



## Building a Service Oriented Network

Building a Service Oriented Network using a Service Delivery Controller

Zeus Technology Limited (UK) Sales: +44 (0)1223 568555  
The Jeffreys Building Main: +44 (0)1223 525000  
Cowley Road Fax: +44 (0)1223 525100  
Cambridge CB4 0WS Email: [info@zeus.com](mailto:info@zeus.com)  
United Kingdom Web: [www.zeus.com](http://www.zeus.com)

Zeus Technology, Inc. (U.S.) Phone: 1-888-ZEUS-INC  
1955 Landings Drive Fax: (866) 628-7884  
Mountain View  
CA 94043 Email: [info@zeus.com](mailto:info@zeus.com)  
United States of America Web: [www.zeus.com](http://www.zeus.com)

# Contents

|   |           |
|---|-----------|
| <b>Introduction .....</b>   | <b>3</b>  |
| What has SOA changed? .....   | 3         |
| The Service Oriented Network.....   | 3         |
| <b>Building a Service Oriented Network .....</b>                            | <b>4</b>  |
| Application Delivery Controllers.....                                       | 4         |
| Service Delivery Controllers .....  | 5         |
| Understanding XML .....   | 5         |
| Definition – A Service Delivery Controller .....                            | 5         |
| <b>Using a Service Delivery Controller.....</b>                             | <b>6</b>  |
| Load Balancing and Scalability .....  | 6         |
| Scaling Individual SOA Components .....                                     | 6         |
| Improved reliability .....  | 6         |
| Intelligent Error Detection .....   | 7         |
| Easier administration .....   | 7         |
| Performance .....   | 7         |
| <b>Programming the Service Oriented Network.....</b>                        | <b>8</b>  |
| A Flexible, Adaptable Network driven by TrafficScript™ .....                | 8         |
| TrafficScript™ .....  | 8         |
| An Agile Network controlled by the ZXTM Control API.....                    | 9         |
| The ZXTM Control API.....   | 9         |
| The Enterprise Service Bus .....  | 10        |
| <b>Conclusion .....</b>   | <b>11</b> |
| <b>Appendix 1: Programming the network using TrafficScript™.....</b>        | <b>12</b> |
| 1. Routing SOAP traffic .....   | 12        |
| 2. Ensuring fair access to resources.....                                   | 13        |
| 3. Securing Traffic.....  | 14        |
| 4. Validating SOAP responses .....  | 15        |
| <b>Appendix 2: Programming the network using the ZXTM Control API .....</b> | <b>16</b> |
| 1. Adding a new server .....  | 16        |
| 2. Draining a Server .....  | 16        |
| 3. Retrieving Event Logs .....  | 17        |



## Introduction

The Service Oriented Architecture (SOA) approach is revolutionizing the way that many organizations build their applications.

SOA promises an architecture that is more flexible, responsive, and easy to align with the needs of the business. However, unlike the monolithic applications that SOA replaces, SOA applications are critically dependent on the network for every part of their operation.

### What has SOA changed?

SOA applications are built from loosely coupled services named 'Web Services'. These components communicate with each other using a protocol called 'SOAP' (Simple Object Access Protocol), which transfers XML messages using HTTP – the transport mechanism used by web servers and other web applications.

Each transaction in an SOA application may hit many separate web services. Amazon.com has been using SOA techniques for some time, and Werner Vogels (Amazon.com's CTO) described how a single page request to amazon.com can access more than 100 separate SOA-based services<sup>1</sup>.

*Adopting SOA doesn't mean a forklift upgrade of your network infrastructure. With programmable traffic management devices, you can easily add the intelligence you need to support this new application architecture.*

### The Service Oriented Network

Clearly, the success of an SOA application depends critically on the infrastructure's ability to pass these messages between the SOA components. The network takes a key part in this, transmitting data rapidly and reliably, but the demands of SOA go far beyond this.

If an individual SOA component fails or becomes overloaded, messages should be load-balanced to other components that provide the same service. As components are added, upgraded or moved, the messages need to be routed accordingly. Performance problems need to be diagnosed, flash-floods of SOA traffic managed, and processor-intensive operations like SSL decryption and XML normalization need to be offloaded from the SOA components to improve their capacity.

None of these services are provided by SOA components themselves. Instead, a new concept – the Service Oriented Network (SON) – has emerged. The SON acknowledges that message transmission is the role of the network, but that new services are needed to support the needs of SOA applications. In essence, SON recognizes that the network needs to become as agile, flexible and nimble as the SOA applications it is supporting.

