2

Distributed Computing Concepts

The Netzyme® Enterprise (NZE) Server is an integration middleware suite that provides a rapid distributed application development environment and an operating infrastructure for distributed applications. By handling both, we have taken control of how distributed applications communicate and can therefore integrate them into any platform for your use. This is our understanding of what "middleware software" should be. These two facets of NetZyme®, which provide for functionality and development respectively, can be visualized as two layers.

- The NetZyme® Enterprise Core (NZ-CORE) handles all communication between distributed NetZyme® modules and all low-level network details of constructing/deconstructing the data exchanged between modules. NZ-CORE encapsulates fault tolerance, load balancing, scalability, session support, and run-time management.
- The NetZyme® Enterprise API (NZ-API) provides a broad set of tools that are needed to develop thorough distributed enterprise systems.

NetZyme® Enterprise adapts to, shares data, and performs distributed processing automatically in both B2B (one-to-one communication paradigm) and E2E (many-to-many, or neural, communication paradigm) environments.

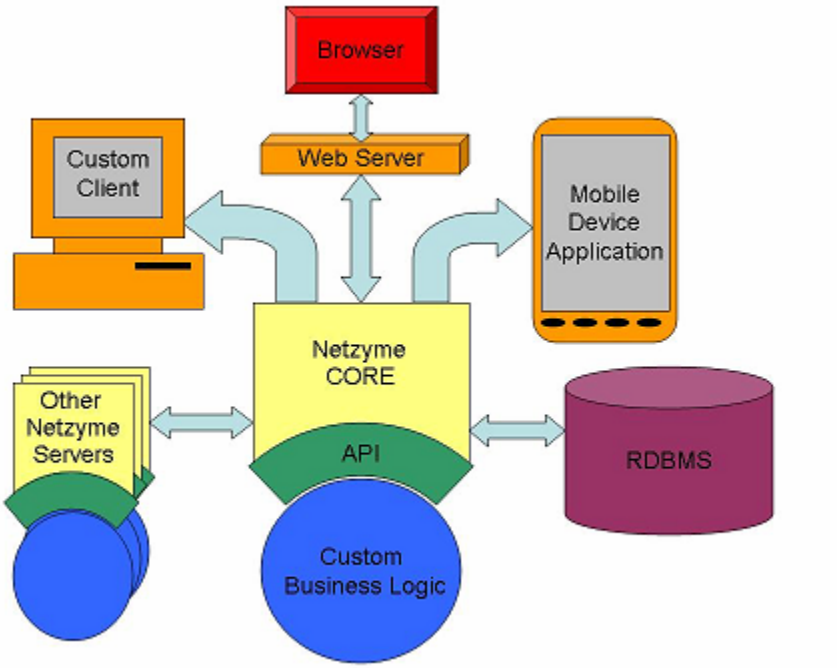

Figure 2-1 *.An example of a NetZyme® based infrastructure. Notice that the custom code (Business Logic) is tightly coupled with the Netzyme Core. As this diagram also shows, Netzyme is not limited to Web Applications.*

2.1. Core Concepts

To maximize the benefits of NetZyme®, several fundamental concepts must be understood. This section explains the following concepts and how each is factored into the NetZyme® framework:

- The need for distributed processing
- Multi-tier distribution
- Near linear scalability
- Extendibility
- Fault tolerant / mission critical computing
- Platform and language data encapsulation

These concepts contribute to the foundation that makes NetZyme® the leading innovation in the enterprise-grade software infrastructure market. By considering these concepts from the beginning, Creative Science Systems has created a framework strong enough in all of these areas to enable the most rigorous industrial applications.

2.1.1. The Need for Distributed Processing

Distributed applications are applications with the capability to manage work across more than one system and to centralize information so that it is accessible remotely. The most common examples of distributed applications comprise the Internet, which exemplifies every known use and benefit of a distributed application. On the Internet, a centralized locale of information can service a large and widely distributed community of people.

When an Internet location is properly running, a connected user can access needed information from anywhere in the world.

Consider this situation:
An online service is beginning to be popular and their network traffic, which once was no more that 100 users at a time, has increased such that every time the site is visited by more than 10,000 users, the server crashes. By distributing the functionality across multiple servers the online service is able to support as many users as it needs, simply by adding another server. Even the latest hardware and software technology is hindered by some threshold. Thus, the need for the distribution of work becomes evident as the number of users grows for a single server. This online service is a common example of an operation that needs a scalable solution.

Consider another situation:
It takes a single (dedicated) computer 5 hours to perform some computational task, such as rendering a large volume of graphics. In this situation, time, not money, is the bottleneck. If there were an application that could distribute the work between five networked computers and then reassemble the results, perhaps one that would complete the computation in 1.5 hours; then, distributed processing would reduce the computation time by 70%.

