

# SOA with kbmMW



**Your guide on this tour:**

**Kim Madsen/CEO  
Components4Developers**



# Agenda



**What is SOA?**  
**What is kbmMW?**

Zzzzzzzz zzz z ... Grmph.... zzzz

# What is SOA?

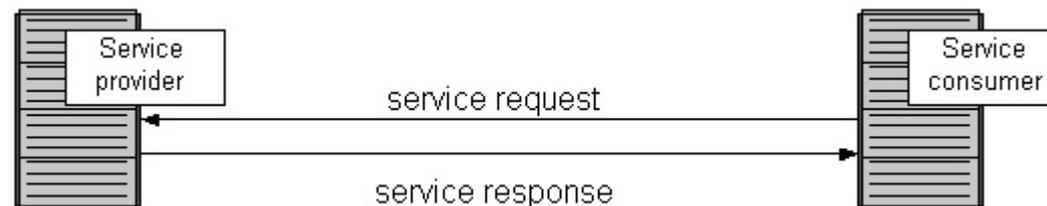


## SOA : Service Oriented Architecture

**Service** = A well defined, self contained blackbox of functionality that is provided to the 'world' and that does not depend on the context or state of other services.

A number of such services essentially constitute a SOA.

**OLD concept... NEW packing!**



# What is SOA? (cont.)



## Textbook example of SOA

**A CD player** – Its a device with a specific functionality. It enables you to play CD's (your data) without the requirement of other services. You can even switch out the CD player with another and still play your CD.

If you have many CD's you need to play, you can have multiple CD players operating at the same time and put any CD in any CD player... **Loadbalancing!**

Each CD player provide the same service, but the quality of the outcome may be different. Thats called quality of service (**QoS**)



# What is SOA? (cont.)



## SOA implementations

examples of...

**DCOM**

**Corba**

**SOAP – WebServices**

**RPC (Remote Procedure Calls)**

# What is SOA? (cont.)



## Benefits of SOA?

**Reusability** - The same CD player can be used for many different music CD's.

**Failover capabilities** - If one CD player stops working, plug in another one.

**Loadbalancing** – Got many CD's to play? Plugin some more CD players.

**Black box** – You dont have to care whats going on inside to use its functions.

# What is kbmMW?



## An advanced, flexible and full featured n-tier application server based framework

What is n-tier?

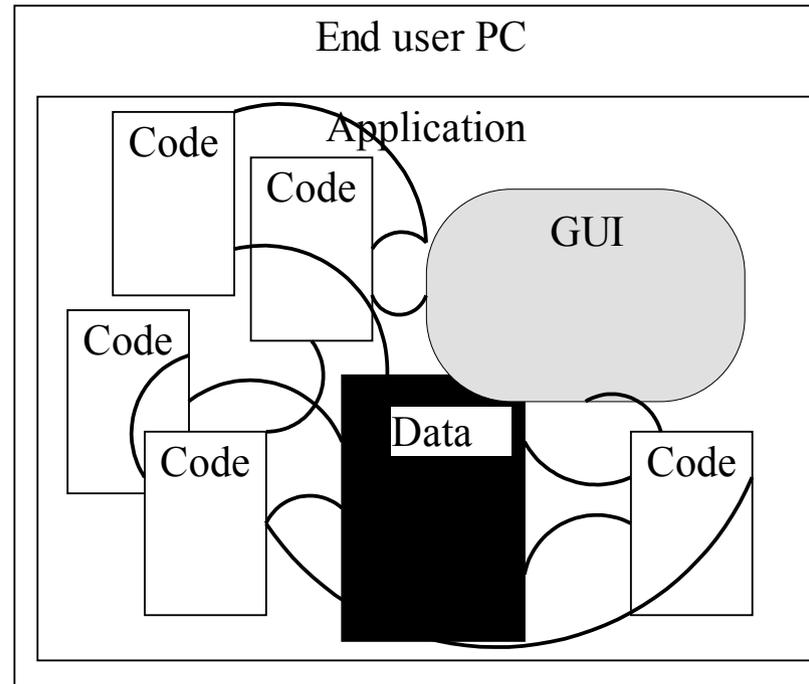
- 1-tier: Enduser application doesnt require any other applications to operate
- 2-tier: Enduser application require another application, typically on another machine to operate.  
Typically a database (Client/Server)
- 3-tier: Enduser application require two other applications running to operate.  
Typically an application server and a database in sequence.

More tiers are not uncommon.

# What is kbmMW? (cont.)



## 1-tier



# What is kbmMW? (cont.)



## 1-tier (cont.)

### Advantages

- Easy to install
- Only limited by speed of the users computer

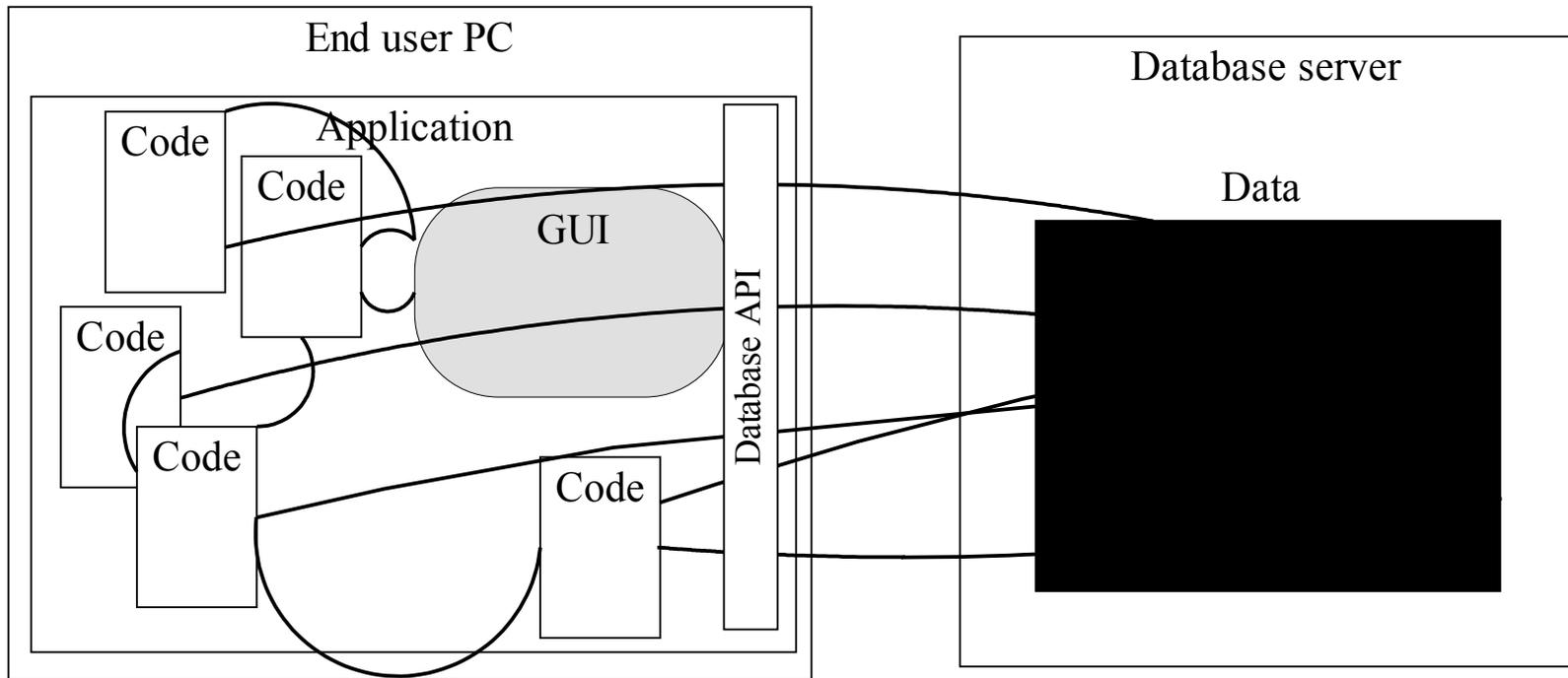
### Disadvantages

- Bulky installation – must contain all whats needed incl. optional embedded DB
- Limited by speed of the users computer
- No really good way to have multiple users sharing data at the same time
- Complex to ensure lots of users having latest version
- Complex to ensure that the data of lots of users are backed up regularly
- Complex to ensure that it will run stable on every users equipment
- If the users machine breaks down, no way to easily continue on another
- Relatively difficult to restrict data access for unauthorized users
- Huge job to make an alternative UI having similar or the same functionality.
- No LAN/WAN access to the data

