“Politehnica” University of Timisoara
Faculty of Automation and Computer Science

Master of “Automotive Embedded Software”

Dissertation Project

Business Software Solution – virtualMag

By

Zivojin Mirca

Supervisor:
Prof. Vasile Stoicu – Tivadar

2003

Table of Contents

Chapter 1: Introduction
1.1 Abstract
1.2 Project description

Chapter 2: Literature review
2.1 About Web Services
2.2 About Apache Axis engine
2.3 About Jasper Reports

Chapter 3: Project Specifications
3.1 Project design
3.2 System functionalities
3.3 User interfaces
3.4 Database structure
3.5 System communication
3.6 Type of reports
3.7 Security and risk analysis
3.8 Project management

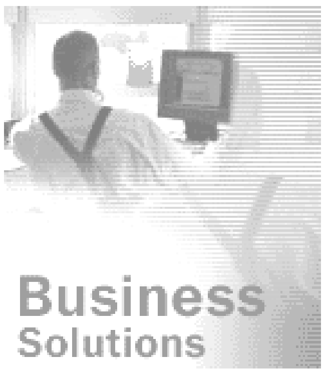
Chapter 4: Project design
4.1 Programming structure
4.2 Component description
4.3 Sequence diagram

Chapter 5: Installation

Chapter 6: Final Results
6.1 Component Tests
6.2 System Tests
6.3 Tests results
Chapter 7: Conclusions

Chapter 8: Bibliography

Appendix A



1 INTRODUCTION

6.1 Abstract

In today’s business environment is necessary to have information’s as fast is possible, to achieve this we need a software solution for this problem. Internet is an important and wary powerful environment for getting information’s faster and is accessible from any part of globe.

The goal of this project is to integrate more applications that work's together and to have a unique communication language (XML – Extensible Markup Language), this is possible trough a web services applications.

This project can be done and without integration with Web services by creating a desktop application in Microsoft™ Visual Basic that interrogates database using Microsoft™ Jet OLE DB Provider or by Microsoft™ SQL Server. But in this case the problem that arises is integration with other web applications that has to publish or to interrogate that database and the most important the costs. In this case we can also do integration between applications using the Microsoft™ Web services technology but during the extended costs I decided for Java Sun™ technology, Tomcat Apache web server with Axis 1.1 Web Services [***2003,a] engine and MySql server for database.

1.2. Project description

The aim of this project is to integrate trough a web services this business functionalities:

- *Administration of Warehouse*
- *Marketing and Management (B2B and B2C)*
- *Administration of Reports and Documents*

All this functionalities has to be designed and integrated to communicate to each other trough a web interface and trough desktop application and web browser. Database will be at single place and will contain data for all applications.

Administration of Warehouse, it means a desktop application that represents a client that exchange data with database server trough Web service. This client is very important for administrating the warehouse. The main functionalities that have to implement are:

a) **Add new item into warehouse**, when new items arrives they must to be introduced into warehouse and then using this functionality the warehouse administrator introduces the items details into database, prints the items bar code label, sets the item inspection and places them into warehouse.

b) **Remove the item from warehouse**, items from warehouse is used for creating the final products that are delivered to the client. Using this functionality the item is removed from warehouse.

c) **Modify the item inspection**, when the expiration date has expired the item has to be re-inspected, the quality result can be 3 for accepted, 2 for good for use and 0 if is rejected, also when re-inspection is done the new bar code label has to be printed.

d) **Add, edit, delete furnisher**, this functionalities are used for furnishers administration.

e) **Add, edit, delete manufacturer**, this functionalities are used for manufacturer administration.

f) **Add, edit, delete item,** this functionalities are used for items administration.

g) **Add, edit, delete item class,** this functionalities are used for item class administration.

h) **Item tracking**, this functionality is used for tracking the items that are introduced into warehouse and items that has bean removed from warehouse.

i) **Search items** searches the item into warehouse by date of reception, item index and by item bar code label. After the item is found it displays details about that item (placement, expiration date and stock)

j) **Final products administration**, for this functionality we need to track all items that has bean used and returned for that final product.

k) **Create reports**, this functionality creates reports for:

1. Warehouse inventory
2. Item tracking
3. Final product items

Marketing and Management, it means a possibility to create web presentations of products. The presentations are represented by product categories and alphabetically. Every product has a picture and a description. This is done from a web browser and has to have these functionalities:

a) **Administrate the web catalogs**, by this we can create different catalogs for different product categories.

b) **Administrate the web products**, by this we can create a product that belongs to a particular category with a brief description and a picture.

c) **Administrate the clients,** by this we can administrate the clients and to send him the promotional offers.
d) **Create the web reports**, by this we can create reports that are regarded to the items tracking from warehouse.