SON is not an expensive, disruptive forklift upgrade of your existing network hardware. Instead, an SON is created by building these services on top of an existing network, using proven and mature solutions, in order to realize the full promise of SOA.

---

<sup>1</sup> <http://www.acmqueue.com/modules.php?name=Content&pa=showpage&pid=388>



# Building a Service Oriented Network

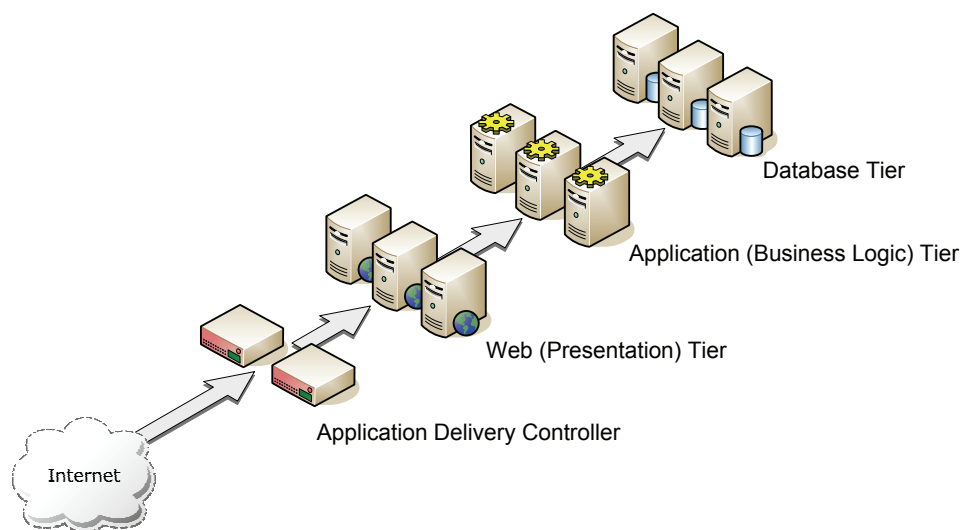
## Application Delivery Controllers

The idea of running applications across the network is not new. The World Wide Web itself is over 15 years old, but is predated by many other network-based applications. The challenges that complex web sites face, and the technologies that they use, are not far from the challenges that SOA applications face today.

Initial problems like service availability and scalability were addressed by network-focused load balancing products, but as the technology got smarter, a new set of products, dubbed 'Application Traffic Managers' or 'Application Delivery Controllers' emerged.

These traffic management devices span the network/application divide. Although they are sometimes regarded as network appliances, products like F5 Network's BigIP LTM, Citrix's Netscaler and Zeus' ZXTM are application-level proxies that inspect, transform and route network traffic and can fully understand the semantics and requirements of each transaction they manage. F5 and Zeus even include entire programming environments (iRules™ and TrafficScript™ respectively) so that they can be configured with immense precision to make the best possible use of available network and application resources.

Application Delivery Controllers (ADCs) are commonly deployed at the edge of web-based applications, managing external requests to the application like a gatekeeper. They improve the security of the web-based application, greatly increase its capacity, offload generic processing tasks like encryption and compression, and shape and route traffic to ensure maximum availability and resilience to the effects of traffic floods.



*ADCs manage traffic entering and leaving a multi-tiered application*

ADCs are also sometimes deployed between the tiers in large applications, to deliver reliability and resilience should a component in one of the tiers fail or become overloaded.

The benefits that ADCs bring to traditional applications closely match the requirements of a Service Oriented Network.



## Service Delivery Controllers

ADCs are intimately familiar with protocols like HTTP, used by web clients to communicate with web-based applications. HTTP is a core protocol in SOAP (Simple Object Access Protocol), the most common means that SOA Web Services use to communicate with each other. However, understanding HTTP alone is not sufficient to understand and manage SOA traffic.

### Understanding XML

The messages that are sent using SOAP are formatted in XML. XML is a very rich, highly structured type of data, with very strict rules on how the messages are structured and interpreted. The traditional methods of processing data used by Application Traffic Managers – search, replace, regular expressions – cannot be used to process XML data.