# What is kbmMW? (cont.)



## 2-tier (Client - Database) C/S



# What is kbmMW? (cont.)



## 2-tier (cont.)

### Advantages

- Easy access for multiple users to operate the same data
- Spread the load over two machines, the end-user client and the DB
- Failure of an end user PC doesn't imply that others cannot continue to work
- Easy to ensure that the data is regularly backed up
- Easier to protect data from unauthorized access

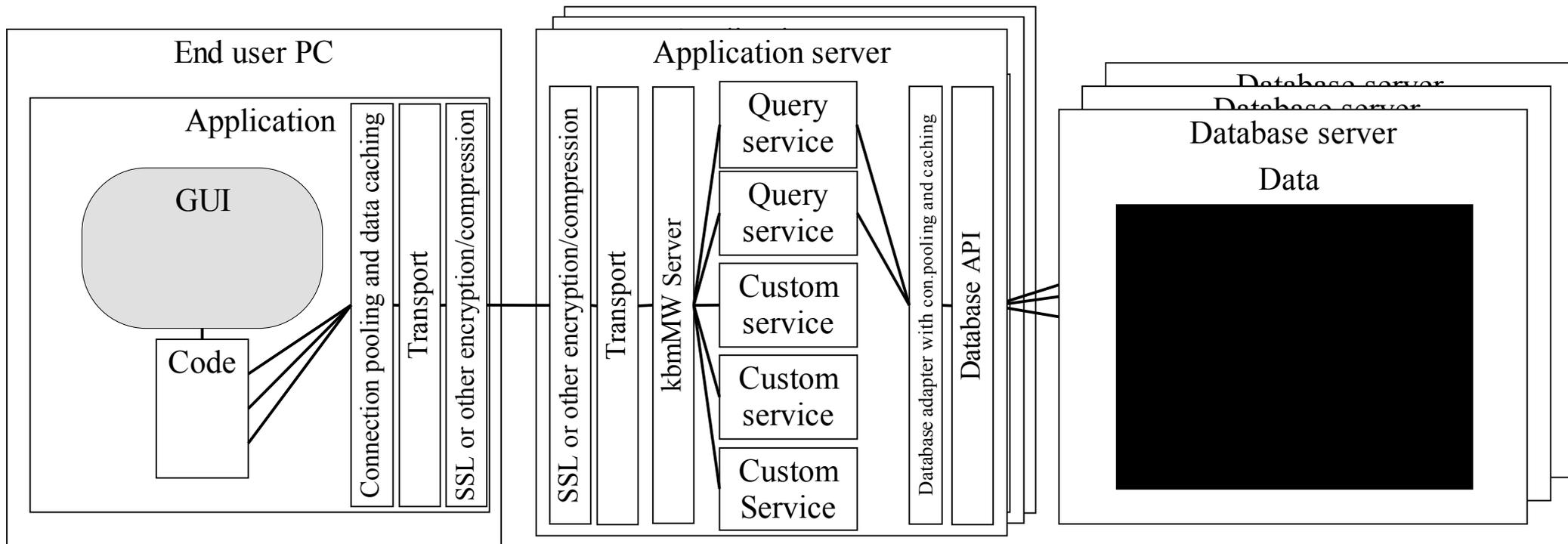
### Disadvantages

- Bulky and complex installation – typically needs installation and proper configuration of DB API's.
- Complex to ensure lots of users having latest version of the client app. and DB API
- Complex to ensure that it will run stable on every users equipment due to DB API configuration, potential firewalls etc.
- Often no protection of data while they travel over the wire
- Complex to allow for fast and secure WAN access to the data
- Complex although possible to create an alternative UI

# What is kbmMW? (cont.)



## 3-tier (Client – App. Server - Database)



# What is kbmMW? (cont.)



## 3-tier (cont.)

### Advantages

- Simple slim client installation. Client require nothing but simple socket access. Easy to ensure that clients have latest version as no complex installation is needed.
- Easy access for multiple users to operate the same data
- Spread the load over 3 or more machines (loadbalancing)
- Failure of any single component in the setup does not imply that others cannot continue to work due to failover and loadbalancing facilities
- Easy to ensure that the data is regularly backed up
- Easy to protect data from unauthorized access
- Easy to add new UI to a properly designed n-tier setup
- Easy to protect the data while they travel over the wire
- No need to expose your database to potential hackers on a LAN/WAN
- Easy to change the backend database to even another brand

### Disadvantages?... nah... not really!

- The developer needs to change the mindset. GUI, Business logic, Data

# What is kbmMW? (cont.)



## Building an application server in 5 minutes

The application server contains a server component, one or more transport components, some optional encryption components etc. and finally one or more services which are special datamodules containing business logic.

This demo show how to create an application server using **Delphi for Win32**. However it could just as well have been an application server based on **.Net** or running on **Linux!**

We create a simple form which is the visual and central part of our application server. On it we put the required components as listed above:

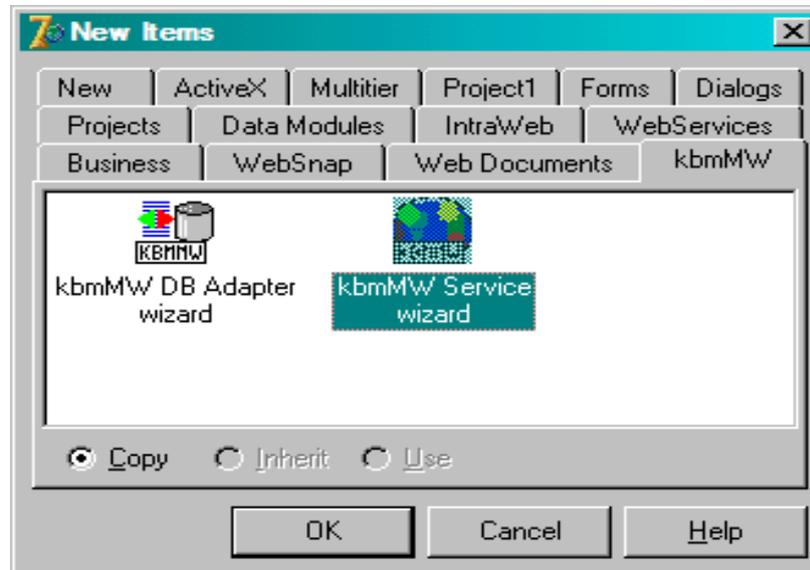


# What is kbmMW? (cont.)



The application server contains a server component, one or more transport components, some optional encryption components etc. and finally one or more services which are special datamodules containing business logic.

