PLATAAN
Navision-partner voor advies en opleiding

# Dynamics-NAV Web Services Overview

'Technical Preview Session'

# Introductions

PLATAAN
Navision-partner voor advies en opleiding

Steven Renders
Microsoft Certified Trainer
Plataan
steven.renders@plataan.be

➔ Check Out: www.plataan.be

# Overview

PLATAAN
Navision-partner voor advies en opleiding

- Introduction
- What are Web Services?
  - How Does it Work?
  - Why Webservices ?
  - WebService Technology ?
  - What is SOAP?
  - What is WSDL?
  - What is UDDI?
- DevGuide.chm
- What is MSMQ ?
- Building a Dynamics-NAV Webservice.
- After-thoughts and conclusions.
- Dynamics-NAV 2009 !

# What are WebServices?

PLATAAN
Navision-partner voor advies en opleiding

- Web services are **application components**
- Web services communicate using **open protocols**
- Web services are **self-contained** and **self-describing**
- Web services can be **discovered** using **UDDI**
- Web services can be **used** by other **applications**
- **XML** is the basis for Web services

# How Does it Work?

PLATAAN
Navision-partner voor advies en opleiding

- The basic Web services platform is **XML** + **HTTP**.
- The **HTTP** protocol is the most used Internet protocol.
- **XML** provides a language which can be used between different platforms and programming languages and still express complex messages and functions.
- Web services platform elements
  - **SOAP** (Simple Object Access Protocol)
  - **UDDI** (Universal Description, Discovery and Integration)
  - **WSDL** (Web Services Description Language)

# Why Webservices ?

PLATAAN
Navision-partner voor advies en opleiding

- A few years ago Web services were **not fast** enough to be interesting.
  - Thanks to the major IT development the last few years, most people and companies have **broadband** connection and use the web **more and more**.
- **Interoperability** has highest priority.
  - When all major platforms could access the Web using Web browsers, different platforms could **interact**.
  - For these platforms to work together, **Web applications** were developed.
  - Web applications are **simple** applications run on the web.
  - These are built around the **Web browser** standards and can mostly be used by any browser on any platform.

# Why Webservices ?

PLATAAN
Navision-partner voor advies en opleiding

- **Web services** take Web applications to the **next level**.
  - Using Web services your application can **publish** its function(s) or message(s) to the **rest** of the **world**.
  - Web services uses **XML** to code and decode your **data** and **SOAP** to **transport** it using open protocols.
  - With Web services your accounting departments Win X servers billing system can connect with your IT suppliers Y server.

# Purpose of WebServices ?

PLATAAN
Navision-partner voor advies en opleiding

- Web services have **two** types of **uses**.
- **Reusable** application **components**
  - There are things different applications need very often. So why make these over and over again?
  - Web services can offer **application components** like currency conversion, weather reports or even language translation as **services**.
  - Ideally, there will only be **one type** of each application component, and **anyone can use it** in their application.
- **Connect** existing **software**
  - Web services help solve the **interoperability** problem by giving different applications a way to **link** their data.
  - Using Web services you can **exchange data** between different **applications** and different **platforms**.

# WebService Technology ?

PLATAAN
Navision-partner voor advies en opleiding

- Web Services have **three** basic platform **elements**.
- These are called **SOAP**, **WSDL** and **UDDI**.

# What is SOAP?

PLATAAN
Navision-partner voor advies en opleiding

- The basic Web services platform is **XML** plus **HTTP**.
  - SOAP stands for **S**imple **O**bject **A**ccess **P**rotocol
  - SOAP is a **communication** protocol
  - SOAP is for communication between **applications**
  - SOAP is a **format** for sending messages
  - SOAP is designed to **communicate** via Internet
  - SOAP is **platform independent**
  - SOAP is **language independent**
  - SOAP is based on **XML**
  - SOAP is **simple** and **extensible**
  - SOAP allows you to get around **firewalls**
  - SOAP will be developed as a **W3C standard**

# What is WSDL?

PLATAAN
Navision-partner voor advies en opleiding

- WSDL is an XML-based language for **describing** Web services and how to **access** them.
  - WSDL stands for **W**eb **S**ervices **D**escription **L**anguage
  - WSDL is written in **XML**
  - WSDL is an XML **document**
  - WSDL is used to **describe** Web services
  - WSDL is also used to **locate** Web services
  - WSDL is not yet a W3C standard

