

.NET Enterprise Services


Prabhu Sunderaraman
prabhu@durasoftindia.com

at
Infosys
October 2004

DuraSoft

<http://www.durasoftindia.com>

Abstract

- Abstract : .NET Enterprise services provides a number of services like object pooling, events, queued components and distributed transactions. In this presentation we discuss the Enterprise services in general, and focus on the distributed transaction facilities. The objective of this presentation is to enable the attendees understand the benefits of Enterprise Services and provide them adequate information to start utilizing it in their applications.
- Targeted Audience : Software professionals involved in development using .NET technologies. Should be familiar with assemblies, DB components and Application Domains.
- Examples : Each page with a  has an Example.

.NET Enterprise Services

- Why and What?
- Object Pooling
- Just-in time activation
- Transactions
- Queued Components
- Controlling Object creation
- v 1.1 enhancements
- Conclusion

Before we start!

- Things to know
- AppDomains creates an isolation within a .NET process.
- Part of an assembly's identity is its' name, culture, version and strong name
- In addition to the keywords, certain capabilities of a class may be extended using attributes
- For COM accessibility, you need to register the Type library for the component.
- Microsoft provides Application Blocks to simplify access to SQL Server.

Why use Enterprise Services?

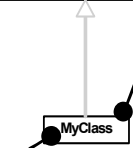
- Benefit from Infrastructural facilities
 - reduces your code size
 - easier to implement complex constructs
 - faster development
 - Scalability
 - more robust
- If you had a need to use COM+, you would need this as well
- OK, what is it?
 - Integration of COM+ services into .NET

Serviced Components

- A Serviced Component allows COM+ services to be available to .NET Framework classes
- Features supported
 - JIT Activation
 - Object Pooling
 - Queued Components
 - Transactions
 - (and other COM+ services)
- **System.EnterpriseServices** namespace provides facilities to access these

Writing a Serviced Component

System.EnterpriseServices.ServicedComponent



- Written in a CLS-Compliant Language
- Must have no-argument (default) constructor
- Declaratively configured using Attributes
- COM+ Catalog uses this information
- COM+ provides the context for execution
- Registered *dynamically* or manually

DuraSoft

DB-7

Dynamic vs. Manual Registration

- Dynamic registration
 - On first request from a managed client, a ServicedComponent object is registered automatically by the runtime
 - assembly not placed in GAC
 - Manual registration required if client is COM
- Manual registration
 - use regsvcs.exe to register
 - required for unmanaged clients to access
 - useful to perform design time tests

DuraSoft

DB-8

Activation Types

- **ActivationOption.Library**
 - ServicedComponent created in caller's process
- **ActivationOption.Server**
 - ServicedComponent created in a new process
 - Assembly must be in GAC
 - Types of parameters passed to methods must either be Serializable or inherit from MarshalByRefObject



.NET Enterprise Services

- Why and What?
- Object Pooling
- Just-in time activation
- Transactions
- Queued Components
- Controlling Object creation
- v 1.1 enhancements
- Conclusion

Object Pooling

- Allows reusing an object among method calls from different clients
- Useful if object creation overhead needs to be eliminated
- You can set the
 - minimum # of objects in pool
 - maximum # of objects in pool
 - creation timeout
- *Client must call DisposeObject()*
 - to signal that object may be returned to pool
- Should ActivationOption be Server or Library?

ActivationOption in Object Pooling

- Server
 - Out of process clients will use the objects that are created in the Default Domain
 - Seems obvious, isn't it?
- Library
 - This is a mess?!
 - Behavior depends on the OS
 - In XP and 2003, one pool **per application domain** is maintained

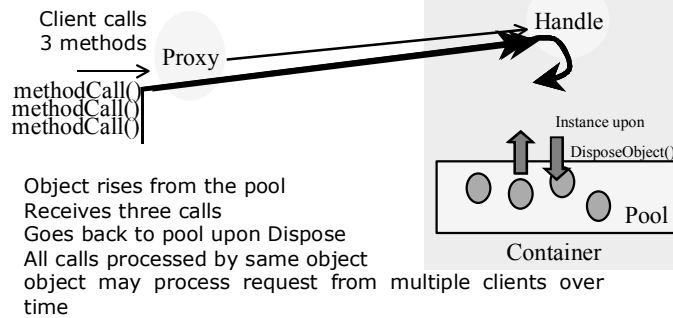
.NET Enterprise Services

- Why and What?
- Object Pooling
- Just-in time activation
- Transactions
- Queued Components
- Controlling Object creation
- v 1.1 enhancements
- Conclusion