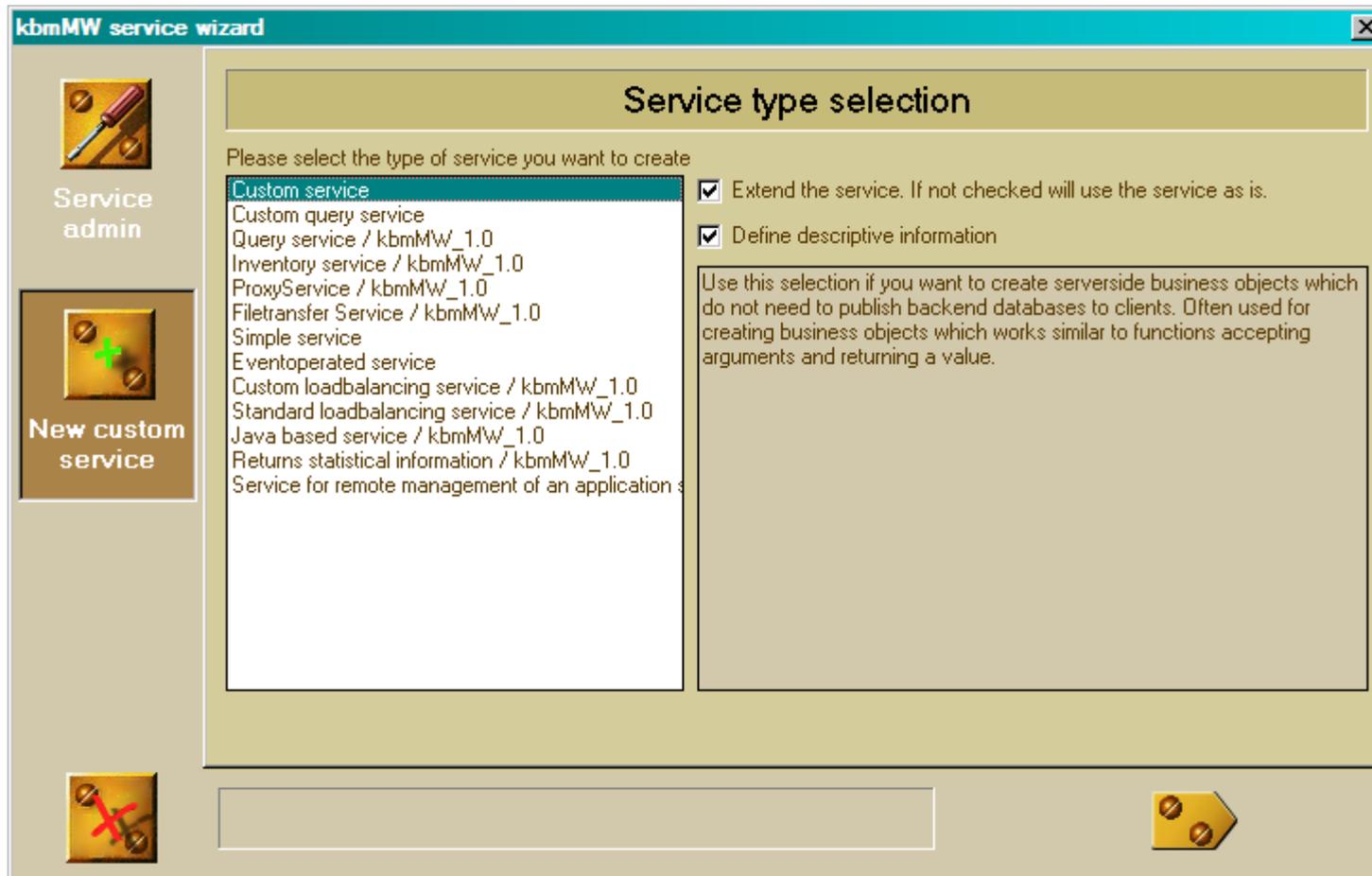
These services can easily be generated using the kbmMW Service wizard:



# What is kbmMW? (cont.)



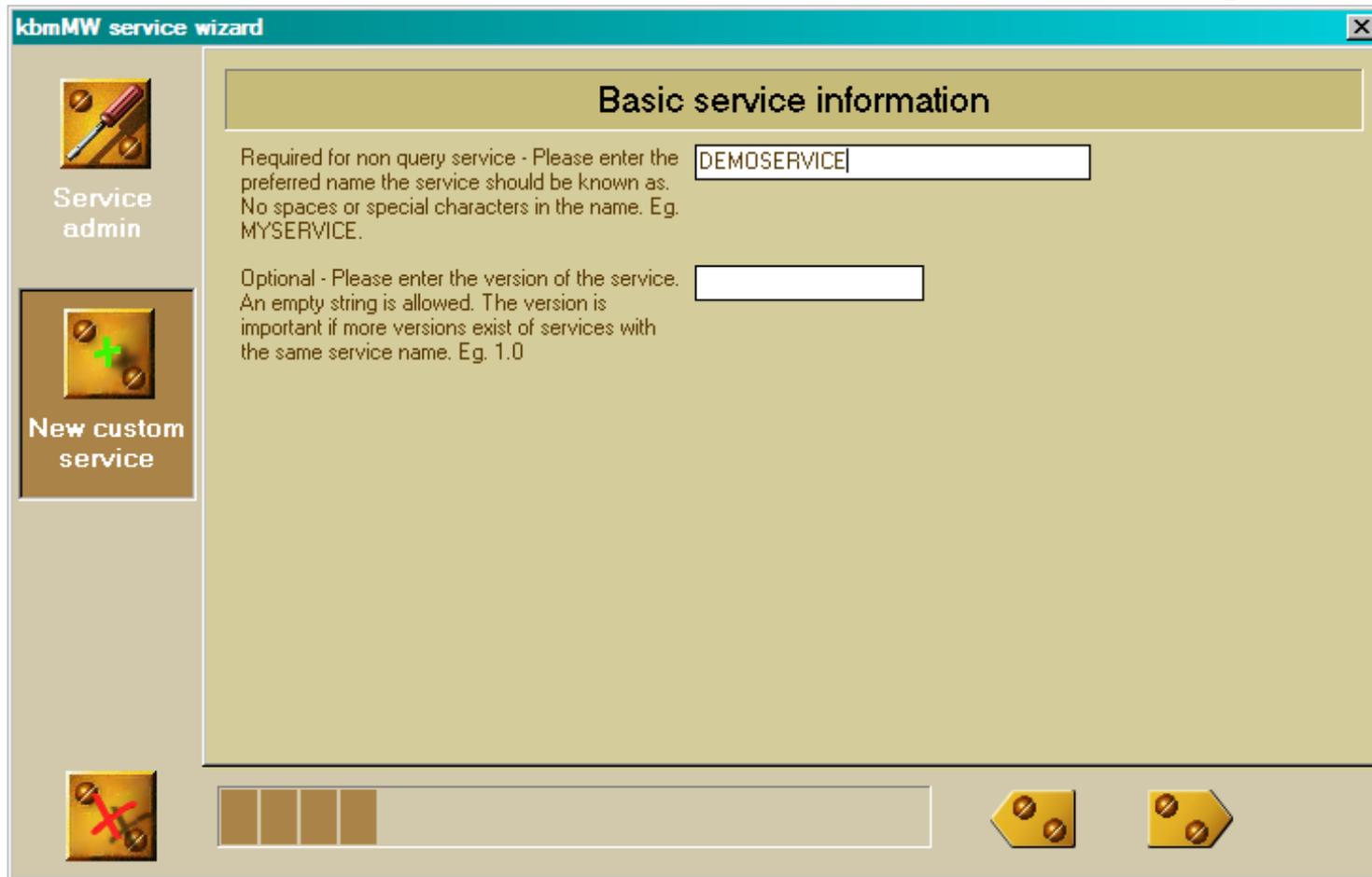
The service wizard is like an advanced repository. New custom made services can be added to the repository and be reused or extended by other developers.



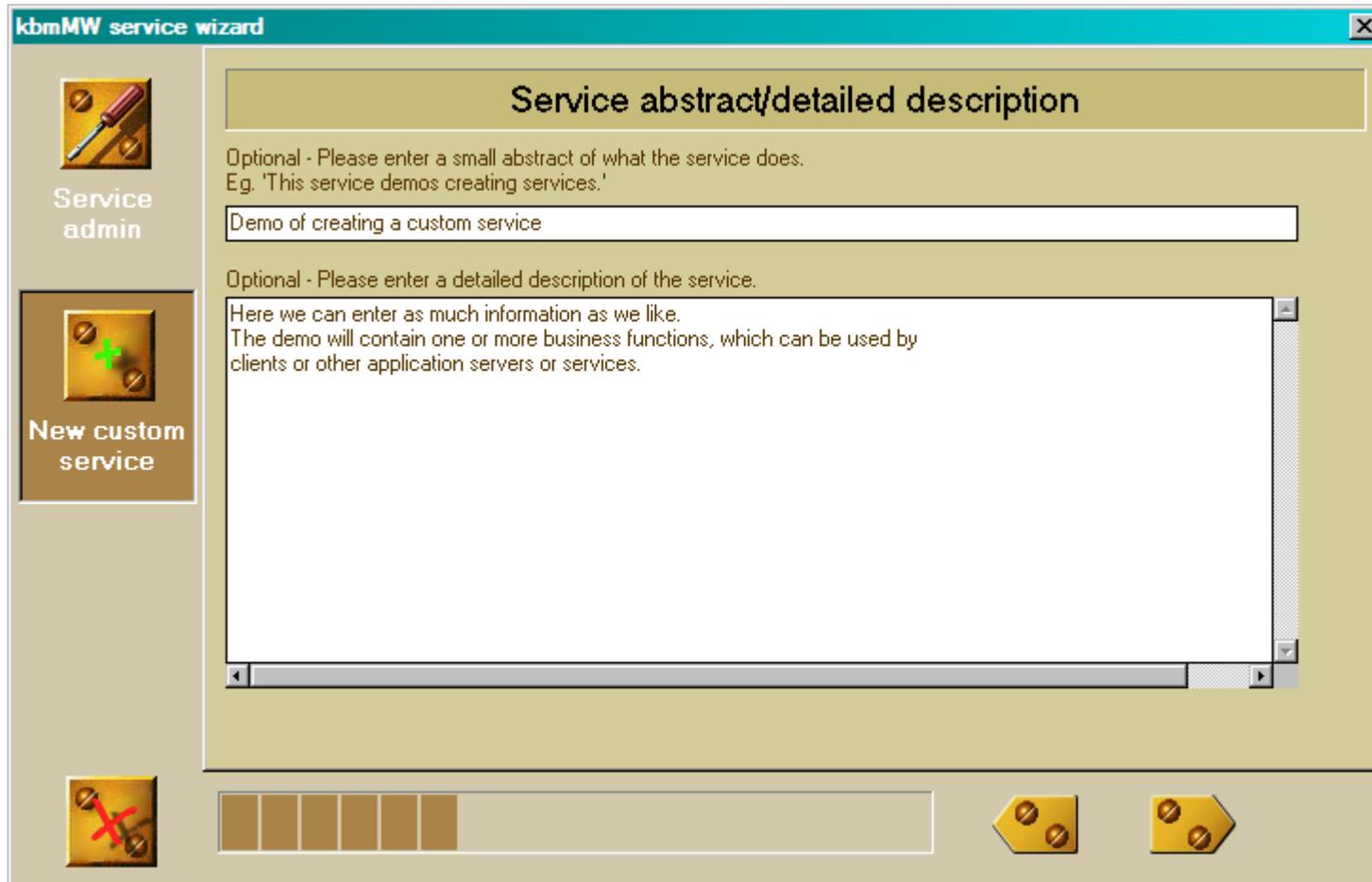
# What is kbmMW? (cont.)