*Understanding HTTP alone is not sufficient to understand and manage SOA traffic. A Service Delivery Controller can also understand and manipulate the XML messages used within the SOA app.*

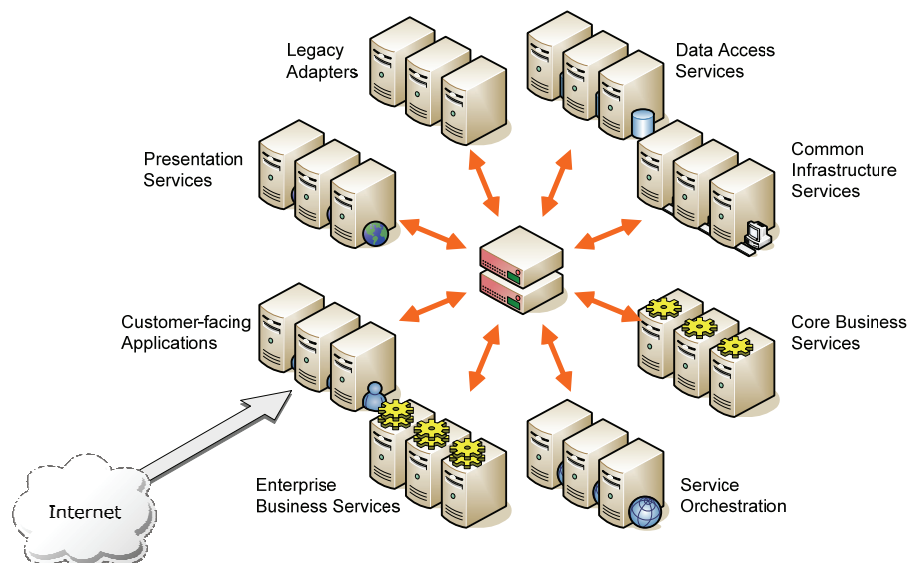
An ADC must understand XML in order to process and manage SOA traffic. The XML industry has developed several standards for this task. The XPath standard provides a means to search XML documents for particular information, and the XSLT standard provides a way to process and transform XML from one form to another. Together, these two standards allow you to fully manage XML data reliably and in a familiar, standards-compliant way.

Several modern ADCs – most notably, Zeus' ZXTM traffic manager and Cisco's AON and ACE products – support these XML-processing standards.

### Definition – A Service Delivery Controller

*A Service Delivery Controller (SDC) is an ADC that understands the XML messages used by SOA systems.*

Embedding a Service Delivery Controller deep inside an SOA application is precisely how organizations go about building a Service Oriented Network.



## Using a Service Delivery Controller

To begin with, a Service Delivery Controller (SDC) brings the same benefits to your SOA application that an Application Delivery Controller brings to a web-based application.

### Load Balancing and Scalability

An SDC lets you safely scale your SOA components, running several redundant instances and load-balancing requests across each. Even if your component is not designed to be clustered, session persistence logic in the SDC can ensure that sessions are detected and pinned to the same instance of the component.

The advanced load balancing capabilities of Zeus' ZXTM product are very sensitive to the response time of each component instance and the number of outstanding requests, so they can cleverly send requests to the components likely to respond most quickly.

### Scaling Individual SOA Components

Individual components can be scaled independently. For example, suppose that you had an ecommerce application that contained a 'people who viewed this item also viewed these' SOA-based service. As you added more and more items to your inventory, that particular SOA service became computationally more and more complex. Using your Service Oriented Network, you could then deploy a second server to run this component and use ZXTM to load-balance SOA traffic across the two servers.

### Improved reliability

SOA applications are much more fragile than traditional monolithic applications. This is because they are built up of multiple, loosely connected components, many of which may be used by many different SOA applications.

*Without clustering using an SDC, your SOA application will become more and more unreliable as it depends on more components. The error rates of each component will accumulate to affect the integrity of the entire application.*

If one component were to fail or to suffer a performance problem because of a denial-of-service attack or programming error, the effects would ripple through and affect all of the applications that depended on this component. An application can fail because of a single component failure or even because an unrelated application suffered a denial-of-service attack.

Suppose that each component has 99.99% availability (52 minutes downtime per year). An SOA application that depends on 10 components could be expected to have 99.9% availability<sup>2</sup>, or almost 9 hours of downtime per year. An SOA application that depended directly or indirectly on 100 components would have 99% availability, or almost 90 hours of downtime.

With a Service Delivery Controller, it's easy to deploy multiple instances of each component and load-balance between them. If one instance fails, the SDC can direct all SOA traffic to

---

<sup>2</sup>  $99.99\%^{10} = 99.9\%$ . As a rule-of-thumb, multiply the average downtime by the number of components (52 minutes \* 10 = 520 minutes)



the remaining components so that the dependent applications continue to function perfectly. With ZXTM, you deploy failover configurations of the SDC itself so that the application is unaffected even if one SDC device were to fail.

### **Intelligent Error Detection**

You can even teach an SDC like ZXTM what a correct response looks like, so if a component starts returning unexpected web services responses (for example, with embedded fault codes), the SDC can avoid it in the future until it recovers.