# What is UDDI?

PLATAAN
Navision-partner voor advies en opleiding

- UDDI is a **directory service** where businesses can **register** and **search** for Web services.
  - UDDI stands for **U**niversal **D**escription, **D**iscovery and **I**ntegration
  - UDDI is a **directory** for storing **information** about web services
  - UDDI is a **directory** of web service **interfaces** described by WSDL
  - UDDI communicates via **SOAP**
  - UDDI is **built into** the Microsoft **.NET** platform

# DevGuide.chm

PLATAAN
Navision-partner voor advies en opleiding

- Dynamics-NAV Install DVD:
  - NAV50BE\Nas\Common\Dynamics NAV**Communication Component**
- This documentation describes a **set** of **components** that allow applications to **communicate** easily, both synchronously and asynchronously, with each other.
- You can **extend** this system to fit your own particular needs.
- The components consist of:
  - the Navision Named Pipe Bus Adapter.
  - the Navision **MS-Message Queue Bus Adapter**.
  - the Navision Socket Bus Adapter.
  - the Navision Communication Component version 2.0.

# Navision MS-Message Queue Bus Adapter

PLATAAN
Navision-partner voor advies en opleiding

- Provides flexible, heavyweight synchronous / asynchronous **communication** between two systems.
- Supports disconnected communication and prioritized messages.
- The **bus adapter** supports 'all' versions of the **M**icrosoft **M**essage **Q**ueue **S**erver (MSMQ).

# Definition of Message Queuing (MSMQ)

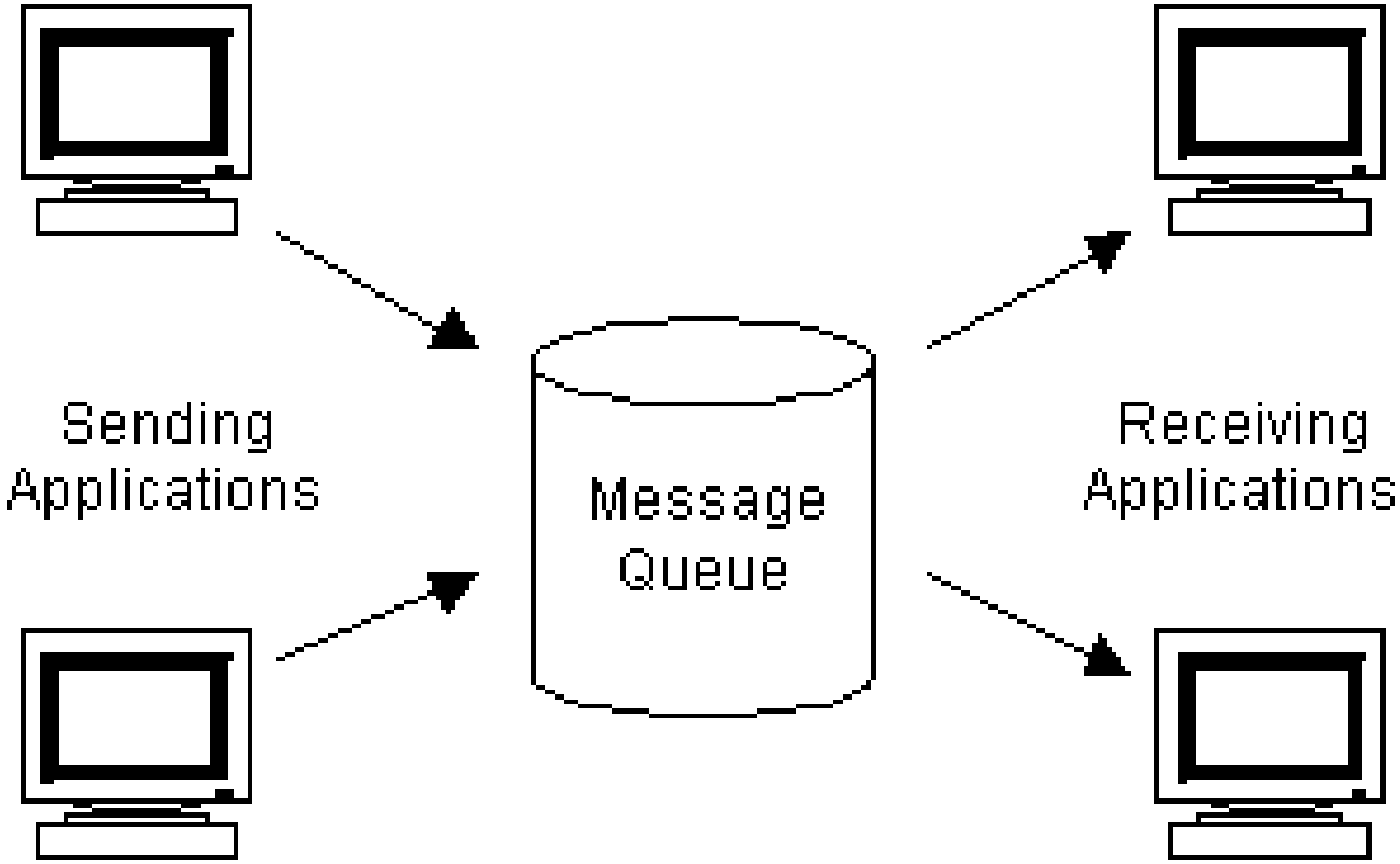
PLATAAN
Navision-partner voor advies en opleiding