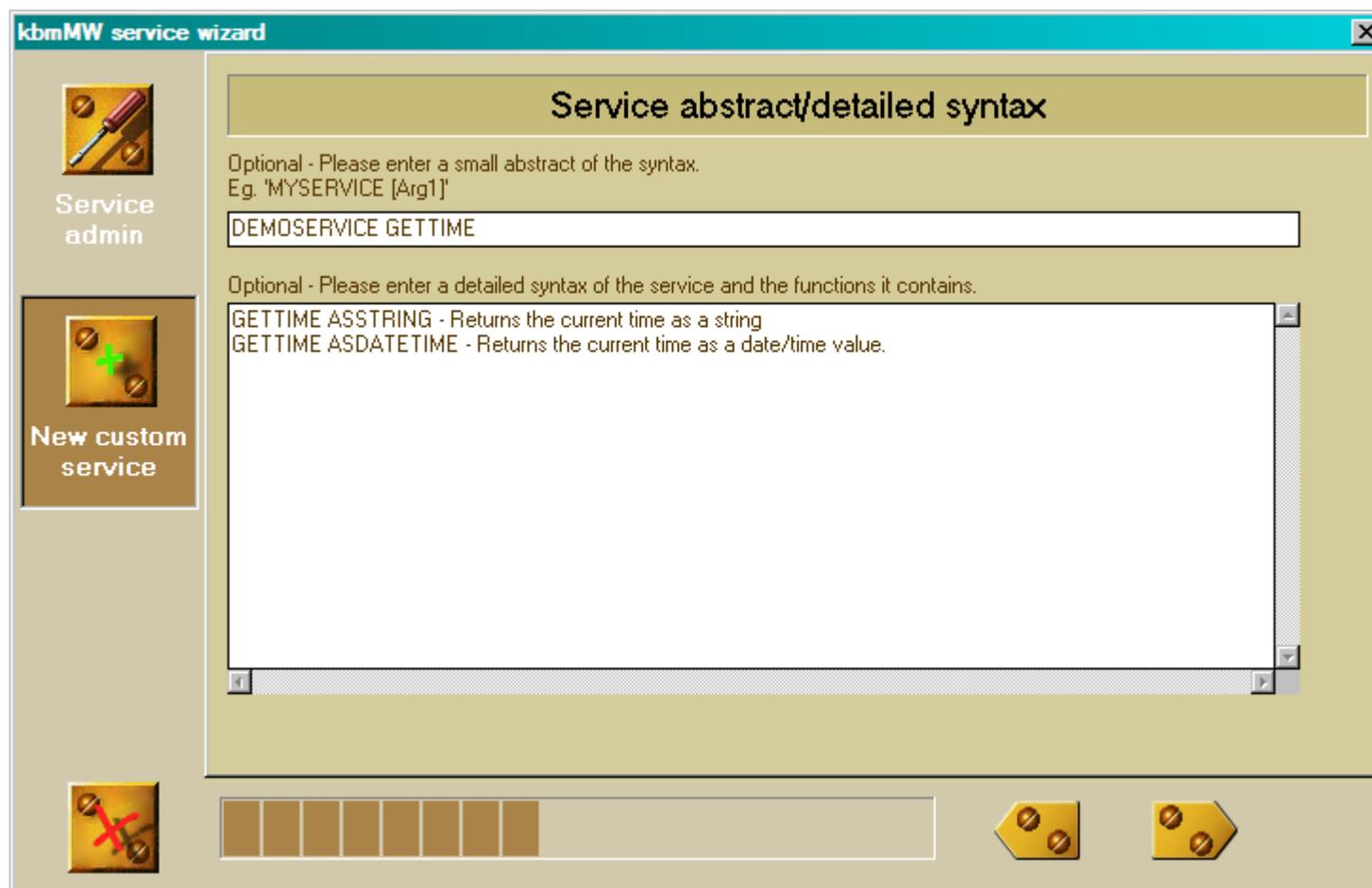
In this case we choose to create a new service based on the TkbmMWCustomService which is the base of all services. We fill out some information of which most it is optional.



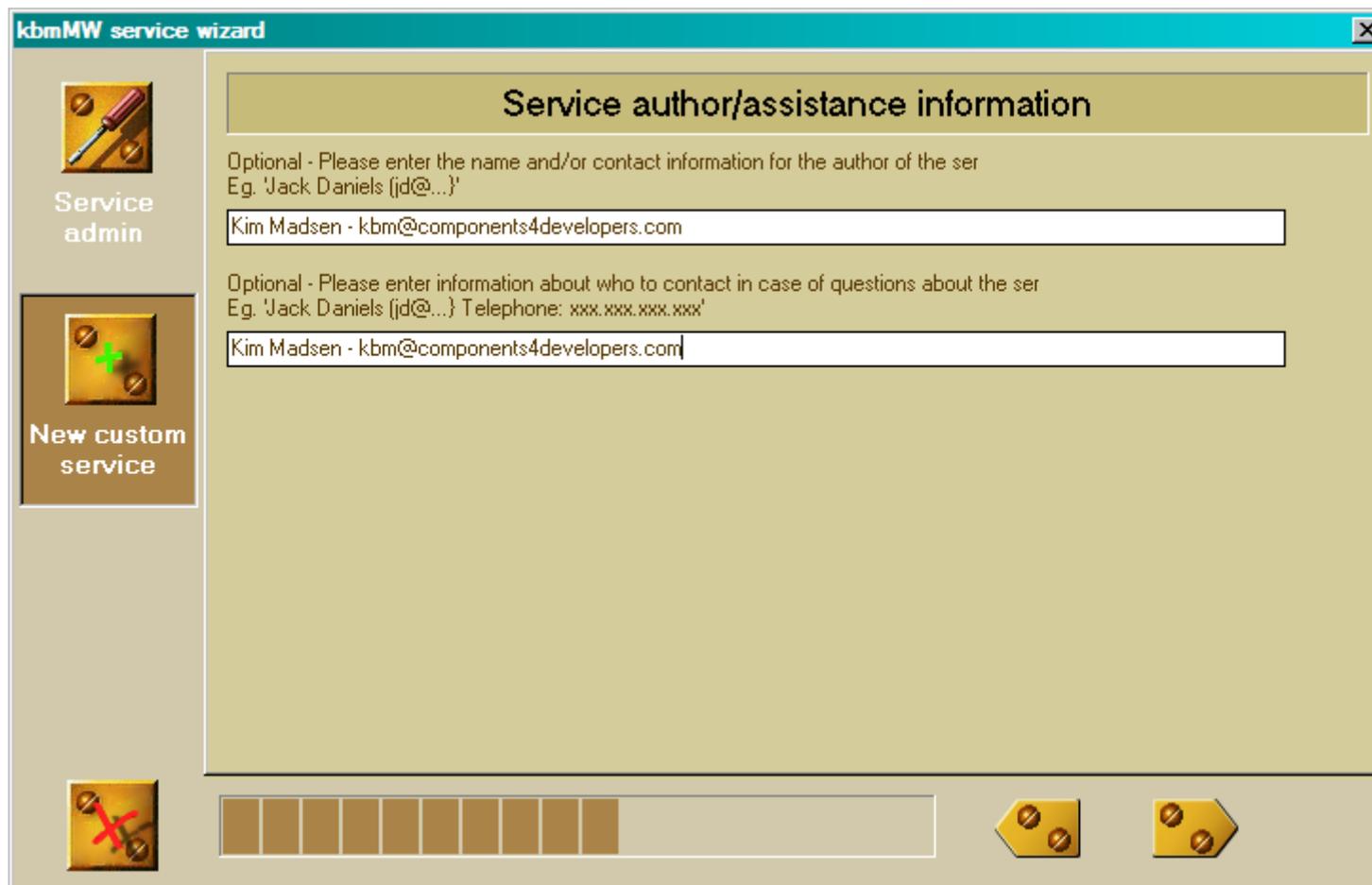
# What is kbmMW? (cont.)