Distributed processing or execution partitioning is built into the NZ-CORE. The NZ-CORE transparently handles distributed computing needs and frees developers focus on application-specific issues. NetZyme® provides this wheel so you don't have to re-invent it.

2.1.2. Multi-Tier Distribution

NetZyme® is designed to facilitate multi-tiered processing. A typical tier structure might consist of four tiers:

[client app] ---- [server app] ---- [business logic app] ---- [database]

A tier is an isolated piece of hardware or software in a distributed application that forms a logical link in a processing structure. Servers, databases, and client-side interfaces are examples of three separate tiers in a distributed application. With support for a multi-tier environment built into NetZyme®, an application can have as many tiers as necessary.

Because NetZyme® was designed to provide a rapid development environment for distributed applications, it has a built-in facility to connect a variety of existing tier components and to insure backwards compatibility for future extensions. This facility is an application development framework, the netZyme CORE (NZ-CORE), that can incorporate any tier that exists today and includes a facility for incorporating new and evolving tier technologies – including cellular phones, desktop machines, PDA's, DAT ECN's (direct access trading electronic communication networks), and the growing list of new innovations becoming available every day.

2.1.3. Scalability

Scalability refers to the level of difficulty involved in adding redundant processes to a distributed application in order to improve throughput, performance, and reliability. A

system is considered scalable if additional component clones can be added seamlessly and with minimal difficulty. When the number of client requests increase for an application running on a single computer, performance will inevitably suffer. Scalability allows you to balance the performance and throughput of an application to return it to original or better processing levels. Scalability is absolutely necessary in order for a distributed application to be fault tolerant. However, do not assume that once your system is scalable, it is fault tolerant. Scalability is only a single, albeit necessary, element of a fault-tolerant enterprise system.

Building a distributed application to be scalable ensures that the system using the distributed application can easily support fluctuations in workload (the number of processes and the demands on resources). Scaling a system does not change its functionality. Determining the number of computers to handle a distributed application, however, reflects the desired throughput, performance, and reliability of that distributed application. By making the system scalable, system developers don't need to worry about the amplitude of the fluctuations in the workload for the system.

These fluctuations in demand are a very natural occurrence in the world of distributed applications. Although it is not guaranteed to happen in all cases, it is very likely – especially with enterprise applications. Therefore, it is a good practice to plan for scalability during the design of an enterprise system.

In order to make your system scalable, these are the considerations:

- *Avoid assuming that all modules exist on the same machine.* Although this is implicit by virtue of an application being distributed, it is a good idea to address each module explicitly as residing on separate servers.
- *Design the system such that each module can be cloned with minimal effort.* When your server reaches capacity, you should be able to bring in a new machine, copy over all the relevant pieces (files, database elements, multi-media, etc), start the server, and your system will support one more factor of capacity for that module. This is the fundamental idea of scalability — application modules can physically be put on different machines and logically perform the same operations.