- Microsoft provides the following definition of Message Queuing:

- **"Message Queuing (MSMQ) technology enables applications running at different times to communicate across heterogeneous networks and systems that may be temporarily off-line”.**

- Applications **send** messages to **queues** and **read** messages from **queues**.
- The following **illustration** shows how a queue can hold the messages used by both sending and receiving applications.

# MSMQ


Sending
Applications
Message
Queue
Receiving
Applications

# Sending a Document Code Example

PLATAAN
Navision-partner voor advies en opleiding

- In this simple example, we use a **codeunit** to **initialize** the Navision Communication Component, **establish contact** to the Navision MS-Message Queue Bus Adapter and **send** a document.
- This example does not include **error handling**.
- For the purpose of this example, we have defined the following **variables**:

| Variable Name | Data Type | Subtype |
|---|---|---|
| MQBus | Automation | 'Navision MS-Message Queue Bus Adapter'.MSMQBusAdapter |
| CC2 | Automation | 'Navision Communication Component version 2'.CommunicationComponent |
| OutMsg | Automation | 'Navision Communication Component version 2'.OutMessage |
| OutS | OutStream | |

# Sending a Document Code Example

PLATAAN
Navision-partner voor advies en opleiding

```
OnRun()
CREATE(CC2);
CREATE(MQBus);
CC2.AddBusAdapter(MQBus,1);
MQBus.OpenWriteQueue('MyMessageQueueServer\XML-
   requests',0,0);
MQBus.SenderAuthenticationLevel:= 2;
OutMsg := CC2.CreateoutMessage('Message
   queue://MyMessageQueueServer\XML-requests');
OutS := OutMsg.GetStream();
OutS.WRITE('Hello world!');
OutMsg.Send(0);
```

# Receiving a Document Code Example

PLATAAN
Navision-partner voor advies en opleiding

- In this example, we use a **single instance** codeunit to **initialize** the Navision Communication Component, establish **contact** to a Navision MS-Message Queue Bus Adapter, **open** the bus adapter's **receive queue** and, finally, **read** the **message** that is received.
- For the purpose of this example, we have defined the following variables:

| Variable Name | Data Type | Subtype | Length |
|---|---|---|---|
| MQBus | Automation | 'Navision MS-Message Queue Bus Adapter '.MSMQBusAdapter | |
| CC2 | Automation | 'Navision Communication Component version 2'.CommunicationComponent | |
| InMsg | Automation | 'Navision Communication Component version 2'.InMessage | |
| InS | InStream | | |
| Txt | Text | | 100 |

# Receiving a Document Code Example

PLATAAN
Navision-partner voor advies en opleiding

```
OnRun()
CREATE(MQBus);
CREATE(CC2);
CC2.AddBusAdapter(MQBus,1);
MQBus.OpenReceiveQueue('MyMessageQueueServer\co
   mcom2_queue',0,0);
CC2::MessageReceived(VAR InMessage : Automation)
InMsg := InMessage;
InS := InMsg.GetStream();
InS.READ(Txt);
MESSAGE(Txt);
InMsg.CommitMessage();
```