# What is kbmMW? (cont.)



# What is kbmMW? (cont.)



# What is kbmMW? (cont.)



# What is kbmMW? (cont.)



**kbmMW service wizard**

  
Service admin

  
New custom service

### Service functions

Optional - Please enter the names of the functions the service should contain, one on each line.

GETTIME

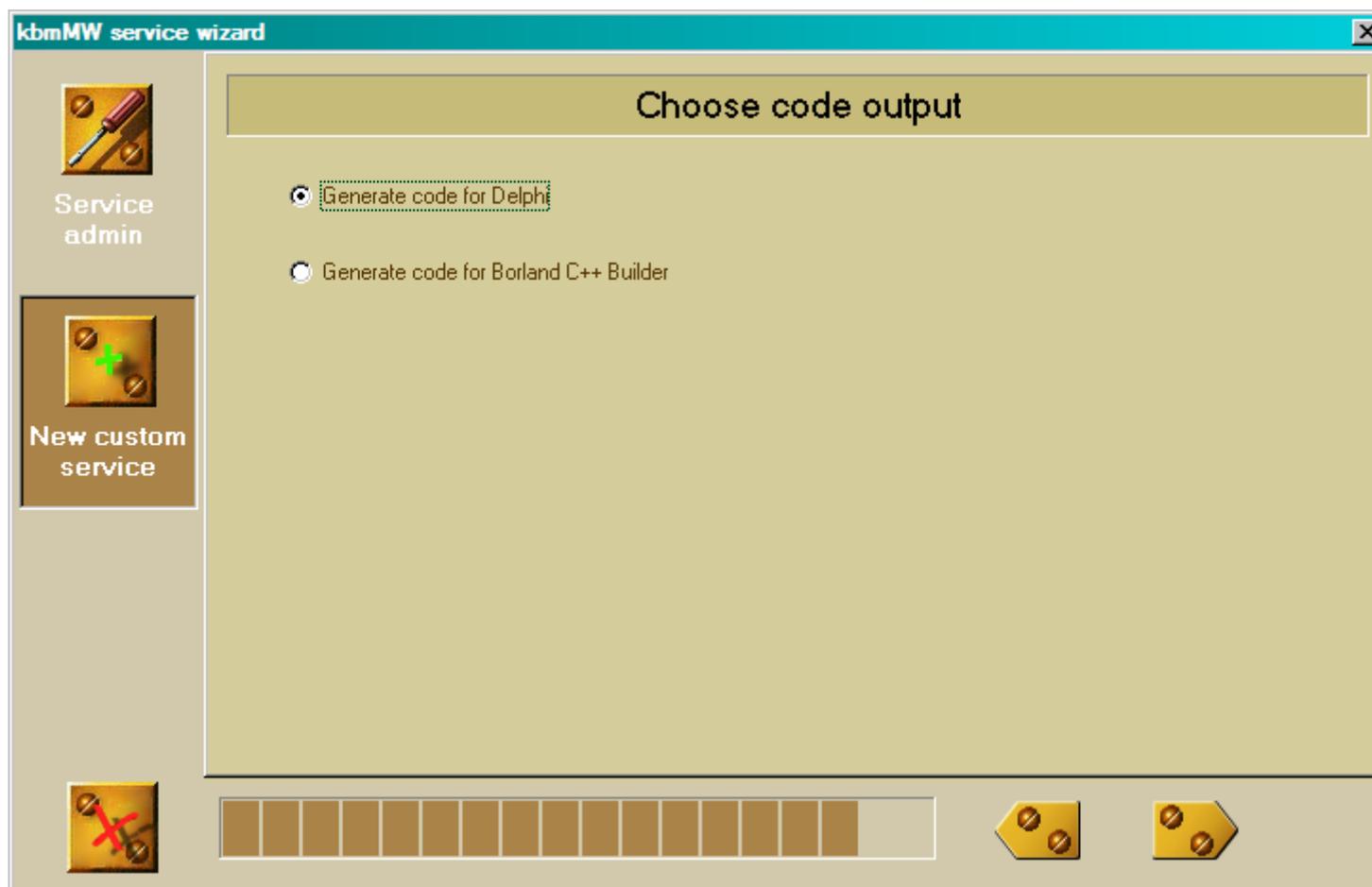
- Create empty ConstructService method.
- Create empty DestroyService method.
- Create empty ConstructFirstService method.
- Create empty DestroyLastService method.
- Create empty InitializeService method.
- Create empty FinalizeService method.
- Create empty InitializeState method.
- Create empty FinalizeState method.
- Require custom servicedefinition

# What is kbmMW? (cont.)



Now all values have been filled in, and we need to decide if the service should be in C++ or Delphi (incl. Kylix and .Net). All options chosen earlier on can easily be altered directly in the generated service module code.

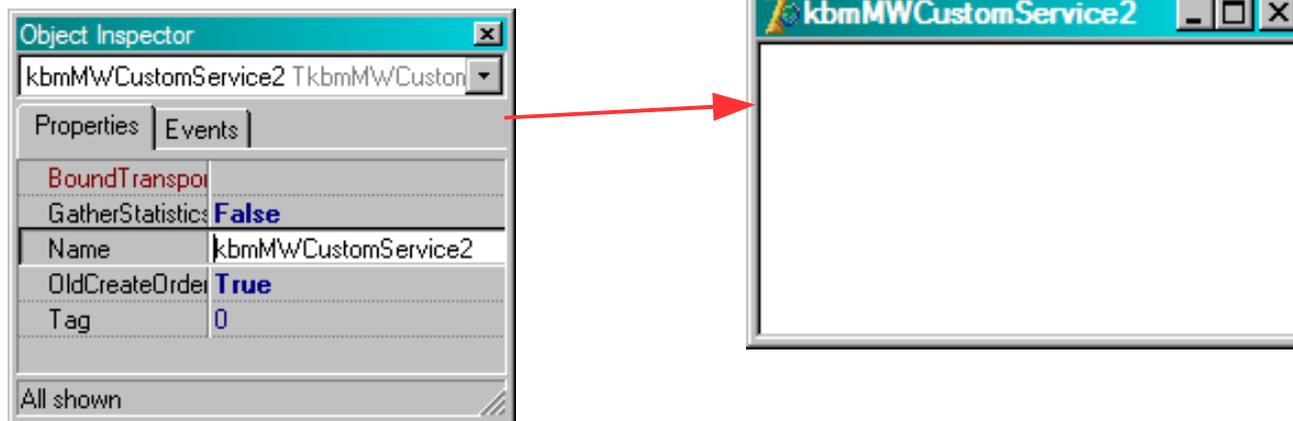


# What is kbmMW? (cont.)



Then we ends up with a service datamodule. In our case it does not contain any components, but in others it most likely will.