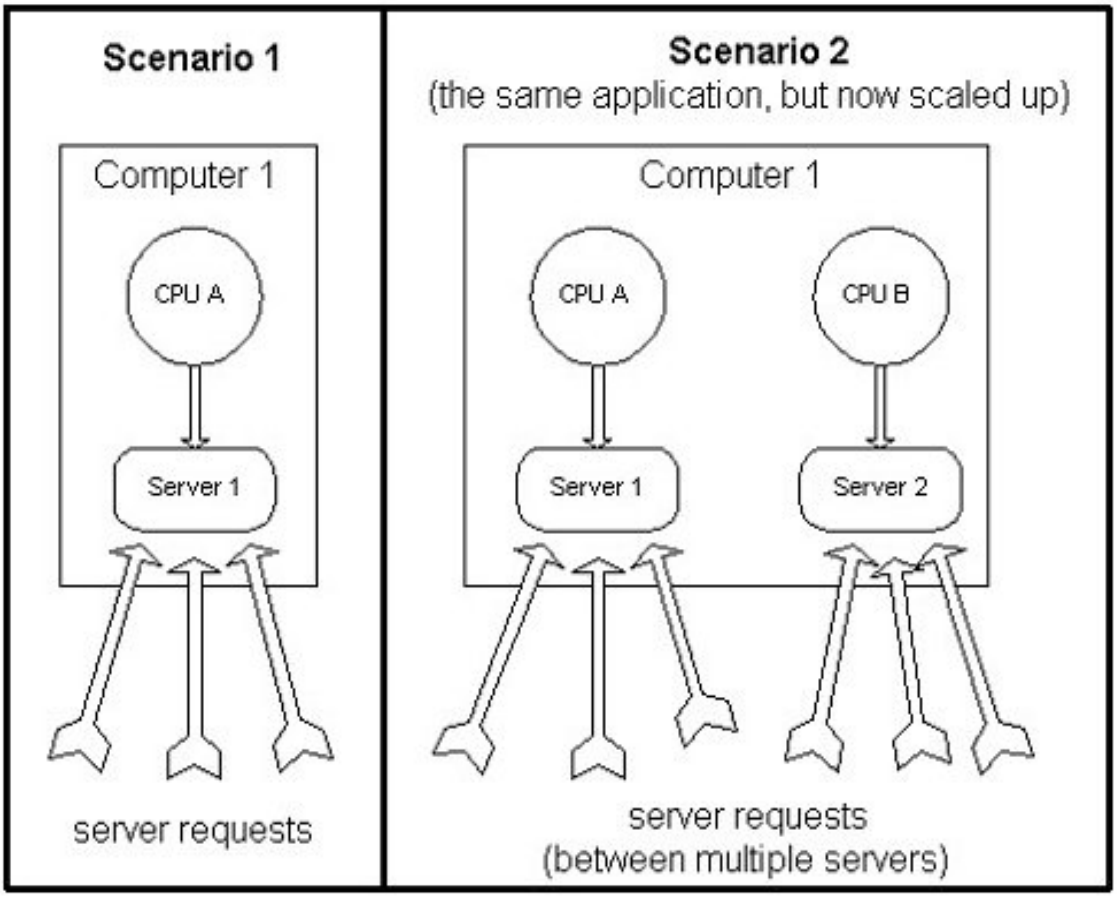

Figure 2-2: *A general example of how to scale a system*

Figure 2-2 shows a diagram of a scalable application. In scenario 1, the CPU is dedicated to the server. When server 1 is saturated with users, a second system (seen in scenario 2) can be activated to accommodate the increase in the number of server requests.

Note: you will need to insure that messaging is provided by the platform of your choice thus enabling your application to service the requests of alternate servers.

NetZyme®-CORE has embedded support for near linear scalability. That is, in order to double your performance and capacity, you would only need to double your resources. This is achieved by combining fault tolerant session management and software based load balancing.

2.1.4. Extendibility

Extendibility is the level of difficulty involved when adding functionality into an existing system. A system is considered extendable when the difficulty level is minimized and new functionality can be added seamlessly.

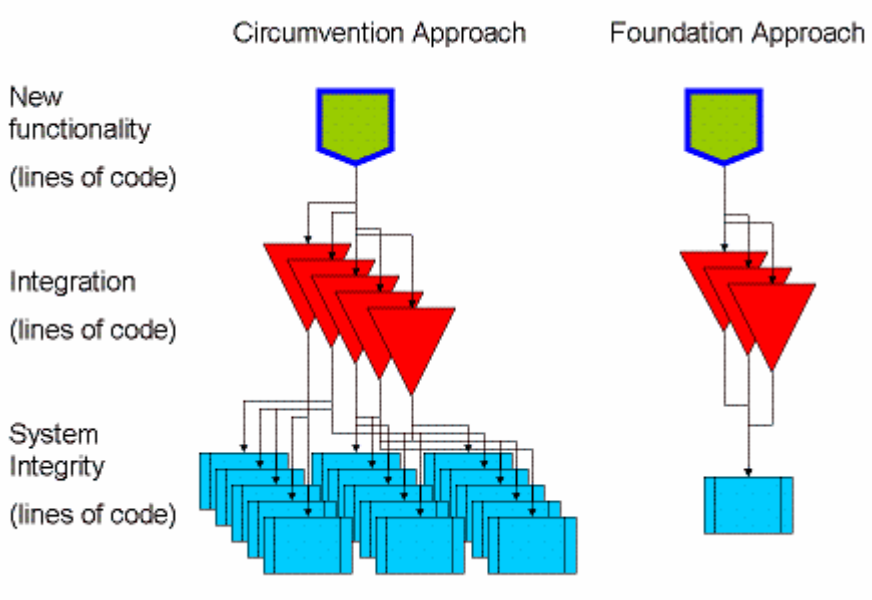

***Figure 2-3**: A comparison of the Circumvention and Foundation approaches to expandability*

There are two basic approaches to extend an existing system, "circumvention" and "foundation". "Circumvention" involves modifying functionality without a generic framework. It is labor-intensive, generating a 1-5-15 ratio of new functionality lines of code, integration lines of code, and system integrity lines of code, respectively. It is inefficient in comparison to the "foundation" approach, which integrates a generic framework that is adaptable to new functionality. This approach is technology-intensive, generating 1-3-1 code ratios. Whereas most of today's solutions use the "circumvention" approach, NZ-CORE core principal is the "foundation" approach.

