

# .NET Enterprise Services weisen den Weg

## Die Zukunft heißt Indigo

von Christian Nagel



.NET Enterprise Services, .NET Remoting oder Web Services mit ASP.NET? Teilweise gibt es bei diesen Kommunikationstechnologien überlappende Funktionalität, teilweise ergänzen sich diese Technologien prächtig. Vor allem benötigen diese Technologien aber unterschiedliche Programmier-techniken.

Mit den .NET Enterprise Services – ein Teil des Windows-Betriebssystems – können Services wie Object Pooling, verteilte Transaktionen, Queued Components usw. genutzt werden (siehe „Von COM+ zu Enterprise Services“ im Enterprise Programming).

So nützlich, umfangreich und leistungsfähig diese Dienste aber auch sind und so einfach sie heute mit .NET-Sprachen genutzt werden können, sie beherbergen dennoch einige Probleme:

- Komponenten müssen von der Basisklasse *ServicedComponent* ableiten. Damit ist es nicht möglich, eine andere Basisklasse mit eigener Funktionalität zu vererben. Hier ist es nur möglich, entweder eigene Funktionalität in eine Klasse, die von *ServicedComponent* ableitet, zu verpacken oder mittels Containment bestehende Funktionalitäten zu nutzen. Ähnlich ist es bei .NET Remoting, wo das entfernte Objekt von der Basisklasse *MarshalByRefObject* abgeleitet werden muss.
- Komponenten müssen, um COM+-Services nutzen zu können, in eine COM+-Applikation installiert werden. Eine Abhilfe dafür bieten *Services without Com-*

*ponents* („Services without Components“ im Enterprise Programming).

- Am Client muss ein Proxy installiert werden, um Enterprise Services eines Server nutzen zu können. Damit ist *Zero Touch Deployment* von Windows Applikationen nicht möglich. .NET Enterprise Services verwendet wie .NET Remoting ein Type-Sharing: Dieselbe Assembly muss am Client und am Server verfügbar sein.
- Die Wurzeln von COM+ in der COM-Welt sind bei vielen COM+-Services noch sehr gut sichtbar, z.B. akzeptiert der Queued Components Recorder am Client nur Parameter mit einfachen Typen (z.B. *int*, *string*) oder solche, die das COM-Interface *IPersistStream* implementieren.
- Contexts werden nur mit dem DCOM-Protokoll über das Netzwerk übertragen.
- *SOAP Services* verwenden .NET Remoting und damit das veraltete SOAP-Protokoll SOAP-RPC.

### Heutige Einschränkungen bei ASP.NET Web Services

Web Services, die mit ASP.NET programmiert werden, haben eine andere Programmier-technik im Vergleich zu .NET Enterprise Services. Mit ASP.NET können zwar auch Attribute verwendet werden, um Web Services und die generierte Beschreibungssprache WSDL zu definieren. Das Ableiten von einer Basisklasse ist bei ASP.NET aber nicht notwendig (das Verwenden der Basisklasse *System.Web.Services.WebService* ist optional).

Mit Web Services ist es möglich, Daten zwischen unterschiedlichen Plattformen auszutauschen, da hier das standardisierte

SOAP Protokoll verwendet wird. Aber SOAP und WSDL sind nicht nur vorteilhaft bei der plattformübergreifenden Kommunikation, sie sind auch nützlich, wenn die gleiche Plattform auf einem Client- und Server-System eingesetzt wird: WSDL ermöglicht die Kommunikation unterschiedlicher Versionen von Client und Server, da es die Entkopplung fördert (siehe „Contact-First Design und Entwicklung“ im EP).

Was bei Web Services allerdings fehlt, sind sicher Service-Funktionalitäten, wie sie von .NET Enterprise Services angeboten werden. Deshalb stehen .NET Enterprise Services sehr oft im Hintergrund eines Web Services. Vor allem fehlt es an Leistungsangeboten in folgenden Bereichen:

- **Security:** Plattformunabhängige Security von Web Services ist mit den so genannten *WS-Security*-Spezifikationen definiert. (Web Services-Spezifikationen beginnen meist mit dem Prefix *WS-*. Die Implementierung dieser Spezifikationen ist ein laufender Prozess, der für .NET z.B. mit den Web Services Enhancements (WSE) angeboten wird.
- **Performance:** Für einige verteilte Lösungen ist die Performance von SOAP und HTTP-Protokoll nicht ausreichend. Mit einem binären Datentransfer könnten Daten schneller übermittelt werden.
- **Reliability:** Mit HTTP können Nachrichten mehrmals übertragen und die Reihenfolge der Nachrichten kann während der Übertragung geändert werden.
- **Transaktionen:** Bei ASP.NET Web Services gibt es heute keinen standardisierten Weg, Transaktionen zu nutzen.