The datamodule have some properties which can be set at designtime



# What is kbmMW? (cont.)



Its interface code look like this (remember we selected Delphi):

```
TkbmMWCustomService2 = class(TkbmMWCustomService)
private
  { Private declarations }
protected
  { Private declarations }
  function ProcessRequest(const Func:string; const ClientId:TkbmMWClientIdentity;
                        const Args:array of Variant):Variant; override;
  function PerformGETTIME(ClientId:TkbmMWClientIdentity;
                        const Args:array of Variant):Variant; virtual;
public
  { Public declarations }
{$IFDEF CPP}class{$ENDIF} function GetPrefServiceName:string; override;
{$IFDEF CPP}class{$ENDIF} function GetAuthor:string; override;
{$IFDEF CPP}class{$ENDIF} function GetAssistance:string; override;
{$IFDEF CPP}class{$ENDIF} function GetSyntaxAbstract:string; override;
{$IFDEF CPP}class{$ENDIF} function GetSyntaxDetails:string; override;
{$IFDEF CPP}class{$ENDIF} function GetDescriptionAbstract:string; override;
{$IFDEF CPP}class{$ENDIF} function GetDescriptionDetails:string; override;
{$IFDEF CPP}class{$ENDIF} function GetFlags:TkbmMWServiceFlags; override;
end;
```

If we would have selected C++ as output, a .hpp and a .cpp file would have been generated instead.

# What is kbmMW? (cont.)



And its implementation code, where the first part of it is descriptive, looks like this:

```
// Service definitions.
//-----

{$IFDEF CPP}class{$ENDIF} function TkbmMWCustomService2.GetPrefServiceName:string;
begin
    Result:='DEMOSERVICE';
end;

{$IFDEF CPP}class{$ENDIF} function TkbmMWCustomService2.GetAuthor:string;
begin
    Result:='Kim Madsen - kbm@components4developers.com';
end;

{$IFDEF CPP}class{$ENDIF} function TkbmMWCustomService2.GetAssistance:string;
begin
    Result:='Kim Madsen - kbm@components4developers.com';
end;

{$IFDEF CPP}class{$ENDIF} function TkbmMWCustomService2.GetSyntaxAbstract:string;
begin
    Result:='DEMOSERVICE GETTIME';
end;
```

# What is kbmMW? (cont.)



(cont.)

```
{ $IFDEF CPP } class { $ENDIF } function TkbmMWCustomService2.GetSyntaxDetails:string;
begin
    Result:="GETTIME ASSTRING - Returns the current time as a string","GETTIME ASDATETIME -
Returns the current time as a date/time value.";
end;

{ $IFDEF CPP } class { $ENDIF } function TkbmMWCustomService2.GetDescriptionAbstract:string;
begin
    Result:='Demo of creating a custom service';
end;

{ $IFDEF CPP } class { $ENDIF } function TkbmMWCustomService2.GetDescriptionDetails:string;
begin
    Result:="Here we can enter as much information as we like.,""The demo will contain one
or more business functions, which can be used by","clients or other application servers or
services."";
end;

{ $IFDEF CPP } class { $ENDIF } function TkbmMWCustomService2.GetFlags:TkbmMWServiceFlags;
begin
    Result:=[mwsfListed];
end;
```

# What is kbmMW? (cont.)



(cont.)

```
{ $IFDEF CPP } class { $ENDIF } function TkbmMWCustomService2.GetSyntaxDetails:string;
begin
    Result:="GETTIME ASSTRING - Returns the current time as a string","GETTIME ASDATETIME -
Returns the current time as a date/time value.";
end;

{ $IFDEF CPP } class { $ENDIF } function TkbmMWCustomService2.GetDescriptionAbstract:string;
begin
    Result:='Demo of creating a custom service';
end;

{ $IFDEF CPP } class { $ENDIF } function TkbmMWCustomService2.GetDescriptionDetails:string;
begin
    Result:="Here we can enter as much information as we like. ","The demo will contain one
or more business functions, which can be used by","clients or other application servers or
services.";
end;

{ $IFDEF CPP } class { $ENDIF } function TkbmMWCustomService2.GetFlags:TkbmMWServiceFlags;
begin
    Result:=[mwsfListed];
end;
```

# What is kbmMW? (cont.)



Second part of the implementation is the actual business logic code:

```
// Master request processor.
//-----

function TkbmMWCustomService2.ProcessRequest(const Func:string; const
ClientIdent:TkbmMWClientIdentity; const Args:array of Variant):Variant;
var
    AFunc:string;
begin
    AFunc:=UpperCase(Func);
    if AFunc='GETTIME' then
        Result:=PerformGETTIME(ClientIdent,Args)
    else Result:=inherited ProcessRequest(Func,ClientIdent,Args);
end;

// Functions published by the service.
//-----

function TkbmMWCustomService2.PerformGETTIME(ClientIdent:TkbmMWClientIdentity;
const Args:array of Variant):Variant;
begin
    // Enter code here to perform function GETTIME
    Result:='';
end;
```

# What is kbmMW? (cont.)



Now modify the PerformGETTIME function to contain your business code.:

```
function TkbmMWCustomService2.PerformGETTIME (
    ClientId: TkbmMWClientIdentity;
    const Args: array of Variant): Variant;
begin
    if (length(Args)=0) then
        kbmMWRaiseServerException('Missing type');

    if UpperCase(Args[0])='ASDATETIME' then
        Result:=Now
    else
        Result:=DateTimeToStr(Now);
end;
```

Now the service needs to be registered to the TkbmMWServer component (for example at main form OnCreate event) and the clients will automatically have access to it.

```
kbmMWServer1.RegisterService(TkbmMWCustomService2, false);
```

# What is kbmMW? (cont.)



Finally we put in a little bit of code to activate and deactivate the server via the buttons put on the form:

```

procedure TForm1.FormCreate(Sender: TObject);
begin
    kbmMWServer1.RegisterService(TkbmMWCustomService2, false);
end;

procedure TForm1.btnListenClick(Sender: TObject);
begin
    kbmMWServer1.Active:=true;
end;

procedure TForm1.btnDontListenClick(Sender: TObject);
begin
    kbmMWServer1.Active:=false;
end;

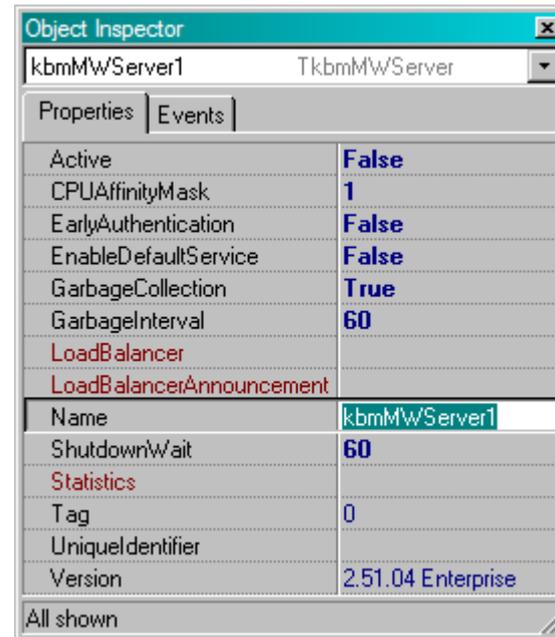
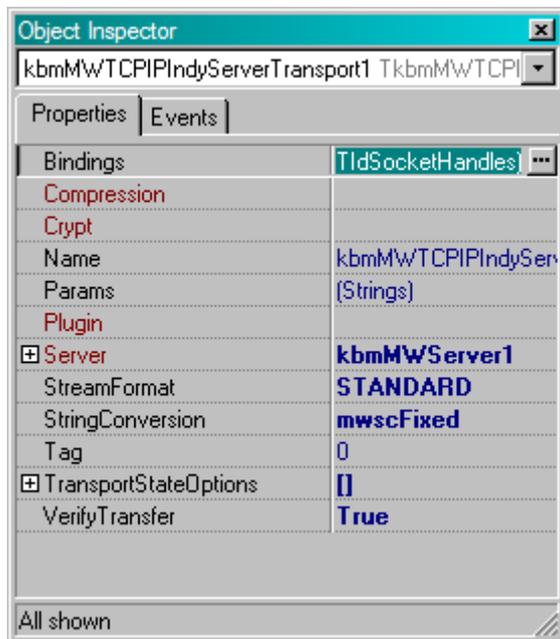
```

Thats all the code needed for our server.

# What is kbmMW? (cont.)



We can then start to play with the properties of the kbmMW components. We could for example choose how clients should access the server (SOAP/XML/HTTP/ZIPPED/STANDARD and more), on which IP port etc.



Then compile and run. The application server is ready to serve!