Just-In-Time Activation

- Manages lifetime of object more efficiently
- Object is activated just prior to first call made on it and it is deactivated immediately when the object is done
- When is object deactivated?
 - When it is stateless - if the done bit is set then it will deactivate the object
- COM+ Allows the "Done" Bit to be Set Declaratively on a Method-by-Method Basis

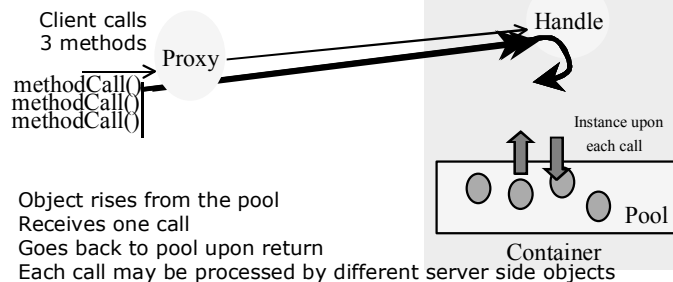
Instance Pooling vs. Swapping Pooling (Object Pooling)



DuraSoft

DB-15

Instance Pooling vs. Swapping Swapping (Object Pooling + JIT Activation)



DuraSoft

DB-16

JIT Activation

- While the client holds reference, the object is disposed
- The context is saved however
- Next call from client to object will be handled by another instance
- Similar to how session beans behave in EJB
 - instance swapping

JIT vs. Object Pooling

- If you intend to make one call only on the pooled object from each client, then combine JIT Activation with Object Pooling
- If you intend to make multiple calls on the object, use only object pooling

Quiz Time

- Which class should your component inherit from? `ServiceComponent`
- What are the two activation types? `Library`, `Server`
- What's the effect of library activation option in XP? One pool per AppDomain of client
- What's the difference between JIT and object pooling? Object deactivated right after a method call in JIT
- What methods should you override for object pooling? `Activate`, `Deactivate`, `CanBePooled`
- What method is the client required to call when done with the pooled object? `DisposeObject`
- What tool is used to manually register the component? `regsvcs.exe`

.NET Enterprise Services

- Why and What?
- Object Pooling
- Just-in time activation
- Transactions
- Queued Components
- Controlling Object creation
- v 1.1 enhancements
- Conclusion

Transaction Theory

- Transactions follow ACID rules
 - Atomicity - A transaction must either entirely succeed or entirely fail.
 - Consistent - A transaction cannot leave the system in an inconsistent state at the end of the transaction.
 - Isolation - No one outside the transaction can see the intermediate results during the transaction
 - Durable - When the transaction is complete, the data sources involved must guarantee that the updates will persist.

Transaction Integrity

- Set of related tasks that either succeed or fail as a unit - atomic
- Establishing and managing a transaction boundary may be challenging
- Requires close examination of code
- All related objects must participate in it
- Need to make sure all operations fall into that one transaction
- Find a timely place to commit or rollback
- Each participant needs to vote towards commit or rollback
- Managing all this may be tedious and error prone

Transaction-less Customer

- We have a customer who wants to buy milk
- He is interested in buying milk only if sugar is available
 - He wants to go home with both these items only
- Lets see how this works with .NET
 - Fails transaction integrity
- How to fix it?
 - You can create a transaction object and pass it around?!

DuraSoft



DB-23

Transaction Aware Customer

- Let's make Customer Transaction aware
- Make it a ServicedComponent
- Mark the class with
[Transaction(TransactionOption.Required)]
 - this makes state of component part of TXN
- Mark the methods with
- [AutoComplete]
 - this sets the vote to commit or abort (if an exception is thrown)
 - Alternately you may use ContextUtil.SetComplete() or SetAbort()

DuraSoft



DB-24

Problem with AutoComplete

- If no exception is thrown, ContextUtil.SetComplete() is called
- If exception is thrown, ContextUtil.SetAbort() is called
- Sounds good, what's your problem?
- Both these methods have a side effect
- Also sets ContextUtil.DeactivateOnReturn = true
- So, your component is gone upon return!