Administration of Reports and Documents, this represents a service that creates reports in PDF, HTML or XLS document format. This service deserves all applications that need to create reports from data stored into database.
Reports are generated using Jasper Reports [***2001,d] explained letter in this document.

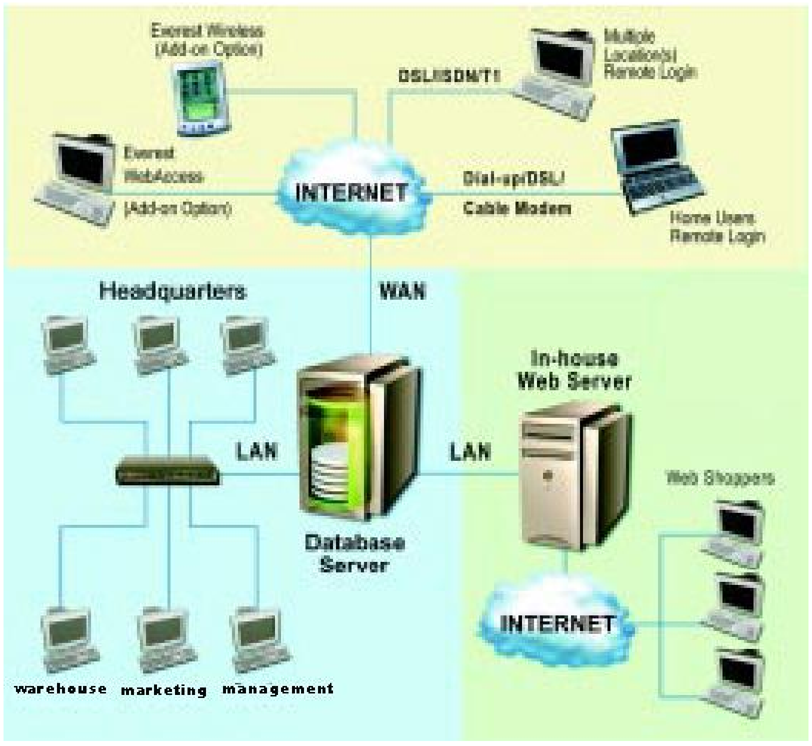
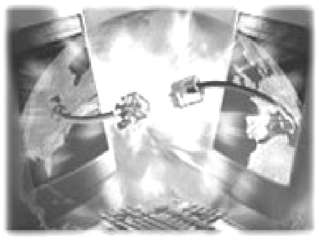


Figure 1.0 Application Structure



2 Literature review

2.1 About Web Services

Web services, in the general meaning of the term, are services offered via the Web. In a typical Web services scenario, a business application sends a request to a service at a given URL using the SOAP protocol over HTTP [***2000,b]. The service receives the request, processes it, and returns a response (see figure x.x). An often-cited example of a Web service is that of a stock quote service, in which the request asks for the current price of a specified stock, and the response gives the stock price. This is one of the simplest forms of a Web service in that the request is filled almost immediately, with the request and response being parts of the same method call. Another example could be a service that maps out an efficient route fo r the delivery of goods. In this case, a business sends a request containing the delivery destinations, which the service processes to determine the most cost-effective delivery route. The time it takes to return the response depends on the complexity of the routing, so the response will probably be sent as an operation that is separate from the request.