### kurz & bündig

Der Beitrag zeigt die Grenzen der heutigen .NET Enterprise Services und Web Services auf und zeigt, in welche Richtung sich die Kommunikationstechnologien für verteilte Anwendungen in Zukunft mit den Web Service Enhancements und „Indigo“ entwickeln werden.

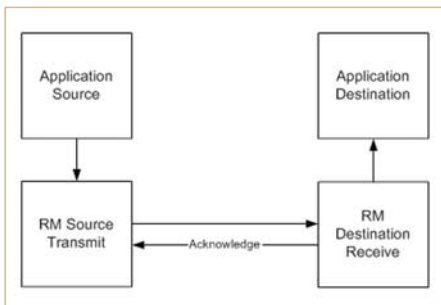


Abb. 1: Reliable Messaging

Einige Probleme von ASP.NET Web Services sind schon jetzt mit WSE 2.0 behoben – z.B. ist es möglich, SOAP mit WSE über einen TCP-Channel zu betreiben. Außerdem erlaubt WSE 2.0 das Betreiben von Custom Hosts und erfordert nicht mehr ASP.NET für das Hosting von Web Services (das SOAP/HTTP Protokoll ohne ASP.NET wird aber erst bei WSE 3.0 unterstützt). WSE macht die Programmierung von Web Services jedoch nicht einfacher und wird auch mit Visual Studio nicht mitgeliefert.

### Die Zukunft der Web Services

An den beschriebenen Problemen wird heute gearbeitet. Als Beispiele seien hier *WS-ReliableMessaging*, *WS-AtomicTransaction* und *WS-BusinessActivity* angeführt. *WS-ReliableMessaging* enthält Spezifikationen, um Qualität bei der Nachrichtenübermittlung zu garantieren:

- **Sequenz:** Für das Sortieren der Nachrichten am Zielsystem werden Sequenzen genutzt. Eine Sequenz kann aus mehreren Nachrichten bestehen. Mit jeder Nachricht wird ein eindeutiger Sequenz-Identifizierer übermittelt.
- **Nachrichtennummern:** Innerhalb einer Sequenz werden die Nachrichten nummeriert. Damit können die Nachrichten beim Zielsystem wieder geordnet werden.
- **Acknowledgements:** Als Bestätigung des Erhalts von Nachrichten werden Acknowledgements (ACK) zurückgeschickt, wobei es aus Performanzgründen nicht notwendig ist, ein ACK für jede Nachricht zu schicken; stattdessen kann ein ACK den erfolgreichen Empfang mehrerer Nachrichten beinhalten. Statt ACKs können auch negative Acknowledgements (NACK) zurückgeschickt werden – hier werden die Nachrichten, die nicht empfangen wurden, angeführt.

Abbildung 1 zeigt, dass der Reliable Messaging (RM) Mechanismus unabhängig von der Implementierung der Applikation ist. Um RM zu erzielen, wird die Nachricht von der Client-Applikation an RM Source Transmit übergeben, in der Sequenz und Nachrichtennummer bei der Nachricht hinzugefügt werden. RM Source Transmit wiederholt auch das Verschicken von Nachrichten, falls nicht alle Nachrichten am Zielsystem ankommen. RM Destination Receive kümmert sich um das Ordnen der Nachrichten sowie das Zurückschicken von ACK oder NACK.

Atomare Transaktionen können mit Transaktionen von Enterprise Services verglichen werden; dabei werden Ressourcen für die Dauer der Transaktion gesperrt. Atomare Transaktionen dürfen nicht über Geschäftsgrenzen geführt werden, da Locks nicht in die Hand von anderen Firmen gelangen sollen. Die Spezifikation *WS-AtomicTransaction* definiert atomare Transaktionen.

Bei Transaktionen muss deswegen auch zwischen atomaren Transaktionen und lang laufenden Transaktionen unterschieden werden. Die Basis für beide Transaktionsvarianten ist in der Spezifikation *WS-Coordination* gelegt, die beschreibt, wie mehrere Services miteinander zu koordinieren sind.

Abbildung 2 zeigt ein Beispiel, wie ein Coordinator die Participants A und B mit einer atomaren Transaktion koordiniert. Zuerst wird eine *Prepare* Nachricht an alle Web Services geschickt, die an der Transaktion teilnehmen. Kann ein Participant die Aktion der Transaktion erfolgreich vorbereiten, wird eine *Prepared* Nach-

richt zurückgeschickt. Im Beispiel schickt Participant B aber eine *Aborted* Nachricht zurück, da die Aktion bei ihm nicht erfolgreich ist. Der Coordinator schickt daher eine *Rollback* Nachricht an alle weiteren Teilnehmer (in diesem Fall Participant A), um auch sie aufzufordern, die Transaktion abubrechen.