# What is kbmMW? (cont.)



## Building an n-tier client in less than 5 minutes

Next we want to access the application server from a client. kbmMW supports a large number of client types:

- **Delphi Win32/.Net, Kylix** and **C++Builder** based clients (all Editions)
- **SOAP** based clients (all Editions)
- **XML** based clients (all Editions)
- Native **C#** based client which is 100% compatible with Compact Framework (EE)
- Native **Java** based client which is compatible with JRE 1.3 and newer (EE)
- Native 100% portable **C** based client (EE)
- Native portable **PHP** extension for PHP v.4.2+ (EE)
- Windows **OCX** (EE)

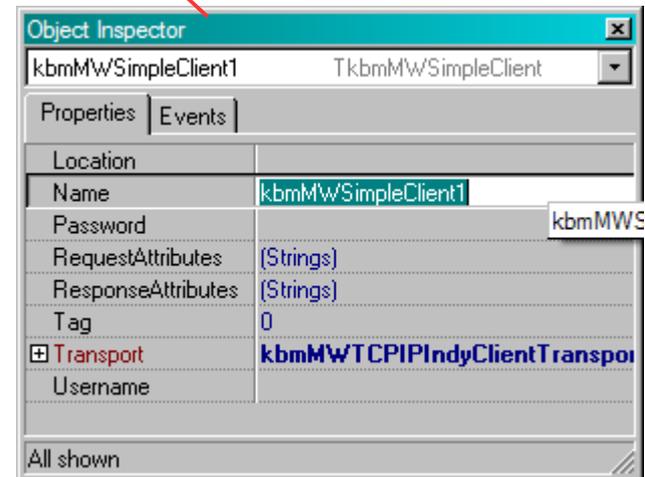
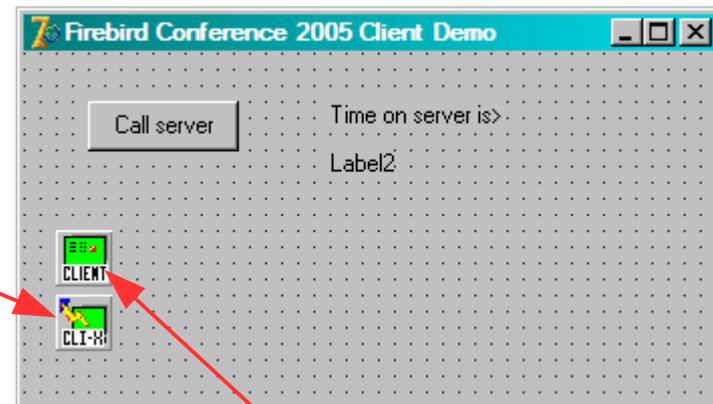
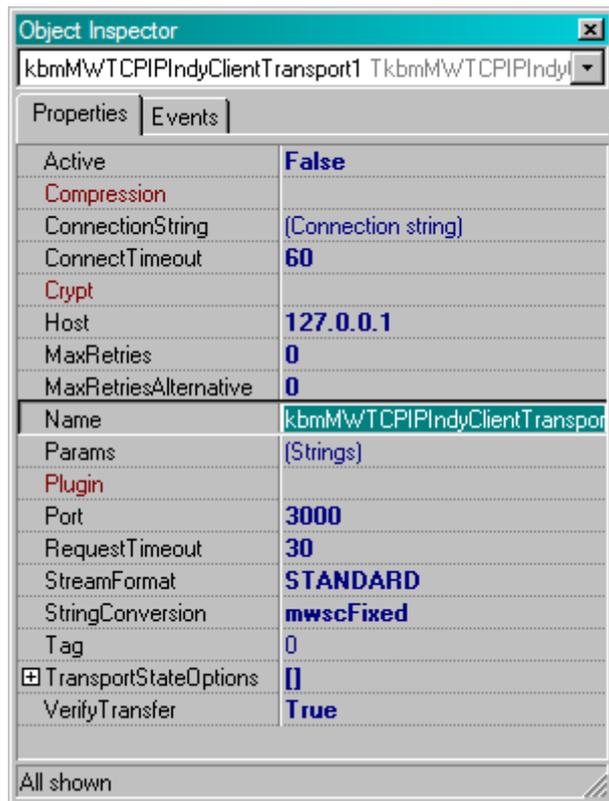
That means that the application server is accessible from **EVERYWHERE!**

# What is kbmMW? (cont.)



For this session we make a client using Delphi:

We add a TkbmMWSimpleClient and a TkbmMWTCPIIndyClientTransport to the form in addition to a TButton and a couple of labels showing the result of the call.



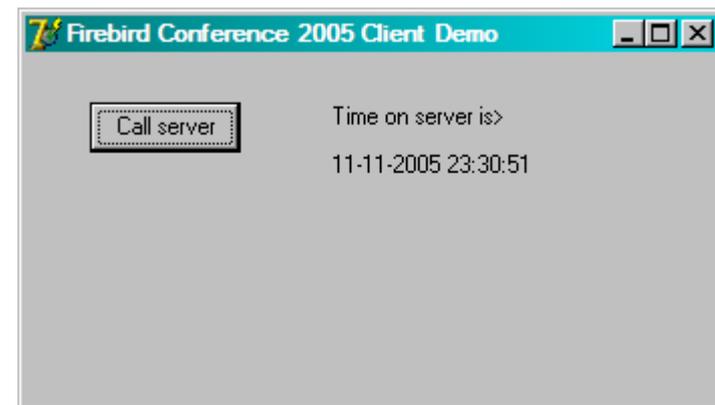
# What is kbmMW? (cont.)