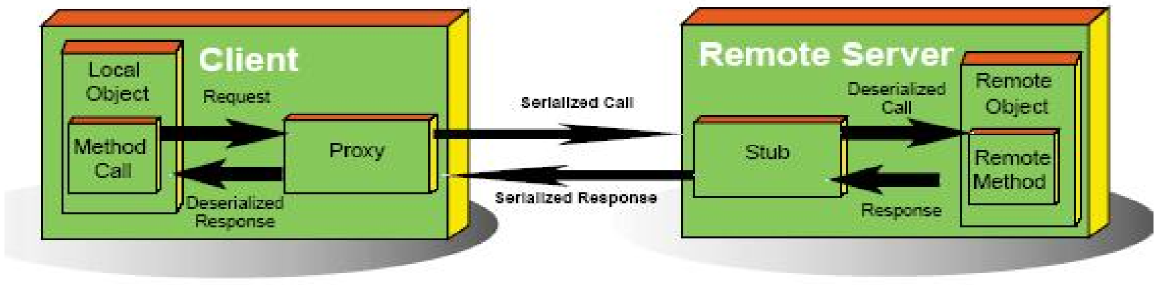


Figure 2.0 – Web Services Architecture

Web services and consumers of Web services are typically businesses, making Web services predominantly business-to-business (B-to-B) transactions. An enterprise can be the provider of Web services and also the consumer of other Web services. For example, a wholesale distributor of spices could be in the consumer role when it uses a Web service to check on the availability of vanilla beans and in the provider role when it supplies prospective customers with different vendors' prices for vanilla beans.

Business-to-Consumer is an additional avenue for Web services. The discovery capability of the technology may finally make it possible for intelligent agents to break into the mainstream. Finally, we may be able to have our computers go out on the Web, find vendors who provide the services we're interested in, determine the lowest bidder (or other criteria, such as immediate availability), and conclude a transaction.
It may also be the need to interact with other companies at a process level rather than at a transactional level. Business conditions change frequently – too frequently in some cases for transactions to be fully defined. We might see Web services ride the tide of business process management, providing the infrastructure, or even the structure itself, of collaborative business. Underlying all of this will be a plethora of technologies. As is typical of the information technology world, there are competing standards and platforms. Which in a way is not all bad? The reality of Web services is that we are pursuing a neutral course in terms of data and data structure, and layering services on top of transformable, machine-independent data. The ability to select a particular platform to provide Web services is an advantage to all – it future-proofs the existing and often enormous investment companies have made in their operational platforms [***2003,g].

2.2 About Apache Axis engine

Apache AXIS is an implementation of the SOAP ("Simple Object Access Protocol") submission to W3C. From the draft W3C [***2000,b] specification "SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined data types, and a convention for representing remote procedure calls and responses".

Architectural overview

Axis is all about processing Messages. When the central Axis processing logic runs, a series of **Handlers** are each invoked in order. The particular order is determined by two factors – deployment configuration and whether the engine is a client or a server. The object, which is passed to each Handler invocation, is a **MessageContext**. A MessageContext is a structure, which contains several important parts: 1) a "request" message, 2) a "response" message, and 3) a bag of properties.

There are two basic ways in which Axis is invoked:

1. As a **server**, a **Transport Listener** will create a MessageContext and invoke the Axis processing framework.
2. As a **client**, application code (usually aided by the client programming model of Axis) will generate a MessageContext and invoke the Axis processing framework.

In either case, the Axis framework's job is simply to pass the resulting MessageContext through the configured set of Handlers, each of which has an opportunity to do whatever it is designed to do with the MessageContext.

Message Path on the Server

The server side message path is shown in the following diagram. The small cylinders represent Handlers and the larger, enclosing cylinders represent **Chains** (ordered collections of Handlers which will be described shortly).