- What's the fix?
 - Do not use AutoComplete or SetComplete/SetAbort
 - set *ContextUtil.MyTransactionVote* to Abort on entry
 - If success, set **ContextUtil.MyTransactionVote to commit**

DuraSoft



Transaction Options

- Disabled
 - Is not transaction aware. Ignores TXN in current context
- NotSupported
 - Component is created in a context with no governing TXN
- Required
 - If caller is in a TXN, participates in it. Otherwise, creates a new one
- RequiresNew
 - Component created with its own TXN
- Supported
 - Participates in a TXN of caller if one exists

DuraSoft

DB-26

.NET Enterprise Services

- Why and What?
- Object Pooling
- Just-in time activation
- Transactions
- Queued Components
- Controlling Object creation
- v 1.1 enhancements
- Conclusion

Queued Components

- Queued Components (QC) is service provided by COM+ to hide the implementation details for MSMQ.
- Provides for loose coupling
- Asynchronous processing of messages



- Works
 - disconnected
 - when server is slow or has failed

.NET Enterprise Services

- Why and What?
- Object Pooling
- Just-in time activation
- Transactions
- Queued Components
- Controlling Object creation
- v 1.1 enhancements
- Conclusion

Controlling the use of Component

- In my original code, I had the constructor of Customer as protected internal
- This make it hard for others to create object of my class
- However, EnterpriseServices requires that I have a public default constructor!
- What if in the group of developers, some one tries to create an object of my class instead of using the factory?
- A simple trick comes in handy

Obsolete Attribute!

Mark constructor as obsolete

```
[Obsolete("Please use load method instead", true)]  
public Customer() {}
```

...

To create an object within my project

```
ctr = Activator.CreateInstance(typeof(Customer))  
as Collector;
```

...

If users of my class use new to create object, they get a compilation error

.NET Enterprise Services

- Why and What?
- Object Pooling
- Just-in time activation
- Transactions
- Queued Components
- Controlling Object creation
- v 1.1 enhancements
- Conclusion

Down side of Component Model

- .NET allows single implementation inheritance
- If you want your component to support Enterprise Services, you must inherit from `ServiceComponent`
- What if you want your component to inherit from another class?
- Transaction is simply an aspect (AOP)
- It is a cross cutting concern
- So, why not make it just that?
- This is where **ServiceDomain** class comes in

ServiceDomain

- Supported only on Windows 2003
- Has `Enter` and `Leave` methods
- Call to `Enter` creates a new context and pushes it onto Context Stack
- `Leave` emulates the behavior of returning from a call to another context
- More efficient, less overhead since no thread marshalling is involved
- `ServiceConfig` class is used to provide context specific information

Quiz Time

- How can you make sure no one uses this public default constructor? Mark it as obsolete
- How do you express the transaction needs of your component? [Transaction(TransactionOption.Required)]
- What attribute helps with method voting automatically based on success? AutoComplete
- What is the side-effect of using AutoComplete?
Deactivates the component
- What class allows you to voice your component's transaction vote? ContextUtil
- What method you may depend on to persist data?
Deactivate
- What classes eliminate the need for using ServiceComponent in 1.1? ServiceDomain, ServiceConfig

DuraSoft

DB-35

.NET Enterprise Services

- Why and What?
- Object Pooling
- Just-in time activation
- Transactions
- Queued Components
- Controlling Object creation
- v 1.1 enhancements
- Conclusion

DuraSoft

DB-36

Conclusion

- Enterprise services will help us
 - Develop faster
 - More robust and scalable
 - Lesser code to write and test
- Has some challenges
- Requires a bit more discipline in developing the components
- A direct access to COM+
- Consider it if you manage your own TXN or other facilities provided

References

- MSDN
- Presentation can be downloaded from <http://www.durasoftindia.com>
- *Please fill in your feedback forms*

Thanks