The NZ-CORE design explicitly provides a framework for the adaptability of technology to ever-changing business requirements. NZ-CORE utilizes the concept of a nucleus server, which makes it a natural fit for requirements of both B2B and E2E enterprise system. This is because NetZyme® is dynamically configured to execute arbitrary business logic on one hand and inherently incorporates a modular architecture, which allows for unobtrusive changes in low level functionality on the other.

NetZyme® was designed with the understanding that technology is constantly evolving at a pace that exceeds that of current business models. Therefore, a system built on top of the NZ-CORE can be augmented with new functional components with minimal effort. From integrating new functionality to connecting a new tier, NetZyme® makes extending your enterprise system an ordered, logical and smooth process.

Of course, it would be much more pleasant for developers and code-maintenance professionals to anticipate the requirements of an application prior to the design phase. In fact, false beliefs in the ability to do just that are a very common mistake. Real-world development cycles are rarely as smooth and simple as textbooks would lead us to believe. NetZyme® is an innovation of both technology and business models, exemplifying the pure "foundation" approach to enterprise system extension and thus compensates for most of the technological, logistical and operational hiccups that arise during the evolution of an enterprise system.

2.1.5. Fault Tolerant / Mission-Critical

A system is considered mission-critical if it is unacceptable for the system to go down for longer than its scheduled downtime. Developing mission-critical applications requires you to define the downtime interval for application.

A mission critical system has the following core features:

- Load balanced
- Distributed
- Fault tolerant
- Scalable

We have already discussed scalability and the principles of a distributed system. The remainder of this discussion is dominated by the issues of fault tolerance and Load Balancing but keep in mind that we are talking about distributed and scalable systems throughout.

Fault tolerance is the capability of a mission-critical application to handle component failure and manage the redundant systems that activate when a component fails. The common issues that cause a system to malfunction include lost network connections, hardware/software failures, terminated processes, and, of course, human errors. When an application can recover rapidly from these adversities, the application is said to be fault tolerant.

The recovering application must have a time constraint on how long it has to identify that a component has gone down and to bring that or its redundant component back up. It is impossible to make this interval of time equal to zero and you should plan your requirements based on your needs, not according to a single ideal. For example, consider an air traffic control system. In order for this system to be considered mission-critical, it is unacceptable to have a disruption in the system for more than even a few seconds because lives are at stake. If you are creating a distributed application to track shipping abstracts and deliveries, a two minute interval is more acceptable.

With the ability to recover from service termination/interruption within a specified time constraint, the application is considered to be always running and therefore can be mission-critical. If the application exceeds the recovery time limit, the application is no longer functioning properly, and ceases to meet the mission-critical requirement.

One well-known method for providing a distributed application with fault tolerance is to establish redundant processes across a network. If a process suffers an interruption in service, the request is routed to another duplicate process running on the network. Another method of using redundant processes would be in a situation when a server is running its maximum allowed number of processes and subsequent requests are redirected to an available redundant processes located somewhere else on the same network.

In either case, utilization of redundant processes can prove to be immensely advantageous. Sometimes the redundant processes occur only when a distinct problem occurs. Such processes are launched only in order to provide auxiliary support and otherwise remain silent. When these silent redundant processes are running, they target allocation resources for the worst-case scenario.

Every industry has mission-critical systems. The need for a reliable fault tolerant infrastructure to support such systems is crucial and is often a key factor in determining the success of a given business venture. NZ-CORE was built on this premise and fortified to withstand the most adverse conditions presented by distributed applications. NetZyme® makes it significantly easier to construct a fault tolerant system. You can add redundant processes to a NetZyme®-based distributed application with little effort. In most cases, it is as simple as editing the configuration file for each additional process. Moreover, by virtue of deploying NZ-CORE, non-fault tolerant applications can be made fully fault tolerant.

Aimed at enterprise grade computing, fault tolerance support is enabled in the NZ-CORE by default thus providing for foolproof development. However, this facility can be disabled at the developer's discretion. This is the core concept of an NZ-CORE "out of the box" foundation that has infinite flexibility in the hands of experienced developers.

2.1.6. Platform and language agnostic data encapsulation

One of the most common obstacles for distributed application development is sharing data. Quite often, developers must resolve incompatibilities in data formats themselves. This is a time consuming task that is almost unavoidable in the development of any distributed application.