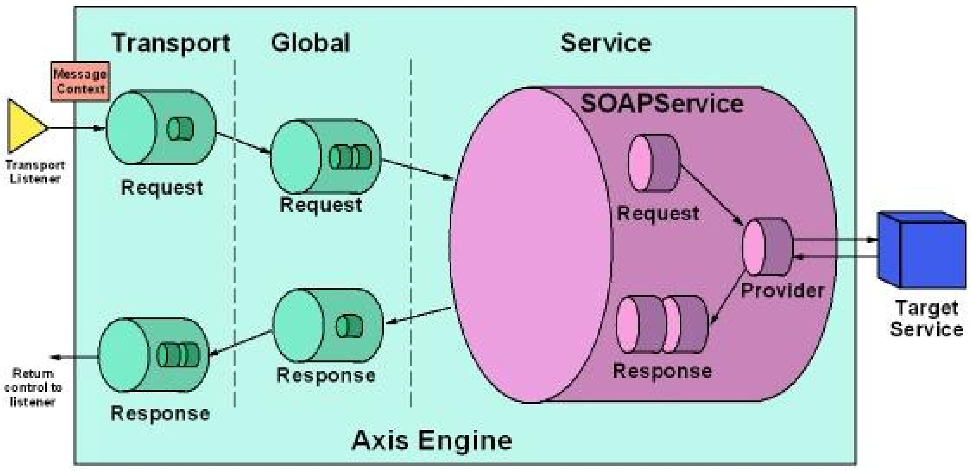


Figure x.x - The Message Path on the Server

A message arrives (in some protocol-specific manner) at a Transport Listener. In this case, let's assume the Listener is a HTTP servlet. It's the Listener's job to package the protocol-specific data into a **Message** object (org.apache.axis.Message), and put the Message into a **MessageContext**. The Listener also loads the MessageContext with various properties. The Transport Listener also sets the **transportName** String on the MessageContext. Once the MessageContext is ready to go, the Listener hands it to the AxisEngine.

The AxisEngine's first job is to look up the transport by name. The transport is an object, which contains a **request** Chain, a **response** Chain, or perhaps both. A **Chain** is a Handler consisting of a sequence of Handlers, which are invoked in turn.

If a transport request Chain exists, it will be invoked, passing the MessageContext into the invoke() method. This will result in calling all the Handlers specified in the request Chain configuration.

After the transport request Handler, the engine locates a global request Chain, if configured, and then invokes any Handlers specified therein.

At some point during the processing up until now, some Handler has hopefully set the **serviceHandler** field of the MessageContext (this is usually done in the HTTP transport by the “URLMapper” Handler, which maps a URL like “http://localhost/axis/services/AdminService” to the “AdminService” service). This field determines the Handler we’ll invoke to execute service-specific functionality, such as making an RPC call on a back-end object.

Services in Axis are typically instances of the “SOAPService” class (org.apache. axis.handlers.soap.SOAPService), which may contain **request** and **response** Chains (similar to what we saw at the transport and global levels), and must contain a **provider**, which is simply a Handler responsible for implementing the actual back end logic of the service.

For RPC-style requests, the provider is the org.apache.axis.providers.java. RPCProvider class. This is just another Handler that, when invoked, attempts to call a backend Java object whose class is determined by the “className” parameter specified at deployment time. It uses the SOAP RPC convention for determining the method to call, and makes sure the types of the incoming XML-encoded arguments match the types of the required parameters of the resulting method.

The Message Path on the Client

The Message Path on the client side is similar to that on the server side, except the order of • ircea• g is reversed, as shown below.

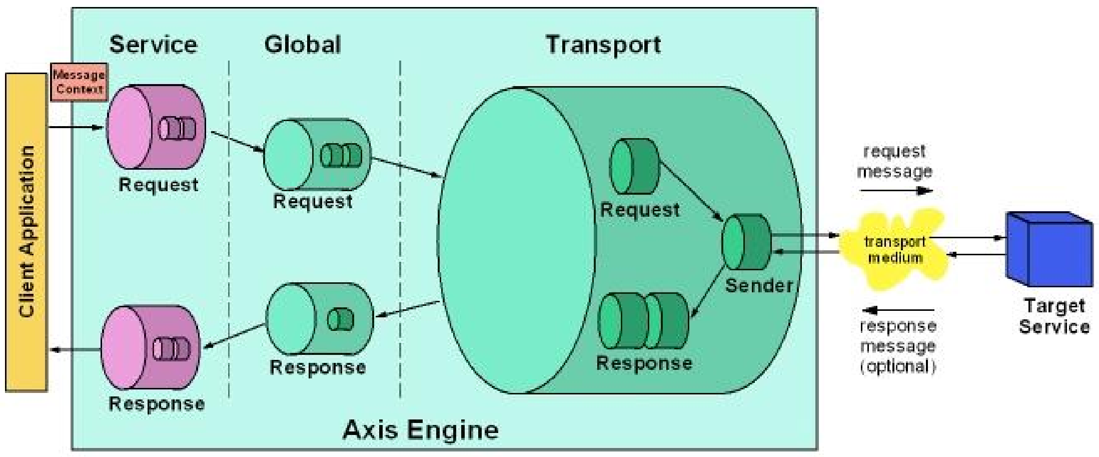


Figure x.x - The Message Path on the Client

The **service** Handler, if any, is called first – on the client side, there is no "provider" since the service is being provided by a remote node, but there is still the possibility of request and response Chains. The service request and response Chains perform any service-specific processing of the request message on its way out of the system, and also of the response message on its way back to the caller.