# IMSMQBusAdapter Interface

PLATAAN
Navision-partner voor advies en opleiding

| Method Name | Description |
|---|---|
| OpenWriteQueue | Opens a queue for sending messages. |
| GetWriteIMSMQQueue | Returns the MSMQQueue object that is used to write to allow direct manipulation. |
| OpenReceiveQueue | Opens a queue for receiving messages. |
| GetReceiveIMSMQQueue | Returns the MSMQQueue object that is used to receive to allow direct manipulation. |
| CreateQueue | Produces a queue that can send and receive messages. |
| OpenReplyQueue | Opens a queue to receive reply messages (when you use SendWaitForReply). |

| Property Name | Description |
|---|---|
| SenderCertificate | Provides an array of bytes that represent the security certificate. |
| AdministrationQueue | Provides the name of the AdministrationQueue in case this can be created or used. |
| SenderAuthenticationLevel | Provides an authentication level that is passed to each message, which is sent through this bus adapter. |
| ReceiverAuthenticationLevel | Requests an authentication level for each incoming message. |
| WriteTimeout | Provides a maximum time in seconds for an application to read the messages sent by this bus adapter. |
| ReceiveTimeout | [In, Out] Provides a maximum time in milliseconds for the bus adapter to receive a message. |
| RemoveWhenCommit | [In, Out] Sets the bus adapter to either remove or not to remove the message from the Journal Queue when an InMessage is committed. |

# Building a Dynamics-NAV Webservice

PLATAAN
Navision-partner voor advies en opleiding

- Now, it is time to engineer on this and use this simple mechanism to **wrap our ERP system into a web service** which **exposes** Dynamics-NAV **business layer** to 3rd party applications.
- If you understand the communication model and if you have been developing your own web services for a while, it should be **quite easy** how to add a web service layer around Dynamics-NAV.
- Our main concern is on **how** to take each **method's arguments** and **repack** them in a way that they are **understood** when the request is forwarded to **Dynamics-NAV**.

# The scenario & architecture

PLATAAN
Navision-partner voor advies en opleiding

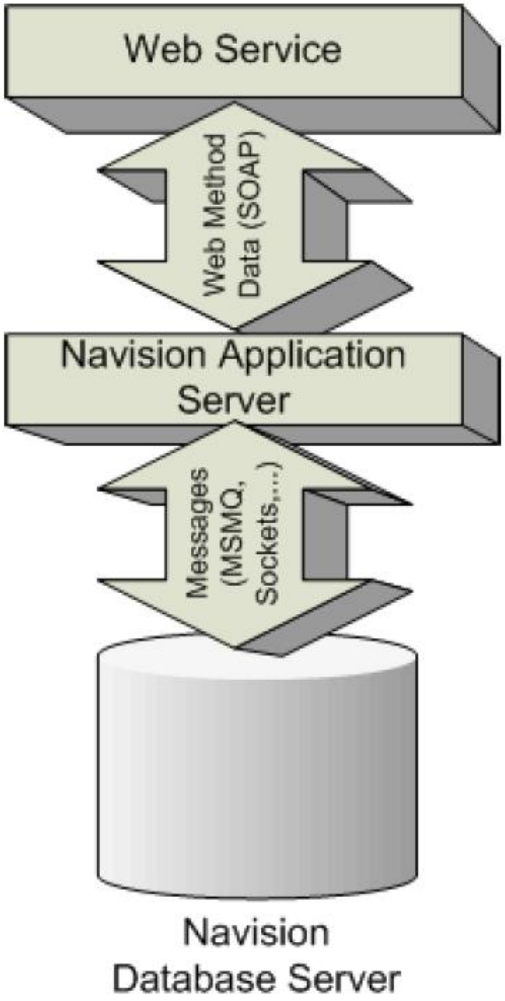
- Suppose you are developing a **warehouse management tool** which the warehouse employees will be able to access from their handheld devices or from any web browser.
- When designing such application, you clearly see a very tight **interaction** between your application and Dynamics-NAV, which centralizes the customer's warehouse items and surrounding information.
- By providing such **connection** between both systems, you can be sure your application will always use the **latest information** as it enters **your ERP system**.

# The scenario & architecture

PLATAAN
Navision-partner voor advies en opleiding

- However, when using a system like Dynamics-NAV, such connection means using Navision **Application Server**, a **communication subsystem** such as named pipes, sockets or **message queuing** and a **protocol** both systems must be aware of for communication purposes.
- From the developer's point of view, it would be a much easier task if all of the requirements above are "**hidden**" behind some sort of an **A**pplication **P**rogramming **I**nterface (API).