Then we put a little bit of code in the buttons OnClick event to call the server:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    Label2.Caption:=kbmMWSimpleClient1.Request('DEMOSERVICE','', 'GETTIME', ['ASSTRING']);
end;
```

With the server running, a click on the client's 'Call Server' button will provide this:  
Thats all! Easy as kids play!



# What is kbmMW? (cont.)



## Bundled services

kbmMW bundles, depending on Edition, many more services. For example:

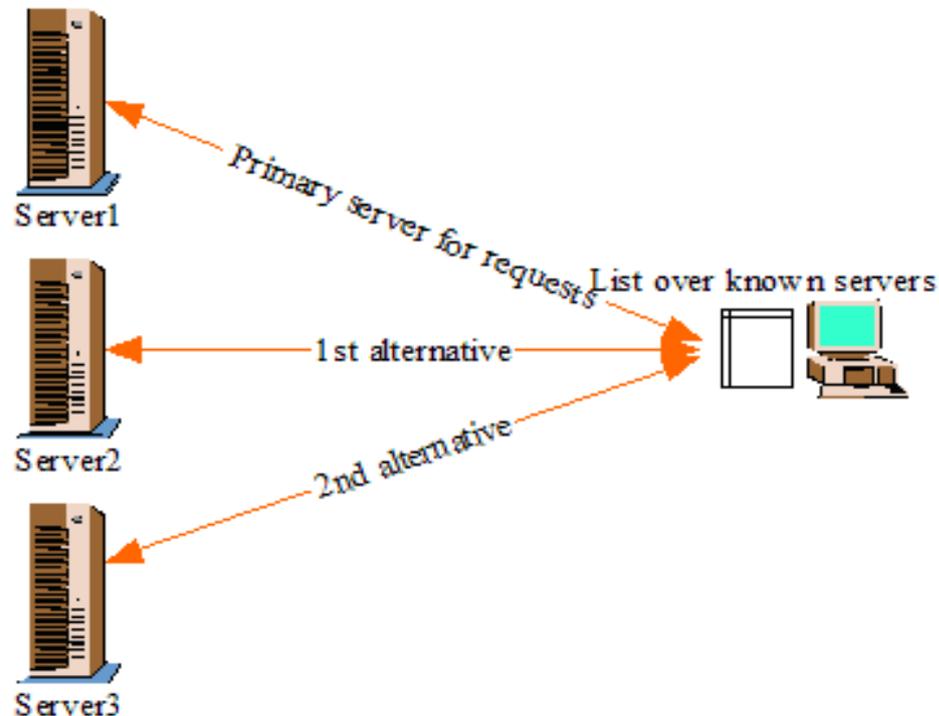
- **QueryService** which makes n-tier remote datasets very easy to use.
- **FileService** which provide a complete file management system with security.
- **JavaService** which allow for business code being written in Java instead of Delphi or C++.

# What is kbmMW? (cont.)



## Failover

If an application server dies of some reason, failover support allows the client to try connecting to alternative application servers without the user ever noticing any problem.



# What is kbmMW? (cont.)z



## Loadbalancing

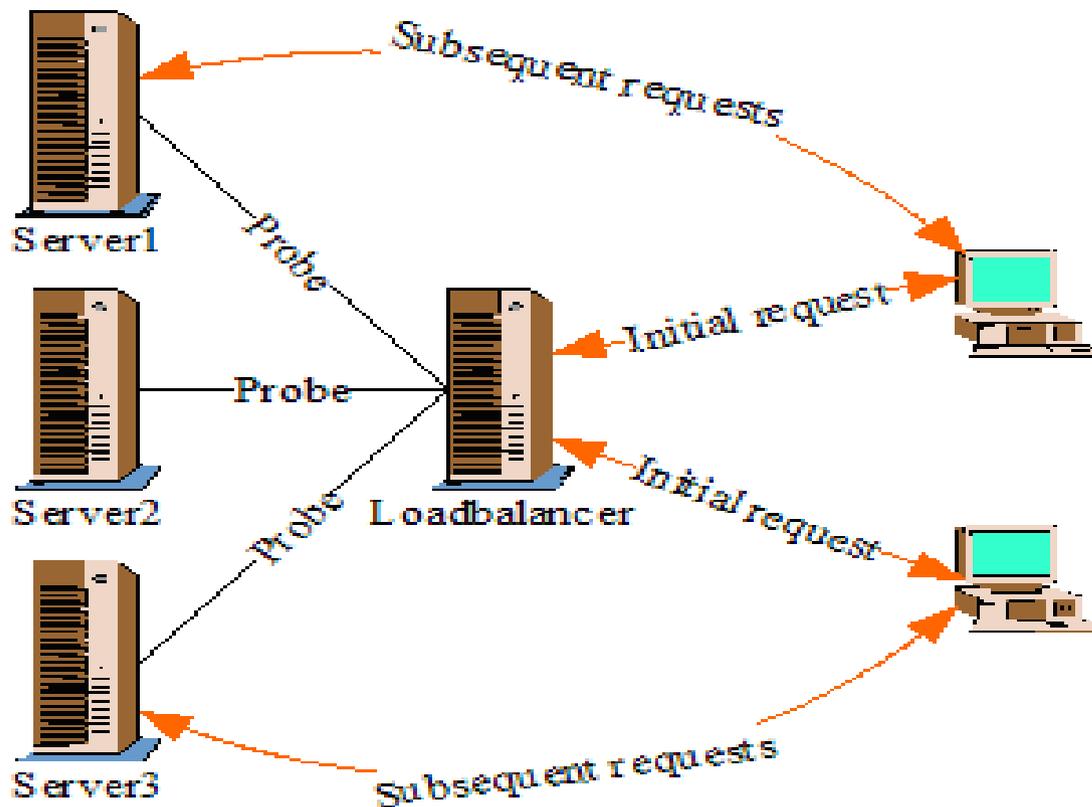
Loadbalancing support means that the client requests can be distributed between several application servers in a number of ways which can be combined:

- Centralized.  
The client always contacts one machine which then redirects the client to another application server to complete the request.
- Fully distributed.  
The client contacts any of the servers in the loadbalanced cluster directly, and they will automatically redirect the client if needed.
- Proxied.  
The client proxies through a central server which chooses which server in the loadbalanced cluster to use.

# What is kbmMW? (cont.)



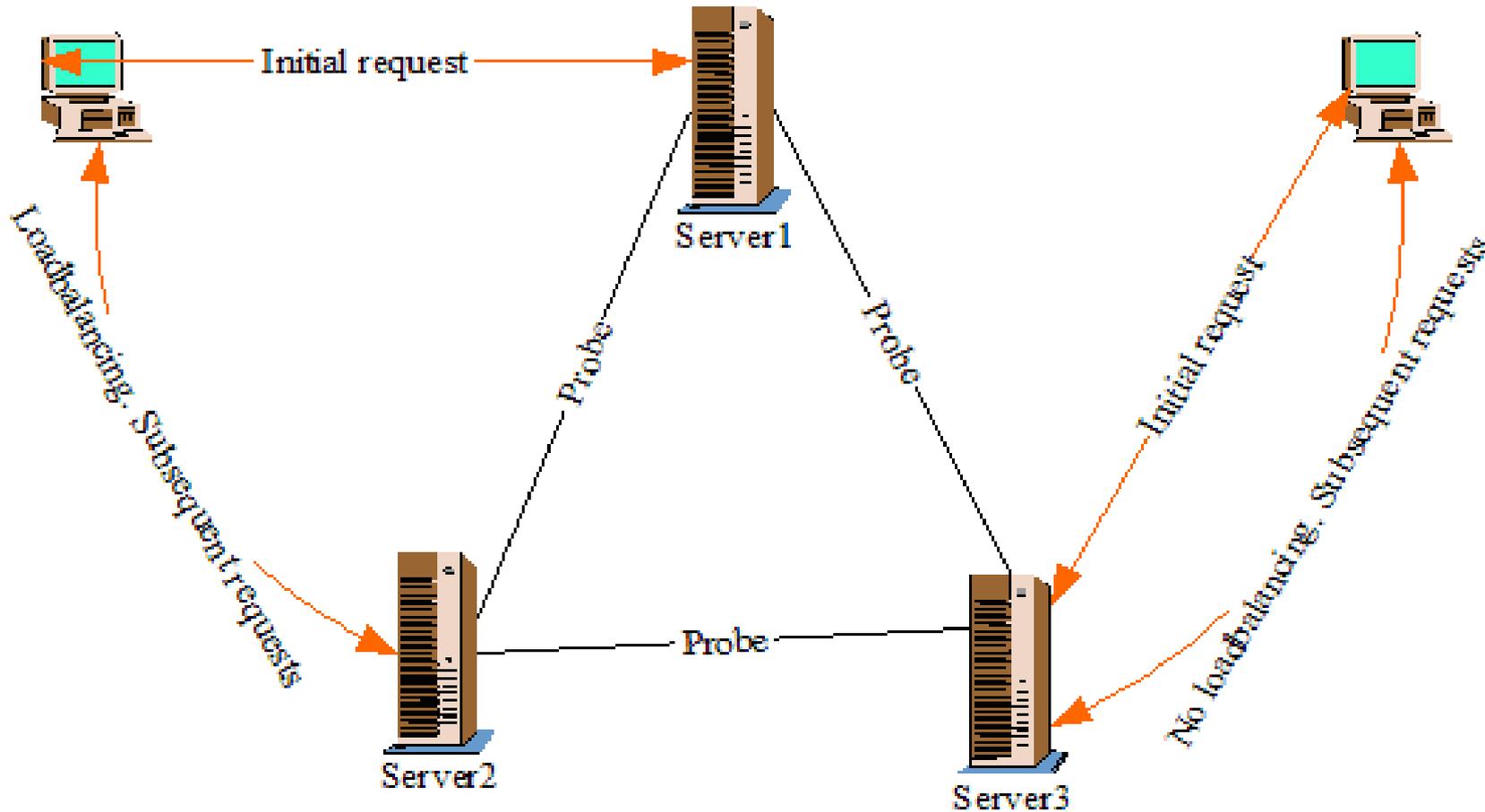
## Centralized loadbalancing



# What is kbmMW? (cont.)



## Fully distributed loadbalancing



# What is kbmMW? (cont.)



## Some keywords on kbmMW

- Very **easy** to use!
- Highly extendable!
- Stateless, statefull and persistent services support!
  - Fully configurable service pools!
  - **Lots** of connection options!
    - Highly **scalable**!
    - Multi CPU support!
      - Very **fast**!
  - No COM/DCOM/CORBA required!
- **Zero** configuration installations of app. servers and clients!
- Large array of datatypes supported, including basic types, arrays, streams, datasets, data deltas and objects!
- **ROYALTY FREE** deployment of compiled servers and clients!

# What is kbmMW? (cont.)



## KbmMW Editions

### Standard Edition (US\$0/named developer!, No runtime fees!)

- Full RPC support. Failover support. FREE even for commercial use...  
Check your business card CD!

### Professional Edition (US\$299/named developer, No runtime fees!)

Adds:

- Highly sophisticated remote database access with caching, distributed dataset delta resolving, briefcase support, metadata access, support for 35+ database API's, single property setting for switching database and much much more!

### Enterprise Edition (US\$499/named developer, No runtime fees!)

Adds:

- Sophisticated learning and teaching loadbalancing support bundling Round Robin, Random and load based algorithms.
- Highly sophisticated, state of the art, publish/subscribe messaging for realtime or near realtime messaging between nodes.
- Several native client types for access from everywhere.
- Detailed and pluggable Windows Performance Monitor support + much more!

# Questions



?



**Thank you !**