### **Easier administration**

The failover and clustering also brings significant benefits when it comes to routine maintenance. Individual components can be upgraded without suffering any downtime at all. This capability is realized using ZXTM's draining capability; when a component is 'drained' it is allowed to complete any outstanding transactions, but no new transactions are sent to it. Within a cluster of components, each one can be drained, shut down, upgraded, restarted and undrained in turn.

## **Performance**

The application acceleration capabilities of an SDC will improve the capacity and performance of your SOA application.

TCP buffering and HTTP keepalive processing will reduce the number of concurrent requests that an SOA component processes. This will dramatically improve its scalability, even over a local, low-latency network.

Common processing tasks can be offloaded from the SOA components onto the specialized SDC implementation. Tasks that can be offloaded include SSL encryption, HTTP compression and XML normalization, and common SOA responses can be cached. All of these processes reduce the load on the SOA platform, releasing more resources for application-specific tasks and giving better scalability and return on investment.



## Programming the Service Oriented Network

Load balancing, scalability, improved reliability and better performance are each important benefits, but a Service Oriented Network goes much further. In order to support the SOA applications, the network needs to be flexible and adaptable to their specific needs, and sufficiently agile that it can be tuned for new business workloads when required.

These benefits come from the ability to program an SDC like ZXTM, and the ability to configure it automatically when conditions change. The programmability makes the network uniquely flexible and adaptable to the needs of your SOA applications.

### A Flexible, Adaptable Network driven by TrafficScript™

The magic in a Service Delivery Controller like ZXTM comes from two sources – the wide range of features and the programmable nature that lets you use these features in precisely the way you want.

*The wide range of optimization and traffic management features can be used in precisely the way you want, using the rich and powerful programming environment.*

#### TrafficScript™

ZXTM includes a programming language called TrafficScript™. The TrafficScript language allows an administrator to write traffic management rules that can inspect incoming and outgoing SOA traffic, transform it in many different ways (for example, using XSLT), route each request according to its payload, and enable or disable various capabilities of ZXTM for each request.

This is the key to making your network adaptable to the requirements of the SOA applications. You can drive the behavior of your network completely using the TrafficScript programming language:

- **Inspect and route SOA traffic** based on the precise contents and meaning of each message and external factors such as system load
- **Apply SOA Service Governance**, monitor the usage and performance of each SOA component and apply rate limits or service blocks if an application uses more than its quota of resources. **Prioritize some SOA requests** over others, and **implement SOA service contracts**, so that key applications always have access to the resources they need.
- **Secure SOA Traffic**, checking requests match published interfaces, are correctly authenticated and do not exploit application vulnerabilities or weaknesses
- **Validate SOA responses**, ensuring that they meet desired criteria and do not contain sensitive information. **Retry SOA requests** against multiple components to work round transient service failures (for example, caused by application restarts or overloading).

Appendix 1 of this document provides 4 such examples of managing SOA traffic using TrafficScript. For more information, please refer to the ZXTM KnowledgeHub website at <http://knowledgehub.zeus.com/>.



## An Agile Network controlled by the ZXTM Control API

SOA applications can be extremely fluid. New services can be brought online rapidly, and new instances added to clusters of components to scale their capacity. With an appropriate management system, these changes can be automated and can even occur on demand.

For example, on the last Monday of each month, an organization may wish to suspend background processing tasks and reallocate those resources to a payroll processing system. In another example, an organization may wish to be able to respond to external events rapidly, reprioritizing transactions in a ticket booking system when a marketing promotion is launched.

The trend to virtualization in the datacenter facilitates this agility. Using virtualization, it is extremely easy to deploy new servers and applications, and businesses will wish to take advantage of this so that they can manage their available resources in a more intelligent, cost-effective way.

*A Service Oriented Network must be agile, able to support rapid and frequent changes to SOA applications. The ZXTM Control API makes it possible to monitor and automate changes.*

### The ZXTM Control API

The ZXTM Control API makes it easy to automatically configure and interrogate the ZXTM Service Delivery Controllers. It is a standards-based Web Service that can be invoked from external agents such as a provisioning system, network monitoring system or intrusion detection system.

The Control API is the means by which the network becomes agile, able to be rapidly reconfigured to support new applications and business requirements. The configuration of the Service Delivery Controller is no longer static but can be updated and modified on demand.

- **Standards Compliant:** The Control API is a SOAP-based Web Service. It uses the same interface that SOA components use to communicate with each other. The API can be invoked by any application environment that supports SOAP.
- **Fully Featured:** The Control API is just as rich as ZXTM's web-based user interface. Any configuration change that can be made using the user interface can be automated using the Control API. Similarly, all diagnostic information and logs that can be viewed in the user interface can be reached through the Control API.
- **Full Audit Trail:** Remote agents that reconfigure ZXTM must authenticate themselves first. Each authentication and all configuration changes are logged for audit purposes.