From the onset, Creative Science Systems designed NetZyme® to have a platform and language independent data encapsulation facility. The Netzyme CORE connects different platforms seamlessly. The Netzyme API provides development tools to overcome the intrinsic barrier of differing data formats and protocols.

2.2. NetZyme® Architecture

The NetZyme® infrastructure supports multi-tiered application deployment by providing:

- **An extendable API**—the Netzyme API is a collection all the base classes needed to integrate different systems and data formats. It was built to enable truly distributed systems to operate independently of language and protocol types. See chapters 3, 5, and 11 for details about the Netzyme API and chapter 7 for Palm OS and Pocket PC development tools.
- **Middleware support**—A complete suite of integration tools to utilize SSL, CORBA, Web Services, and JMS for your applications. See chapters 4, 5, 6, 8, 9, and 10 for information about these subjects.
- **Uniform data presentation**—Netzyme provides platform independent data encapsulation methods to handle simple and complex data objects as well as tools to generalize your current distributed applications to allow them to communicate with any platform in any programming language. See chapters 3, 5, and 10 for details on data presentation and integration.
- **Application servers (Netzyme Enterprise Server)**—a family of application servers including a generic and extendable socket based application server that provides a robust set of functions. See chapters 5 and 11 for details on the Netzyme Enterprise Server.

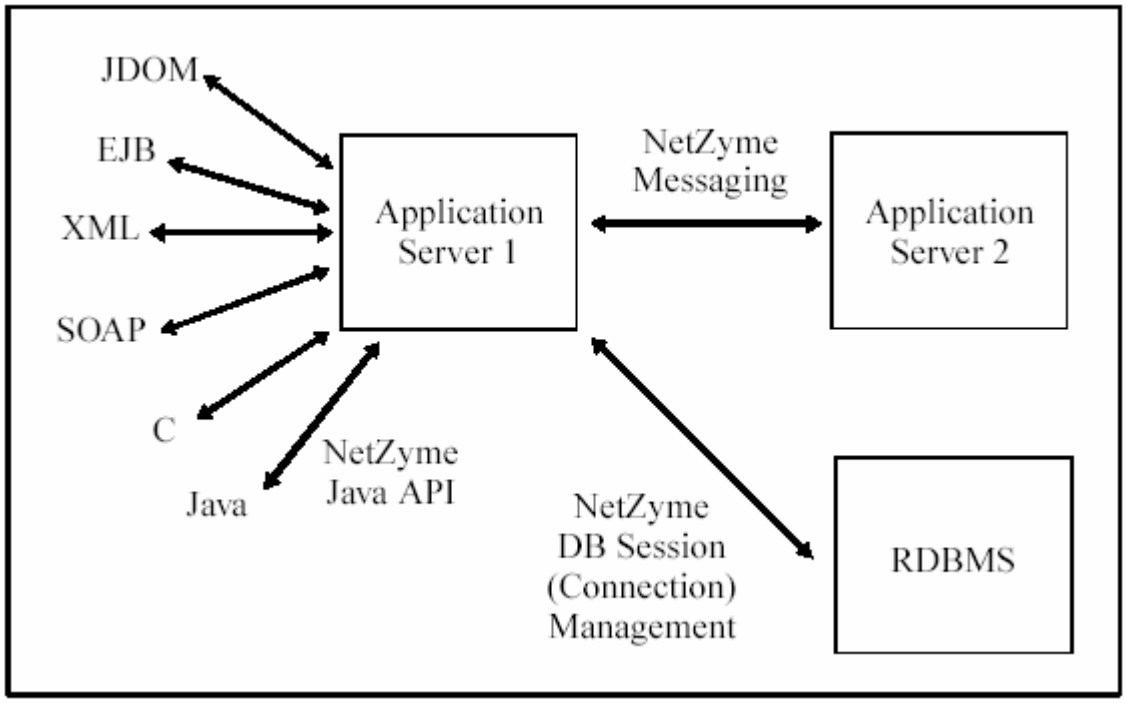

Figure 2-4: *Netzyme Software Infrastructure*

Figure 2-4 demonstrates the robust variety of languages and capabilities that have been integrated into the Netzyme software. It integrates languages, Database management, and server integration into one complete infrastructure.

2.2.1. NetZyme® API

The latest release of Netzyme includes C and Java APIs with which to develop distributed applications. These libraries are designed to enable cross-platform development and efficient data transfer between tiers. Each class in each NZ-API package performs optimally. A committee of senior engineers determines selection of each class to add to the NZ-API. These engineers continue the committee review as the class is designed and verify that the class is built properly. All classes are put through rigorous testing so that enterprise systems can be built with this technology and work reliably.