*WS-BusinessActivity* ist die Spezifikation für lang laufende Transaktionen. Bei dieser Spezifikation werden Ressourcen nicht gesperrt. Wenn eine Aktion innerhalb einer Business Activity fehlschlägt, erfolgt eine Kompensation, die die vorhergehende Aktion rückgängig macht. Die Aktionen bei lang laufenden Transaktionen können auch Interaktionen vom Anwender beinhalten und Stunden oder Tage dauern.

Abbildung 3 zeigt Ihnen ein Beispiel der Nachrichtenübermittlung vom Coordinator mit den an der Activity teilnehmenden

### Listing 2

#### Definition der auszutauschenden Daten eines Indigo-Dienstes

```

[DataContract]
public class PersonInfo
{
    PersonInfo(int id, string firstname, string lastname)
    {
    }

    [DataMember] private string Firstname;
    [DataMember] private string Lastname;

    private int id;

    [DataMember(VersionAdded=2)]
    public string City;
}
  
```

### Listing 3

#### Transaktionalität eines Indigo-Dienstes einstellen

```

[ServiceContract(SecureChannel=true, SecurityMode=
    "Windows")]
[ServiceSettings(InstanceMode=InstanceMode.PerCall)]
public interface IDemoService
{
    [ServiceOperation(TransactionFlowAllowed=True)]
    public Message Foo(Message msg);
}

[TcpTransport]
public class DemoService : IDemoService
{
    public Message Foo(Message msg)
    {
        // Implementierung mit Nutzung einer automatischen
        // Transaktion
    }
}
  
```

### Listing 1

#### Definition der Operationen eines Indigo-Dienstes

```

[ServiceContract]
public interface ICalculator
{
    [ServiceOperation]
    int Add(int a, int b);
}

[Service]
public class Calculator : ICalculator
{
    public int Add(int a, int b)
    {
        return a + b;
    }
}
  
```

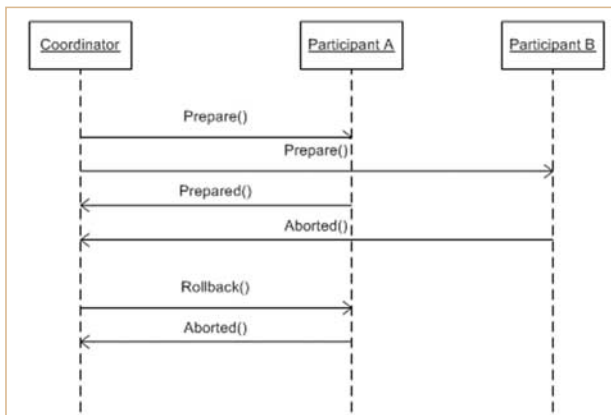


Abb. 2: Atomare Transaktionen

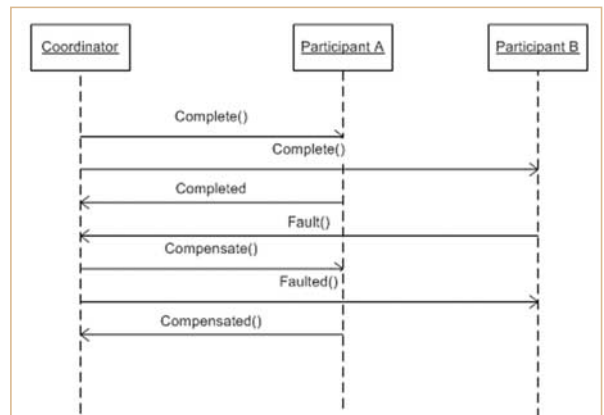


Abb. 3: Business-Aktivität

Web Services. Der Coordinator schickt Complete-Nachrichten an die Web Services, mit denen die Aktionen durchgeführt werden. Nachdem die Aktionen erfolgreich durchgeführt wurden, schicken die Participants die Nachricht *Completed* zum Coordinator. Ein Participant, der die Aktion nicht erfolgreich durchführen kann, antwortet mit *Fault*. Aufgrund der *Fault*-Nachricht eines Teilnehmers schickt der Compensator die *Compensate*-Nachricht an die anderen Teilnehmer. Aufgrund der *Compensate* Nachricht wird dann die Aktion – die bei dieser Art der Transaktion auch von Außenstehenden gesehen werden kann – wieder rückgängig gemacht.