# Proposed layered architecture



# The scenario & architecture

PLATAAN
Navision-partner voor advies en opleiding

- We will start by **choosing** the set of Dynamics-NAV **functions** we would like to **expose** and then discuss **how** they **map** onto the architecture.
- At least **two** functions must be exposed, one that allows for data **reading** and another one which performs data **writing**.
  - **GetItem** (Item no.) which returns an item instance
  - **InsertItem** (Item no., Description, ...) which inserts a new item based on a set of field values
- Each of these functions/procedures will have a **counterpart** both within **Dynamics-NAV** Database Server and at the highest level, being exposed as **Web Methods**.

# The scenario & architecture

PLATAAN
Navision-partner voor advies en opleiding

- When requesting an **item** find, the Web Method will get the **argument** (the item number) and will **forward** a message to Dynamics-NAV.
- Dynamics-NAV will then **read** the message and **route** it to the appropriate handler within its **business layer**.
- The **response**, if available, is **sent** back as a message and the **Web Method** must be able to **transform** it according to the external representation.
  - Although they might look similar, the data format used by Dynamics-NAV and externally are completely independent of each other.
  - Their design may be similar and that will ease the translation process, but there is no formal reason for doing so.

# From the web service down to the business layer

PLATAAN
Navision-partner voor advies en opleiding

- When a **Web Method** is **called**, along with it goes a set of **arguments**.
- All of this information must **reach** the **business layer** which will then **respond** accordingly.
- For this purpose, instead of using a complex XML structure, we have chosen to **pack both the method name and the arguments into a string**, pretty much as if it were a local call:
  - <Method Call> ::= <Method Name> ( <Argument>* )
- All that Dynamics-NAV has to do is **parsing** that string, retrieving both the method **name** and the **arguments**.
- Later you will see how this parsing may be performed.

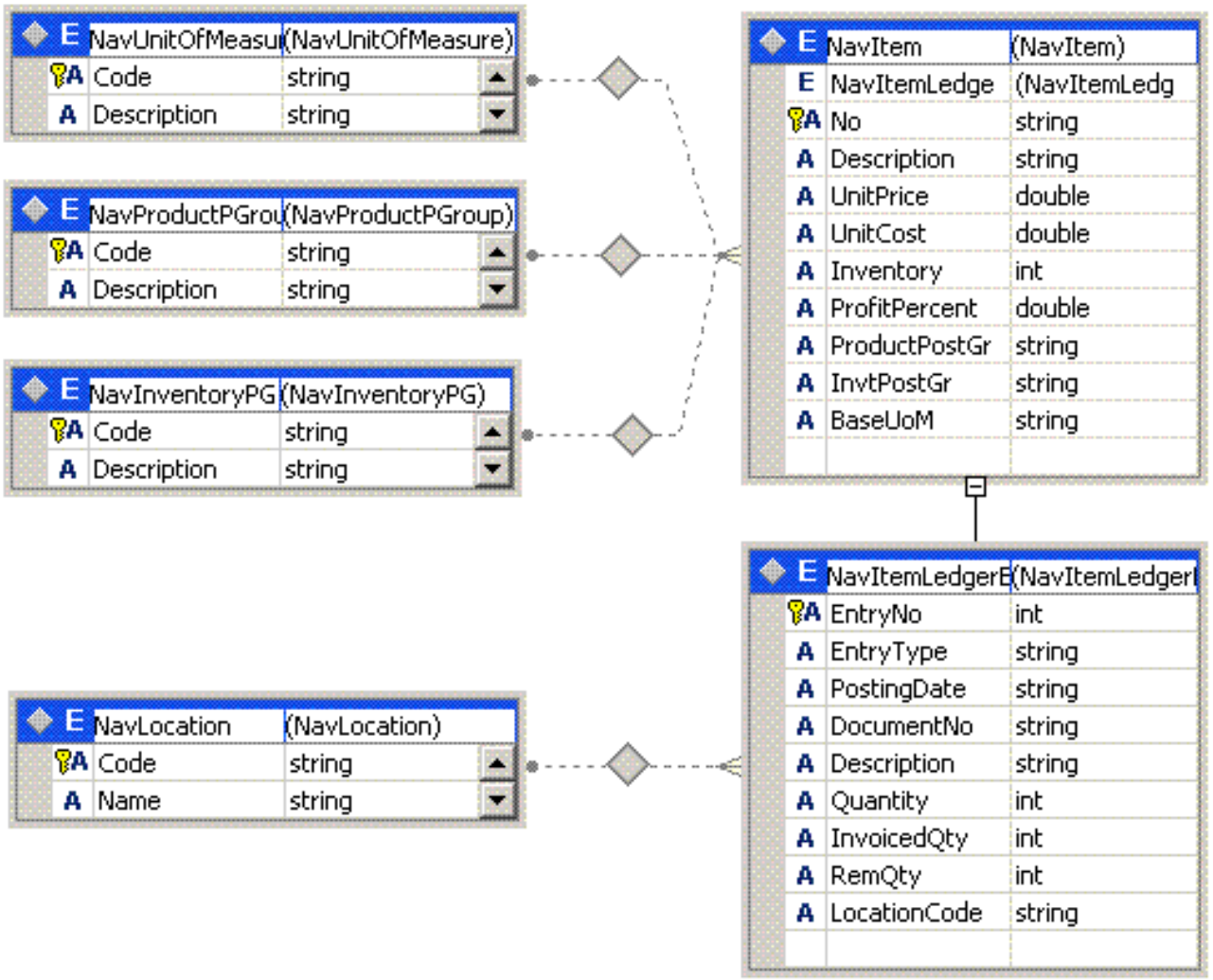
# From Navision back up to the web service