Chapter 3, the next step in this training, will familiarize you with many of the useful capabilities of the API. We include examples written by our engineers to demonstrate its capabilities.

2.2.2. Middleware Support

NetZyme® Enterprise manifests itself as a server. By virtue of supporting such protocols as HTTP, SOAP, .NET, CORBA, JMS, etc., any clients can communicate with it and fully utilize facilities exported over a given protocol.

In addition, NetZyme® has specific protocol for efficient transactions in Remote Object Invocation (ROI) fashion. To enable it, NetZyme® specific communications over ROI

protocol for external clients, classes for Java based clients and libraries for C based clients are provided.

Full data representation compatibility between C and Java is maintained, thus enabling seamless integration between different languages capable of utilizing C libraries and Java. These capabilities allow you to use Extensible Markup Language (XML), HTML, binary, and schema data types for your client/server applications.

As an illustration of the foundation approach to enterprise system extendibility, let's take a look the XML support furnished by NetZyme®. This support was added on top of preexisting data encapsulation in such a way that earlier developed applications can be easily XML-enabled, in most cases with absolutely no code changes.

A great deal of processing is needed for converting raw data into a presentable format. The majority of this processing can be done by the NetZyme® XML facilities. Consequently, as with the NetZyme® communication and data encapsulation facilities, the use of XML in conjunction with NetZyme® removes yet another layer of unnecessary complexity from the development process.

NetZyme® also augments the best possible XML performance. Creative Science Systems designed the XML facilities from the ground up and discovered that when improperly used, XML can be extremely slow and sluggish.

By using NetZyme®, you don't have to spend time building such facilities or worrying later if they're going to work correctly, NetZyme® does it all for you.

2.2.2.1. XML Schema

In addition to handling providing XML capabilities, NetZyme® also provides an XML Schema (as defined by W3C recommendation) to Java compiler that takes any XML Schema document, validates it and, if the document is correct, generates java classes to activate the schema. This transfers the logic of working with XML documents to a programmatic level that allows you to manipulate XML data directly through a standard Java class manipulation facility. The generated classes allow run-time marshalling of XML documents into instances of 'normal' Java classes as well as un-marshalling instances of generated Java classes with specific data into an XML document 'on-the-fly'.

This option provides robust and error-free XML document processing since all the logic of run-time data validation and conversion is handled automatically.

2.2.2.2. J2EE EJB Container

NetZyme® infrastructure also provides all the necessary components to support a J2EE environment by virtue of a fully implemented EJB Container server with embedded JMS capabilities. One would think that there is no need to provide yet another EJB Server but in the current specification of the EJB Container Server, there are no provisions for either fault tolerance or processor load balancing. That makes any EJB Server a point of potential failure in the enterprise environment as well as a bottleneck for increasing throughput. NetZyme® removes all these issues by introducing its enterprise grade capabilities to handle failures and loads, making a NetZyme® EJB Container server an equal participant in the enterprise environment

to other fault tolerant server technologies. Clustering, fail-over and load balancing are enabled by simply changing configuration files.

The NetZyme® JMS implementation also addresses another area that is lacking in standard JMS specification - message routing. NetZyme has the ability to dynamically bind topic/destination information for messages using pattern-based message routing that highly compliments standard JMS facilities. All of components that are used by EJB Container server are created as an open API that allows you to use all the components such as JMS, JTM, resource management etc. with or without EJB Container server for other development purposes.

2.2.3. Uniform data exchange

NetZyme® applications benefit from universal data exchange objects. *TdObjects* (Tagged Data Objects) provide uniform connectivity to industry standard DB engines. *TdxObjects* and *XsltObjects* implement XML document functionality, allowing data presentation to be handled efficiently and independent of business logic. Because *XsltObjects* provide inherent use of XSL style sheets, business and presentation logic can be developed in parallel and later merged using the rules set forth by an XSL style sheet. In essence, this reduces project development time, simplifies application testing and debugging and minimizes the effort needed to implement application modifications.

2.2.3.1. Serverlizer™

One of the intrinsic difficulties in exchanging data uniformly is providing client applications that will run on any platform. The serverlizer, a netzyme Rapid Application Development (RAD) tool that changes monolithic applications into distributed ones, offers the capability of automatically producing clients for a variety of platforms regardless of the native language of your original application.