After the service request Chain, the global request Chain, if any, is invoked, followed by the transport. The **Transport Sender**, a special Handler whose job it is to actually perform whatever protocol-specific operations are necessary to get the message to and from the target SOAP server, is invoked to send the message. The response (if any) is placed into the responseMessage field of the MessageContext, and the MessageContext then propagates through the response Chains – first the transport, then the global, and finally the service.

6.1 About Jasper Reports

JasperReports is a powerful report-generating tool that has the ability to deliver rich content onto the screen, to the printer or into PDF, HTML, XLS, CSV and XML files. It is entirely written in Java and can be used in a variety of Java enabled applications to generate dynamic content [***2001.d].

Its main purpose is to help creating page oriented, ready to print documents in a simple and flexible manner. JasperReports organizes data retrieved from a relational database through JDBC according to the report design defined in an XML file. In order to fill a report with data, the report design must be compiled first.

Through compilation, the report design is loaded into a report design object that is then serialized and stored on disk (dori.jasper.engine.JasperReport). This serialized object is then used when the application wants to fill the specified report design with data. In fact, the compilation of a report design implies the compilation of all Java expressions defined in the XML file representing the report design. Various verifications are made at compilation time, to check the report design consistency. The result is a ready to fill report design that will be then used to generate documents on different sets of data. In order to fill a report design, one can use the fillReportXXX() methods exposed by the dori.jasper.engine.JasperManager class. Those methods receive as a parameter the report design object, or a file representing the specified report design object, in a serialized form, and also a JDBC connection to the database from where to retrieve the data to fill the report. The result is an object that represents the ready to print document (dori.jasper.engine.JasperPrint) and can be stored onto the disk, in a serialized form, for later use, or can be delivered to the printer, to the screen or can be transformed into a PDF, HTML, XLS, CSV or XML document.

3

Project Specifications

3.1 Project design

In this project we have used tree-tier layer design composed of the client – web application server – database server. In the figure x.x you can see these components.

System diagram

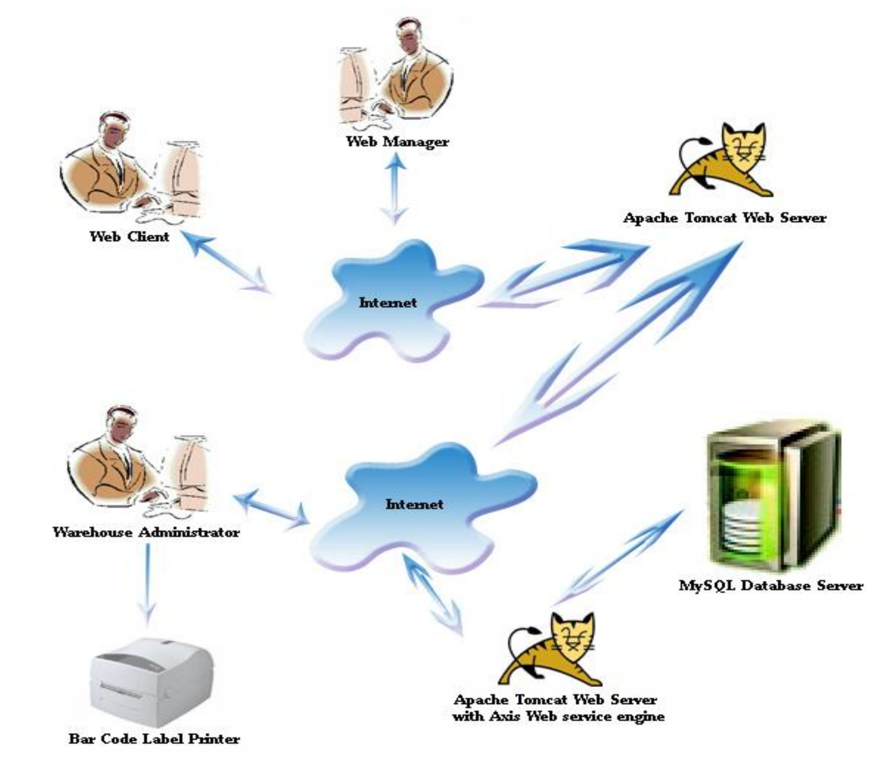


Figure x.x

I’ll describe now the components from this figure:

1. MySQL Database server [***1995,c] – this represents the main database in witch are stored all data used in the client applications
2. Apache Tomcat Web Server [***1999,e] with Axis Web Services engine – this represents a Web Services engine that communicate with clients and database server, I’ll describe the communication protocol lather.
3. Warehouse Administrator – this is a client written in Visual Basic language and represents the client that administrates the warehouse using this application. This is useful because all data are stored in the main database and it can be accessed by other authorized clients like “Marketing and Management”
4. Bar Code Label Printer – this is a printer device that is connected to the “Warehouse Administrator” computer and is meant for printing the items bar code for items in the warehouse.

Until now I have described the low level of the infrastructure and that means that has to be integrated into ***intranet*** *infrastructure.*

5. Apache Web Server – this represents the web server on witch is installed the company web site and also the api interface for communication with Web Services application server.
6. Web Manager – this represents the “Marketing and Management” user and he is responsible for client’s administration and also for publishing on the web the company’s products.
7. Web Client – this an end-user that can access the company’s web site and to see the latest company’s products.

The web server on witch is installed the company’s web site ca be any where in the world and then the communication with the Web Services server is done trough the internet or it can be installed on the same server on witch is installed the Web Services an than we’ll use the intranet.

Use – Case diagram

Now I'll like to present the use – case diagram for the customer, warehouse administrator and manager (Marketing and Management)