For example, when a new service component is brought online, the Control API can be used to inform ZXTM to load-balance SOA traffic to that new component. If a component is due to be decommissioned, ZXTM can be informed to drain traffic from that component.

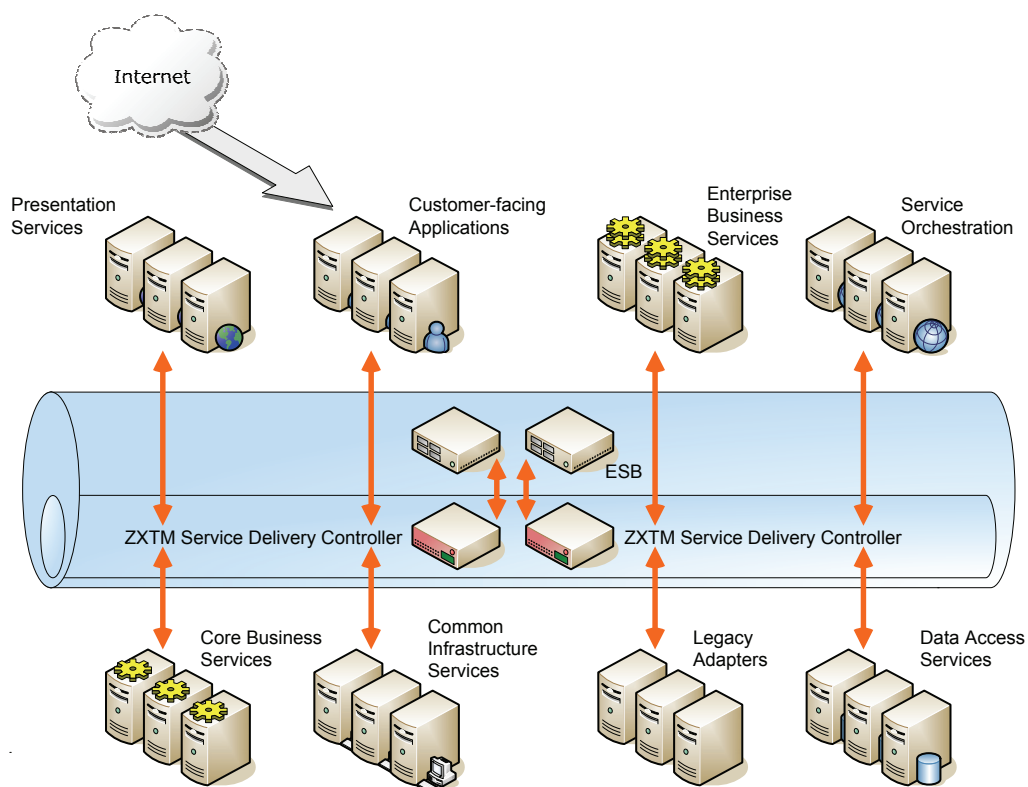
Appendix 2 of this document provides examples of how to reconfigure the network using the ZXTM Control API. For more information, please refer to the documentation at <http://knowledgehub.zeus.com/docs/>.



## The Enterprise Service Bus

An SDC fits best when considered as a front-end to an Enterprise Service Bus (ESB). ESBs implement many key capabilities within an SOA system – orchestration, routing and asynchronous messaging – but sitting in the middle of the SOA network, they create their own choke point and point of failure.

An SDC can sit in front of an ESB, or better still, a high-availability cluster of ESBs. It can manage traffic on behalf of the ESB, performing tasks it is specialized for (such as SSL offload, XML processing, health monitoring, performance monitoring and content-based routing) without having to invoke the ESB. Only messages which need special handling need be passed up the stack to the ESB.



*The ZXTM Service Delivery Controller provides XSLT Transformation, Routing, Failover, Scaling, Draining, Offload and Acceleration, Prioritization, Service Level Monitoring and Security and Validation services on behalf of the Enterprise Service Bus.*

Customers who have used Zeus' ZXTM traffic manager in this way have reported dramatic performance improvements in their ESB deployment:

- 15-fold increase in transactions per second and 15-times reduction in response time when performing XSLT transformations between schemas
- 15-to-20-fold increase in transactions per second for content-based routing



## Conclusion

Application Delivery Controllers (ADCs) are mature, proven systems that bring improved performance, reliability and scalability to networked services like web sites and applications. They can bring exactly the same benefits to complex SOA applications.