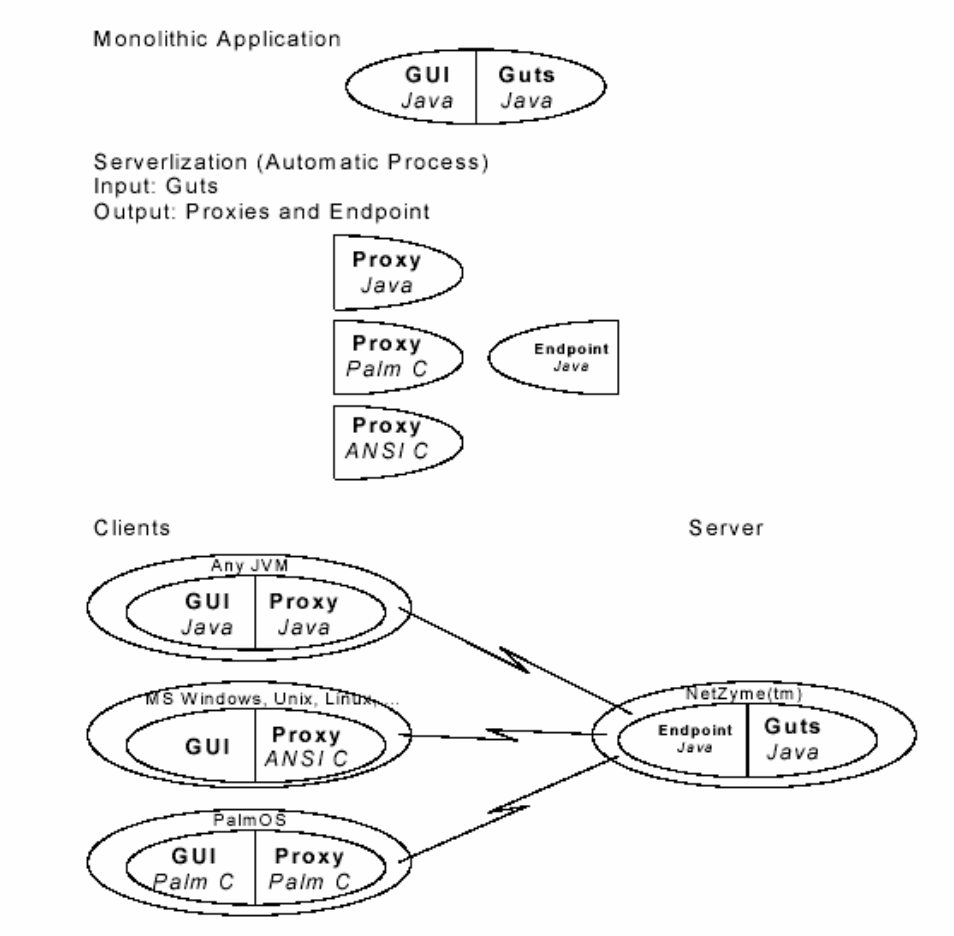

Figure 2-5: *A Serverlizer conversion from a monolithic, language specific application to a distributed, client-server, multi-platform application*

This also handles the challenge of building a distributed application by transforming a monolithic application into scalable, fault tolerant client/server system, which can account for, and accommodate with constant performance quality, the inherently unreliable connectivity, misbehaved clients, and watertight information security. This example takes a monolithic Java application and dynamically generates client proxies and GUIs to connect your clients.

Transforming a monolithic application into a Java based client/server n-tier architecture application is greatly simplified by leveraging of a software infrastructure backbone, which streamlines a number of cumbersome processes.

NetZyme® also provides handler API that allows seamless interoperability between NetZyme® and legacy systems by means of creating special purpose request handlers that on one hand provide access to rich processing power of NetZyme® and on the other hand interoperability with existing systems. This is the primary means of integrating CORBA and SOAP, as well as other systems.

2.2.4. Netzyme Enterprise Server

NetZyme® Enterprise servers are socket based multi-thread application servers. The final chapter of this tutorial will acquaint you with how to administer the server, build

and deploy applications, configure fault tolerance, and balance server loads according to your specifications. NetZyme® Enterprise servers provide:

- Global and application level data storage
- Session management
- Database connection management
- Event dispatcher
- Thread pool manager
- Communication manager
- Run-time Java compiler
- Run-time class loader and invoker
- Application builder

The general purpose server functions are designed allow you to use its components for specific duties such as session management or DB connection management.

Application servers can be configured to provide specific functionality to improve throughput to handle high volume requests. For example, a server farm of application servers can be configured to specifically handle session management duties where the volume of sessions is very high. Session objects can be kept in server's memory or for increased reliability stored within a database, in which case the server's db connection manager functionality could be used to improve storage and retrieval of session objects.

Scaling with NetZyme® application server is easy. Adding new servers to a server farm can be made as easy as copying the application to the new server and modifying a configuration file on the master or gating server.