PLATAAN
Navision-partner voor advies en opleiding

- After the business layer has processed the request, it has to **bubble** the data back up to the calling **Web Method**, which will then **return** it.
- Now we must rely upon a somewhat **complex structure** which may **hold** all the bubbling **data**.
- We have decided to include some **complementary data** along with our inventory items:
  - general product and inventory posting groups, locations and units of measure.

# The schema both layers must respect.

PLATAAN
Navision-partner voor advies en opleiding



# The schema both layers must respect.

PLATAAN
Navision-partner voor advies en opleiding

- For now, let us just assume that our core entity is the item.
- An **item** is identified by its **number** and it contains zero or more item **ledger** entries.
- Also, 3 of the item's properties are validated against 3 tables: its **base unit of measure** and its **product** and **inventory posting groups**.
- Finally, each ledger entry has a **location** code which is validated against the location collection.
- This schema maps onto the Dynamics-NAV **object model**.
- (a very small subset of the object model).

# Setting up the environment

PLATAAN
Navision-partner voor advies en opleiding

- Ensure your Dynamics-NAV database is being served by either a Navision **Database Server** or a Microsoft **SQL Server**;
- Configure a Navision **Application Server**, making sure it points to a specific **start-up parameter** value and a specific **company** within a specific database located on a specific server.
- Create **two message queues** which will be used to support the bidirectional communication.
- Add a new **codeunit** which will **receive** the messages, **parse** the requests and **respond** to them (marked as **single instance**).
- Edit trigger **NasHandler** () on Codeunit 1, adding a new case option for the Parameter.

# NASHandler trigger

- In our development environment, we have used a **WEBSERVICE** start-up parameter value and we have created a new codeunit named Web Service Handler.
- We have added the following code to the NasHandler trigger in codeunit 1:

```
IF CGNASStartedinLoop = FALSE THEN
CASE Parameter OF
'WEBSERVICE':
   WSHandler.RUN;
'MAILLOG':
   CODEUNIT.RUN(CODEUNIT::"E-Mail Dispatcher");
ELSE
   ...
```

# Visual Studio .NET

PLATAAN
Navision-partner voor advies en opleiding

- Additionally, we have Microsoft Visual Studio .NET and all its requirements for the development of ASP.NET web services.

- Check out the Web Services Developer Center
  - http://msdn.microsoft.com/webservices

# The first Web Method: GetItem

PLATAAN
Navision-partner voor advies en opleiding

- In order to present the first Web Method, we will split the whole process in 4 parts:
  - Packing the **method** name and its single **argument**, the item number and sending the string to Dynamics-NAV.
  - Having Dynamics-NAV **receiving** the string, **unpacking** it and **redirecting** to the appropriate code which will serve the request.
  - **Serving** the request, which means **finding** the **item** whose number is the one received as argument, packing the data into an **XML document** conformant to the schema and **sending** it **back**.
  - **Receiving** the **response**, **validating** it against the schema, building the **dataset** and **returning** it to the caller system.