However, the potential of Service Oriented Networks - network infrastructure with the flexibility, adaptability and agility to support SOA applications - goes far beyond that. This promise is realized by using a Service Delivery Controller to add services on top of the network hardware.

The ZXTM Application Delivery Controller can be used most effectively in this manner, taking advantage of its flexible TrafficScript™ configuration language, full XML awareness and standards-based ZXTM Control API.

ZXTM is the only ADC with a software footprint, so it can be deployed on server hardware and in virtual hardware, for maximum flexibility. To find out more, and try ZXTM solutions for yourself, please contact Zeus Technology at [www.zeus.com](http://www.zeus.com) and [knowledgehub.zeus.com](http://knowledgehub.zeus.com).



## Appendix I: Programming the network using TrafficScript™

Here are several examples showing how TrafficScript can route and manage SOA network transactions.

### I. Routing SOAP traffic

Let's say that the network is handling requests for a number of different SOAP methods. The traffic manager is the single access point – all SOAP traffic is directed to it. Behind the scenes, some of the methods have dedicated SOAP servers because they are particularly resource intensive; all other methods are handled by a common set of servers.

The following example uses ZXTM's pools. A pool is a group of servers that provide the same service. Individual pools have been created for some SOA components, and a 'SOAP-Common-Servers' pool contains the nodes that host the common SOA components.

```
# Obtain the XML body of the SOAP request
$request = http.getBody();

$namespace = "xmlns:SOAP-ENV=\"http://schemas.xmlsoap.org/soap/envelope/\"";
$xmlpath = "/SOAP-ENV:Envelope/SOAP-ENV:Body/*[1]";

# Extract the SOAP method using an XPath expression
$xmlmethod = xml.XPath.matchNodeSet( $request, $namespace, $xmlpath );

# For 'special' SOAP methods, we have a dedicated pool of servers for each
if( pool.getActiveNodes( "SOAP-".$xmlmethod ) > 0 ) {
    pool.select( "SOAP-".$xmlmethod );
} else {
    pool.select( "SOAP-Common-Servers" );
}
```

Code sample 1: Routing SOAP requests according to the method

#### Why is this useful?

This allows you to deploy services in a very flexible manner. When a new instance of a service is added, you do not need to modify every caller that may invoke this service. Instead, you need only add the service endpoint to the relevant pool.

You can rapidly move a service from one server to another, for resourcing or security reasons (red zone, green zone), and an application can be easily built from services that are found in different locations.



## 2. Ensuring fair access to resources

With ZXTM, you can also monitor the performance of each pool of servers to determine which SOAP methods are running the slowest. This can help troubleshoot performance problems and inform decisions to re-provision resources where they are needed the most.

You can shape traffic – bandwidth or transactions per second – to limit the resources used and smooth out flash floods of traffic. With the programmability, you can shape different types of traffic in different ways. For example, the following TrafficScript™ code sample extracts a 'username' node from the SOAP request. It then rate-shapes SOAP requests so that each remote source (identified by remote IP address and 'username' node value) can submit SOAP requests at a maximum of 60 times per minute:

```
# Obtain the source of the request
$source = request.getRemoteIP();

# Obtain the XML body of the SOAP request
$request = http.getBody();

$namespace = "xmlns:SOAP-ENV=\"http://schemas.xmlsoap.org/soap/envelope/\"";
$xmlpath = "/SOAP-ENV:Envelope/SOAP-ENV:Body/*/username/text()";

# Extract the username using an XPath expression
$username = xml.XPath.matchNodeSet( $request, $namespace, $xmlpath );

# $key uniquely identifies this type of request from this source.
$key = $ip . ", " . $username;

# The 'transactions' rate shaping class limits each type to 60 per minute
rate.use( "transactions", $key );
```

Code sample 2: Rate-shaping different users of SOAP traffic

### Why is this important?

An SOA component may be used by multiple different SOA applications. Different applications may have different business priorities, so you might wish to prioritize some requests to a component over others. Applying 'service governance' policies using ZXTM's rate shaping functionality ensures that all SOA applications get fair and appropriate access to critical components, and that no one application can overwhelm a component to the detriment of other applications.

This can be compared to time-sharing systems – each SOA application is a different 'user', and users can be granted specific access to resources, with individual limits where required.

When some SOA applications are externally accessible (via a web-based application for example), this is particularly important because a flash flood or malicious denial-of-service attack could ripple through, affecting many internal SOA components and internal applications.



### 3. Securing Traffic

Suppose that someone created a web services component for a travel company that enumerated all of the possible flights from one location to another on a particular day. The caller of the component could specify how many hops they were prepared to endure on the journey.