The NetZyme® application server supports:

- Dynamic UI generation (HTML, XML/XSL, WML, for example)
- Java based business logic
- An infrastructure that supports: load balancing, messaging, session management, middleware

2.2.4.1. Configuration Management

Good configuration allows an application to adapt dynamically, so recompilation is unnecessary. It is good development practice to expect changes by making a piece of information configurable — this includes database information (name, location, and user name/password), modes of functionality (e.g. demo vs. licensed), expected server IP's (for client components), and other such pieces of information. The NetZyme® configuration facility makes it very easy to deal with configuration issues. It provides a full suite of configuration data management classes. With these, you can change configurations during runtime.

Developers do not need to think of configuration data in terms of a file. A file is only one medium to persistently store configuration data. Configuration data can be generated internally by an application and then written to a file, database, or passed across a network. This makes configuration easier to access, manipulate, and store.

2.3. NetZyme® and Other Distributed Application Technologies

NetZyme® is implemented as pure Java and supports Java 2. Consequently, it is compatible with any technology supported by Java. This very powerful feature allows developers to easily connect their NetZyme® applications to other emerging technologies, which includes Enterprise Java Beans, CORBA, and Sun's RMI implementation. It has recently integrated the new .NET standard into its broad interoperability framework. This addition was relatively easy by the foundation strategy used to create Netzyme®.

The NetZyme® development team has been working with Java since the pre-alpha versions of the language were released in 1996. The NetZyme® inventors are Sun Microsystems certified Java instructors. This extensive and rather intimate knowledge of Java is the reason why NetZyme® performance is surpassing any known Java implementations, approaching C language benchmarks.

2.3.1. Sockets and Sessions

This note is to bring you up to speed on TCP Sockets and Client-Server Sessions. We have tested and can verify that the Netzyme Server can support over 50,000 concurrent Client-Server Sessions. This speaks to its excellent expandability. However, the hardware and Operating System may not support the number of sockets necessary to use all of these sessions. This is not a fundamental design flaw but rather a limit inherent to all networked systems.

A session is a memory based object and thus requires only RAM and processor bandwidth to exist. Because most computers have a lot of RAM and session objects are usually relatively small, the number of session objects is usually only limited by the software that constructs the objects. The Netzyme server software can create and manage a very large number of session objects in both the client and the server.

A TCP socket is a logical network path that is generated by a combination of the hardware and the Operating System that manages your network device(s). Even if your hardware could handle an unlimited number of sockets, your Operating System may support only a few or provide only a certain number before it locks up due to the load on the network device that it is trying to manage. Likewise, a slower network device will handle fewer sockets at a time. Therefore, although Netzyme will handle over 50,000 concurrent sessions, you will not be able to use them all without the appropriate hardware.

2.4. Benchmarks

2.4.1. Footprint

Minimum installation (on top of JRE and necessary 3rd party' classes)	800 KB
Full installation (on top of JRE and necessary 3rd party' classes)	4 MB

2.4.2. Performance

Table 1: Web Based Application Throughput

Hardware	Environment	Measurements

Single CPU 400 MHz PII, 128 Meg RAM	Web-bound Round Trip (Browser, Apache HTTPd, NetZyme® Enterprise AppServer)	20 ms and up, depending on the complexity of business logic processing
	Web-bound Round Trip (Browser, Apache HTTPd, NetZyme® based AppServer, Oracle)	80 ms and up, depending on the complexity of DB queries
	Bare Messaging Routing, UDP based	Up to 200 messages per second
CPU 700 MHz PIII, 128 Meg RAM	Web-bound Round Trip (Browser, Apache HTTPd, NetZyme® Enterprise AppServer)	15 ms and up, depending on the complexity of business logic processing
	Web-bound Round Trip (Browser, Apache HTTPd, NetZyme® based AppServer, Oracle)	60 ms and up, depending on the complexity of DB queries
	Bare Messaging Routing, UDP based	Up to 300 messages per second

Table 2: Sample web site throughput

Hardware	Environment	Measurements
Two double CPU boxes (600Mhz PIII, 1G RAM)	One box – NT based NetZyme® Enterprise; Another box – NT Based Oracle 8i	Up to 5 million hits per day

2.5. Check yourself

Before continuing on to the next module, check that you are able to do or answer the following:

- What class of software does the NetZyme® Enterprise belong to?
- List advantages of using NetZyme® Enterprise middleware.
- What 6 core concepts of distributed computing does Netzyme® support?
- Name the 4 main components of NetZyme® Enterprise Architecture and describe their functionality.

2.6. Related Documentation

The White papers for Netzyme can be found on the web at:

http://www.creativescience.com