Achieved Very Quickly with Reasonable Effort
- Have to Pay Attention to Table Sizes

Figure 16 - The Key Conclusions

The key conclusions from the project were as follows:

1. **Canonical Format**. The any-to-any approach was hopeless. Using Unicode as a canonical format greatly simplified the design.

2. **UTF-8**. Using UTF-8 rather than UCS-2 greatly simplified the design task while also allowing Tivoli to maintain backward compatibility with previous English-only systems.

3. **Unicode on the Wire**. Tivoli will eventually convert its products to run entirely in Unicode. A large part of this conversion will happen naturally as a byproduct of moving Tivoli's applications to Java in the next few years. For the present, however, substantially all of the benefit of Unicode is obtained by only implementing Unicode on the wire as described above. The gains to be achieved in modifying the rest of the several million lines of code to process Unicode natively are modest in comparison.

4. **Effort and Speed**. By focusing only on implementing UTF-8 on the wire, the changes were completed by a small team of a few engineers in a few months. It should be noted, that Tivoli had an advantage in picking up the conversion tables ready-to-go from the IBM National Language Technical Center in Toronto. Tivoli also was able to obtain source code examples for optimal implementations of the conversion routines from IBM's AIX and OS/2 teams. Nevertheless, Tivoli is aware of other industry teams which have spent years attempting to fully migrate their products to UCS-2, many of which still are not done yet.

5. **Table Size**. In implementations which are striving to be compact and lightweight, attention must be paid to the size of conversion tables.