Um Performanz bei Web Services zu gewinnen, ist es nicht mehr notwendig, SOAP über das HTTP Protokoll zu betreiben. Mit WSE 2.0 ist es jetzt schon möglich, Web Services über das TCP Protokoll zu nutzen. Mit *SOAP-over-UDP* können sogar Multicast-Nachrichten übermittelt werden, um damit mehrere Systeme mit einer einzigen Nachricht zu informieren.

### Die Zukunft verteilter Anwendungen: Indigo

Unter dem Codenamen *Indigo* arbeitet Microsoft derzeit an der Kombination der Vorteile von ASP.NET Web Services, WSE, .NET Remoting und auch noch .NET Enterprise Services unter Vermeidung ihrer Nachteile. Bei Indigo sind die vorgestellten Web Service Spezifikationen neben vielen weiteren implementiert und die Applikationen können diese Funktionalität direkt nutzen:

- **Hosting:** Indigo Services können in jedem Applikationstyp laufen. Vergleichbar mit .NET Remoting und WSE ist es möglich, Indigo Services nicht nur mit

ASP.NET Lösungen laufen zu lassen, sondern z.B. auch in einer Peer-to-Peer Lösung in Windows Forms Applikationen oder in NT-Services.

- **Attribute:** Wie bei ASP.NET Web Services und .NET Enterprise Services können Services und der Contract mit .NET-Attributen spezifiziert werden.
- **Kommunikations-Kanäle:** Vergleichbar mit .NET Remoting ist es bei Indigo möglich, Transport-Channels auszutauschen. Neben HTTP stehen unter anderem auch TCP, UDP sowie auch DCOM zur Verfügung.
- **Erweiterbarkeit:** Wie bei .NET Remoting ist es möglich, nicht nur eigene Channels, Formatters und Proxies zu programmieren, sondern auch Funktionalität bei den Methodenaufrufen am Client und am Server einzubauen. Im Gegensatz zu .NET Remoting kommen hier jedoch SOAP Headers zum Einsatz.
- **Unterstützung:** Heutige Technologien werden direkt unterstützt. Indigo erlaubt z.B. Hosting in einem COM+-Prozess, Kommunikation mit Enterprise Services und ASP.NET Web Services.

Listing 1 zeigt, wie ein Service mit Indigo implementiert werden kann. Mithilfe von Attributen wird gesteuert, welche Methoden und Interfaces den Contract (WSDL) definieren, das Attribut `[Service]` definiert die Service-Klasse. Beachtenswert ist, dass die Operationen eines Kontrakts als echtes Interface formuliert werden!

Die bevorzugte Variante, die auszutauschenden Daten mit Indigo zu spezifizieren, benutzt das `[DataContract]` Attribut wie in Listing 2 gezeigt. Während mit .NET Runtime Serialisierung (`[Serializable]` Attribut) alle privaten Daten serialisiert werden und bei der XML-Serialisierung alle

`public`-Felder und Properties, ist die Serialisierung bei Indigo ganz unabhängig von den Zugriffs-Modifizierern. Das Attribut `[DataMember]` kann bei `public` oder `private` Feldern oder Properties spezifiziert werden; außerdem ist ein default-Konstruktor nicht erforderlich.

Auch Transaktions-Services können wie bei .NET Enterprise Services mit Attributen spezifiziert werden. In Listing 3 werden mit dem Attribut `[ServiceOperation(TransactionFlowAllowed=True)]` die transaktionalen Eigenschaften eingestellt. Mit weiteren Attributen werden Security-Erfordernisse und Transport-Protokolle gesetzt.

### Zusammenfassung

Web Services haben eine große Zukunft, mit Microsoft Technologien heißt diese Zukunft Indigo. Es ist aber jetzt nicht notwendig auf Indigo zu warten. Um mit einer verteilten Lösung optimal auf Indigo vorbereitet zu sein, bieten sich jetzt ASP.NET Web Services und WSE als Facade zu .NET Enterprise Services an und *System.Messaging* für eine Kommunikation über das Netzwerk, bei der Client und Server nicht gleichzeitig verfügbar sein müssen. ●

### ● Links & Literatur

- [1] Christian Nagel: Enterprise Services mit dem Microsoft .NET Framework, Addison Wesley, erscheint im Frühjahr 2005
- [2] Christian Nagel et al.: Professional C#, 3rd Edition, Wrox Press
- [3] Web-Services-Spezifikationen: [msdn.microsoft.com/webservices/understanding/specs/default.aspx](http://msdn.microsoft.com/webservices/understanding/specs/default.aspx)
- [4] Übersichtsseite zum Thema „Indigo“: [msdn.microsoft.com/Longhorn/understanding/pillars/indigo](http://msdn.microsoft.com/Longhorn/understanding/pillars/indigo)