Unfortunately, once the component was deployed, a serious bug was found. If a caller asked for a journey with the same start and finish, the component got stuck in an infinite loop. If a caller asked for a journey with a large number of hops (1000 hops perhaps), the computation cost grew exponentially, creating a simple, effective denial of service attack.

Fixing the component is obviously the preferred solution, but it's not always possible to do so in a timely fashion. Often, procedural barriers make it difficult to make changes to a live application. However, by controlling and manipulating the SOA requests as they travel over the network, you can very quickly roll out a security rule on your SDC to drop or modify the SOAP request. Here's a snippet:

```
$request = http.getBody();

$from = xml.XPath.matchNodeSet( $request, $namespace, "//from/text()" );
$dest = xml.XPath.matchNodeSet( $request, $namespace, "//dest/text()" );

# The error response; can read a precanned response from disk and return
# it as a SOAP response
if( $from == $dest ) {
    $response = resource.get( "FlightPathFaultResponse.xml" );
    connection.sendResponse( $response );
}

$hops = xml.XPath.matchNodeSet( $request, $namespace, "//maxhops/text()" );
if( $hops > 3 ) {
    # Apply an XSLT that sets the hops node to 3
    $transform = resource.get( "FlightPath3Hops.xslt" );
    http.setBody( xml.XSLT.transform( $request, $transform ) );
}
```

Code sample 3: Checking validity of SOAP requests

#### Why is this important?

Using the Service Delivery Controller to manage and rewrite SOA traffic is a very rapid and lightweight alternative to rewriting SOA components.

Patching the application in this way may not be a permanent solution, although it's often sufficient to resolve problems. The real benefit is that once a fault is detected, it can be resolved quickly, without requiring in-depth knowledge of the application. Development staff need not be pulled away from other projects immediately. A full application-level fix can wait until the staff and resources are available; for example, at the next planned update of the component code.



## 4. Validating SOAP responses

If a SOAP server encounters an error, it may still return a valid SOAP response with a 'Fault' element inside.

If you can look deep inside the SOAP response, you've got a great opportunity to work around such transient application errors. If a server returns a fault message where the faultcode indicates there was a problem with the server, wouldn't it be great if you could retry the request against a different SOAP server in the cluster?

```
$response = http.getResponseBody();

$ns = "xmlns:SOAP-ENV=\"http://schemas.xmlsoap.org/soap/envelope/\"";
$xmlpath = "/SOAP-ENV:Envelope/SOAP-ENV:Body/SOAP-ENV:Fault/faultcode/text()";

$xmlfaultcode = xml.XPath.matchNodeSet( $request, $namespace, $xmlpath );

if( string.endsWith( $xmlfaultcode, "Server" ) ) {
    if( request.retries() < 2 ) {
        request.avoidNode( connection.getNode() );
        request.retry();
    }
}
```

Code sample 4: If we receive a *Server* fault code in the SOAP response, retry the request at most 2 times against different servers.

### Why is this important?

This particular example shows how the error checking used by an SDC can be greatly extended to detect a wide range of errors, even in responses that appear "correct" to less intelligent traffic managers.

It is one example of a wide range of applications where responses can be verified, scrubbed and filtered. Undesirable responses may include fault codes, sensitive information (like credit card or social security numbers), or even incorrectly-localized or formatted responses that may be entirely legitimate, but cannot be interpreted by the calling application.

Pinpointing errors in a loosely-coupled SOA application is a difficult and invasive process, often involving the equivalent of adding 'printf' debug statements to the code of individual components. By inspecting responses at the network, it becomes much easier to investigate and diagnose application problems and then work round them, either by retrying requests or transforming and rewriting responses as outlined in the previous example.



## Appendix 2: Programming the network using the ZXTM Control API

The following code examples illustrate some of the ways that the Control API can be used to automate reconfiguration or diagnosis tasks with ZXTM.

### 1. Adding a new server

Suppose that a new service node is deployed, increasing the capacity and redundancy of a particular SOA component. The ZXTM Service Delivery Controller must be informed so that it will load-balance requests for the relevant service to the new node.