# Packing and sending the request to Navision

PLATAAN
Navision-partner voor advies en opleiding

- Once the Web Service has been created, we need to bring in the two **message queues** we created for this project.
- By naming them **mqFromNavision** and **mqToNavision** it should be easier to understand when to use each.
- Now we are ready to create the first **web method**, as follows:

```
[Web Method]
public NavDS GetItem(string No)
{
    string request = "GetItem(" + No + ")";
    mqToNavision.Send (request, "Navision MSMQ-BA");
    ...
}
```

# Having Navision parsing and redirecting the request accordingly

PLATAAN
Navision-partner voor advies en opleiding

- By adding a **handler codeunit**, Dynamics-NAV is prepared to receive the request.
- However, we still need to add a **procedure** which will **parse** the request, **retrieving** both the **method name** and the **argument** collection.
- Let us assume our codeunit has two global variables:
- **Request—Text** (50)
- **Parameters—Text** (50) with the Dimensions property set to a number that may hold the largest amount of arguments (200 is much more than enough)
  - This procedure starts by retrieving the method name from the string cutting it by the open parenthesis.
  - Then, it loops through the comma-separated argument collection and builds the Parameters array with those values.

# Having Navision parsing and redirecting the request accordingly

PLATAAN
Navision-partner voor advies en opleiding

```
ParseRequest(string : Text[250])
Request := COPYSTR (string, 1, STRPOS (string, '(') - 1);
auxstring := COPYSTR (string, STRPOS (string, '(') + 1, STRLEN
                    (string) - STRPOS (string, '(') -1);
argpos := 1;
commapos := STRPOS (auxstring, ',');
WHILE (commapos <> 0) DO
BEGIN
    Parameters[argpos] := COPYSTR (auxstring, 1, commapos - 1);
    auxstring := COPYSTR (auxstring, STRPOS (auxstring, ',') + 1);
    argpos := argpos + 1;
    commapos := STRPOS (auxstring, ',');
END;
Parameters[argpos] := auxstring;
ParCount := argpos;
```

# Having Navision parsing and redirecting the request accordingly

PLATAAN
Navision-partner voor advies en opleiding

- Now that we have a **request parser**, we are able to fill the code of the message received trigger like this:

```
CC2::MessageReceived(VAR InMessage : Automation "".IDISPATCH")
// load the message into an XML document and find the string node
InMsg := InMessage;
InS := InMsg.GetStream();
XMLDom.load (InS);
XMLNode := XMLDom.selectSingleNode ('string');
// parse the request and according to the Request variable, redirect to
// the appropriate function
ParseRequest (XMLNode.text);
CASE Request OF
'GetItem':
    BizLayer.GetItem (Parameters[1], XMLDom);
ELSE
END;
```

# Serving the request and responding

PLATAAN
Navision-partner voor advies en opleiding

- In the message received trigger, we **redirect** the request to the **GetItem procedure** on a **new codeunit** which represents the **business layer** entry point.
- In order to respond to the web service methods, we have chosen to create this **codeunit** which will hold the Dynamics-NAV **counterpart** for each method.

# Auxiliary procedures that help us building the XML.

PLATAAN
Navision-partner voor advies en opleiding

```
AddElement(VAR XMLNode : Automation "'Microsoft XML,
    v3.0'.DOMDocument";NodeName : Text[250];VAR CreatedXMLNode :
    Automation "'Microsoftt XML, v3.0'.IXMLDOMNode")
NewChildNode := XMLNode.ownerDocument.createNode('element',
    NodeName, '');
XMLNode.appendChild(NewChildNode);
CreatedXMLNode := NewChildNode;

AddAttribute(VAR XMLNode : Automation "'Microsoft XML,
    v3.0'.IXMLDOMNode";Name : Text[260];NodeValue : Text[260])
IF NodeValue <> '' THEN BEGIN
    XMLNewAttributeNode :=
    XMLNode.ownerDocument.createAttribute(Name);
    XMLNewAttributeNode.nodeValue := NodeValue;
    XMLNode.attributes.setNamedItem(XMLNewAttributeNode);
END;
```

# Navision counterpart for GetItem

PLATAAN
Navision-partner voor advies en opleiding

- Now, getting back to the Dynamics-NAV counterpart for the **GetItem** specific situation, this could be the code to add:

- ➔ See Demo. (GetItem)

# Navision counterpart for GetItem

PLATAAN
Navision-partner voor advies en opleiding

- By passing a reference to **XMLDom**, when this procedure ends, that variable will hold a populated **dataset** with all the locations, all the product and inventory posting groups, all the units of measure, items and their item ledger entries.
- We have decided to include all of the locations, product groups and units of measure as they represent a small amount of data and will not overburden our dataset instances.

# message received trigger

- Now that the **second argument** of the function has been filled with an **XML document** representing the **dataset** we would like to return, we have to **complete** the **message** received trigger by adding the **code** which will **instantiate** a message that will be **sent** up to the **web service**:
- ➔ See Demo (CC2::MessageReceived)

# Receiving and validating the response and returning the dataset.

PLATAAN
Navision-partner voor advies en opleiding

- The request has been **sent** to Dynamics-NAV, Dynamics-NAV **understood** it, **handled** it, **responded** to it and now a **dataset** is on a **queue** waiting to be **received**, **validated** and then **returned** to the web method caller.
- Let us add the **code** that will do exactly what is missing in this scenario: (see demo)

```
[Web Method]
public NavDS GetItem(string No)
{
string request = "GetItem(" + No + ")";
mqToNavision.Send (request, "Navision MSMQ-BA");
mqFromNavision.Formatter = new
System.Messaging.XmlMessageFormatter (new Type[] {typeof (NavDS)});
System.Messaging.Message msg =
mqFromNavision.Receive (new System.TimeSpan (0,0,0,30));
NavDS nds = new NavDS ();
nds.ReadXml (msg.BodyStream, System.Data.XmlReadMode.Auto);
return nds;
}
```

# NavDS ?

PLATAAN
Navision-partner voor advies en opleiding

- What is this **NavDS** we are seeing in the code?
- We are creating an object of this type and using its **ReadXML** () method to read the contents of the message.
- Well, it is a **dataset** whose schema has already been depicted in figure 2 above.
- Here goes its formal XML representation:
- ➔ see NavDS.ds

# The WebService in Action

PLATAAN
Navision-partner voor advies en opleiding

- DemoTime !

# After-thoughts and conclusions

PLATAAN
Navision-partner voor advies en opleiding

- As you could see, the **magic** of communication has already been presented.
- This demo allows you - as a software architect and/or developer – to conceal the specifics of such communication, thus providing those responsible for the upper layers of your applications with a simple interface with the ERP system.

# After-thoughts and conclusions

PLATAAN
Navision-partner voor advies en opleiding

- **Kind** of **environments** in which this solution may participate:
- A **complex** system **composed** of **multiple applications** aimed at distinct goals, in which there is a **central entity** responsible for the overall **synchronization**.
  - This entity would probably need to be able to recurrently **read data from** the **ERP** system and to **post changed** or **new** data onto it as well.
  - By being able to call a set of **web methods**, this task would be much **easier** than otherwise having to go all the way to and from Navision;

# After-thoughts and conclusions

PLATAAN
Navision-partner voor advies en opleiding

- A **business-specific** application whose processes merely **intersect** the scope of **Navision**, specially in its later stages when dealing with the company resources, such as customers, general ledger accounts or warehouse picks.
  - When developing these connections, having a Web Service which takes care of these details seems a good resource;
- An **internal web portal** for warehouse employees in which they can check the inventory and perform item adjustments, for instance.
- ...

# What's Next?

PLATAAN
Navision-partner voor advies en opleiding

- Now You Know Web Services, What's Next?
- The next step is to learn about WSDL and SOAP.
- **WSDL**
  - WSDL is an XML-based language for **describing** Web services and how to **access** them.
  - WSDL describes a web service, along with the message format and protocol details for the web service.
- **SOAP**
  - SOAP is a simple XML-based protocol that allows applications to **exchange** information over HTTP.
  - Or more simply: SOAP is a protocol for **accessing** a web service.

# Dynamics-NAV 2009

PLATAAN
Navision-partner voor advies en opleiding

- How will Webservices work in the next Dynamics-NAV version ?

- ➔ Demo Kurt Juvijns.

# Resources available

PLATAAN
Navision-partner voor advies en opleiding

- www.msdn.microsoft.com/webservices
- DevGuide.chm (NAS)
- Microsoft .Net Framework SDK QuickStart Tutorials (http://samples.gotdotnet.com/quickstart/)
- Talking with Navision: Accessing Navision Business Layer through a Web Service (http://msdn2.microsoft.com/en-us/library/ms952079.aspx)
- Talking with Navision: Say Hello to Navision and Expect Navision to Be Polite (Talking with Navision: Say Hello to Navision and Expect Navision to Be Polite)
- Exposing .NET Components to Navision (http://msdn2.microsoft.com/en-us/library/aa973247(printer).aspx?topic=306132)
- www.plataan.be ➔ Training & Blog
- …

# The End...

- Thank you for attendance and participation.


Steven Renders
Microsoft Certified Trainer
Plataan
steven.renders@plataan.be