Within the ZXTM configuration, application instances are referred to as 'nodes'. They are grouped into a 'pool' containing nodes with identical function. You may use the 'addNodes' Control API method to add a new node to a particular pool. The following code snippet illustrates how to do this using Java:

```
try {
    String poolname = NameOfPool;
    String[] newnodes = new String[] { NodeName1, NodeName2, ... };

    PoolLocator pl = new PoolLocator();
    pl.setPoolPortEndpointAddress(
        "https://username:password@hostname:9090" );
    PoolPort pp = pl.getPoolPort();

    pp.addNodes( new String[] { poolname }, new String[][] { newnodes } );
} catch (Exception e) {
    System.out.println( e.toString() );
}
```

### 2. Draining a Server

ZXTM can 'drain' connections from a node, waiting for existing connections to complete, and avoiding sending new connections to the node unless session persistence requirements dictate so. This is a vital step when preparing to take a node out of service, or when it is expected that it will be unavailable for more than a few seconds.

Use the 'addDrainingNodes' function to inform ZXTM to begin draining a node from a pool, and the 'removeDrainingNodes' function to stop a node from draining, so that it receives traffic normally.



The following code sample is a short Perl program that drains the named nodes from the provided pool:

```
#!/usr/bin/perl -w

use SOAP::Lite 0.60;

# This is the url and password of the ZXTM admin server
my $admin_server = 'https://username:password@hostname:9090';

# The pool to edit, and the nodes to drain
my $poolName = shift @ARGV;
my @theNodes = @ARGV;

# Create the SOAP Connection object
my $conn = SOAP::Lite
    -> ns('http://soap.zeus.com/zxtm/1.0/Pool/')
    -> proxy("$admin_server/soap")
    -> on_fault( sub {
        my( $conn, $res ) = @_;
        die ref $res?$res->faultstring:$conn->transport->status; } );

# Start the nodes draining
my $res = $conn->addDrainingNodes( [ $poolName ], [ @theNodes ] );
```

### 3. Retrieving Event Logs

If a remote application wishes to inspect the recent events that a ZXTM system has recorded, it can retrieve the event log using the 'getErrorLogString' function:

```
#!/usr/bin/perl -w

use SOAP::Lite 0.60;

# This is the url of the ZXTM admin server
my $admin_server = 'https://username:password@hostname:9090';

my $conn = SOAP::Lite
    -> ns('http://soap.zeus.com/zxtm/1.0/System/Log/')
    -> proxy("$admin_server/soap")
    -> on_fault( sub {
        my( $conn, $res ) = @_;
        die ref $res ? $res->faultstring : $conn->transport->status; } );

my $res = $conn->getErrorLogString();
print $res->result();
```

The 'Diagnose' interface can also be used to drill down on any problems. It gives a remote application full access to the state of each object that makes up the SDC configuration, reporting warning and error conditions in detail.



## Copyright

© Zeus Technology Limited 2007. Copyright in this document belongs to Zeus Technology Limited. All rights are reserved.

## Trademarks

Zeus Technology, the Zeus logo, Zeus Web Server, Zeus Load Balancer, Zeus Extensible Traffic Manager, ZXTM, ZXTM Global Load Balancer, ZXTM Virtual Desktop Broker and associated logos and abbreviations, TrafficScript, TrafficCluster and RuleBuilder are trademarks of Zeus Technology Limited. Other trademarks may be owned by third parties.

## Contact Information

If you would like to learn more about any of the topics covered by this white paper, please feel free to contact us for more information. You can reach us in a variety of ways:

### By Email

|   |  |
|---|--|
| For general enquiries:                      | <a href="mailto:info@zeus.com">info@zeus.com</a>         |
| For commercial and technical enquiries:     | <a href="mailto:sales@zeus.com">sales@zeus.com</a>       |
| For reseller information:                   | <a href="mailto:partners@zeus.com">partners@zeus.com</a> |
| For press and public relations information: | <a href="mailto:press@zeus.com">press@zeus.com</a>       |

### By Telephone

|                     |  |
|---------------------|--|
| Zeus Technology UK: | +44 1223 525000                          |
| Zeus Technology US: | 1-888-ZEUS-INC <i>or</i> +1 650 965 4627 |
| Fax:                | +44 1223 525100                          |

### By Post or in Person

|  |  |
|--|--|
| Zeus Technology Limited<br>The Jeffreys Building<br>Cowley Road<br>Cambridge CB4 0WS<br>United Kingdom | Zeus Technology<br>1955 Landings Drive<br>Mountain View<br>CA 94043<br>United States |
|--|--|

## [www.zeus.com](http://www.zeus.com)

Our web site contains a wealth of information on our products, services and solutions, as well as customer case studies and press information. For more information, please visit <http://www.zeus.com/>.

## [knowledgehub.zeus.com](http://knowledgehub.zeus.com)

The ZXTM KnowledgeHub is a key resource for developers and system administrators wishing to learn about ZXTM and Zeus' Traffic Management solutions. It is located at <http://knowledgehub.zeus.com/>.