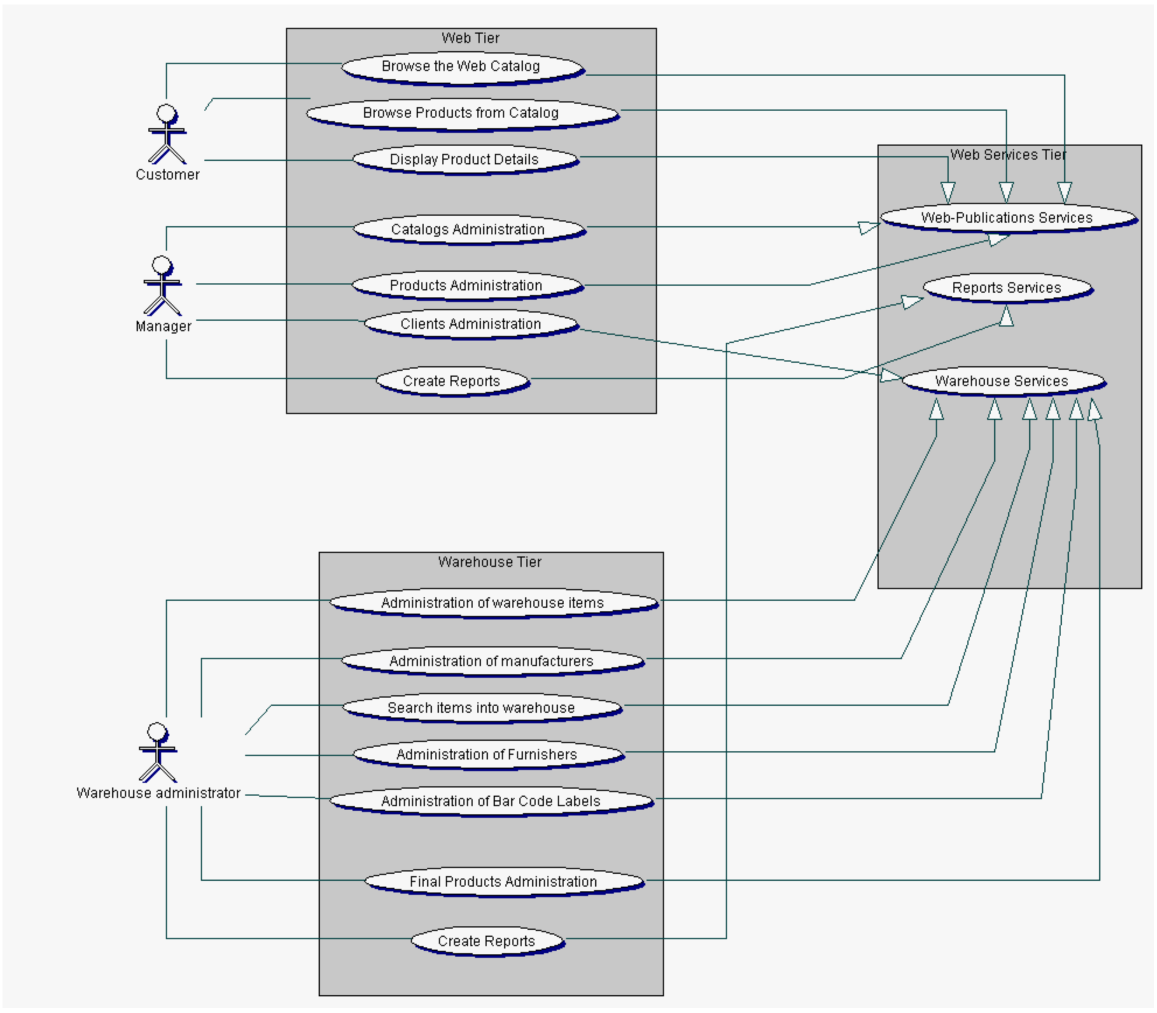


Table x.x

From this diagram we can see the main actions that has to be implemented in order to satisfy the users needs.

Sequence diagrams

Now lets see what's happening when a client that can be a web browser or any other application that can interpret the XML and SOAP functionalities, requests for a function [*Figure x.x*] from a Web Service and the response [*Figure x.x*] from the Web Services server, in this sequence diagrams you can see the interactions:

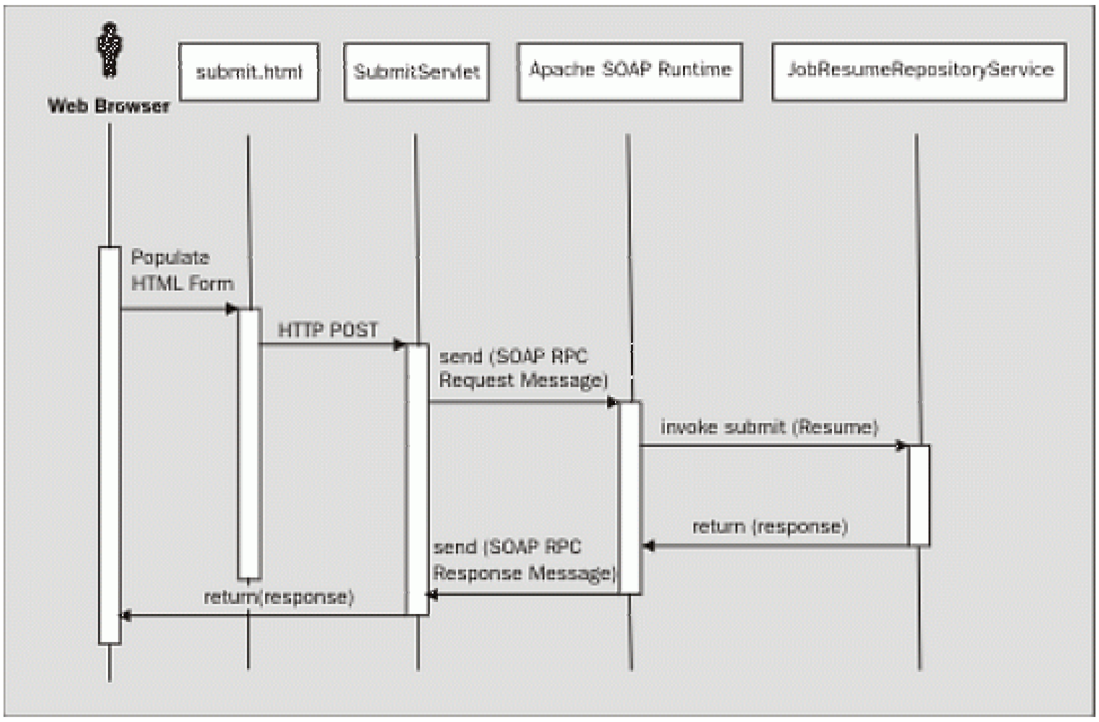

Web Browser
submit.html
SubmitServlet
Apache SOAP Runtime
JobResumeRepositoryService
Populate HTML Form
HTTP POST
send (SOAP RPC Request Message)
invoke submit (Resume)
return (response)
send (SOAP RPC Response Message)
return(response)

Figure x.x

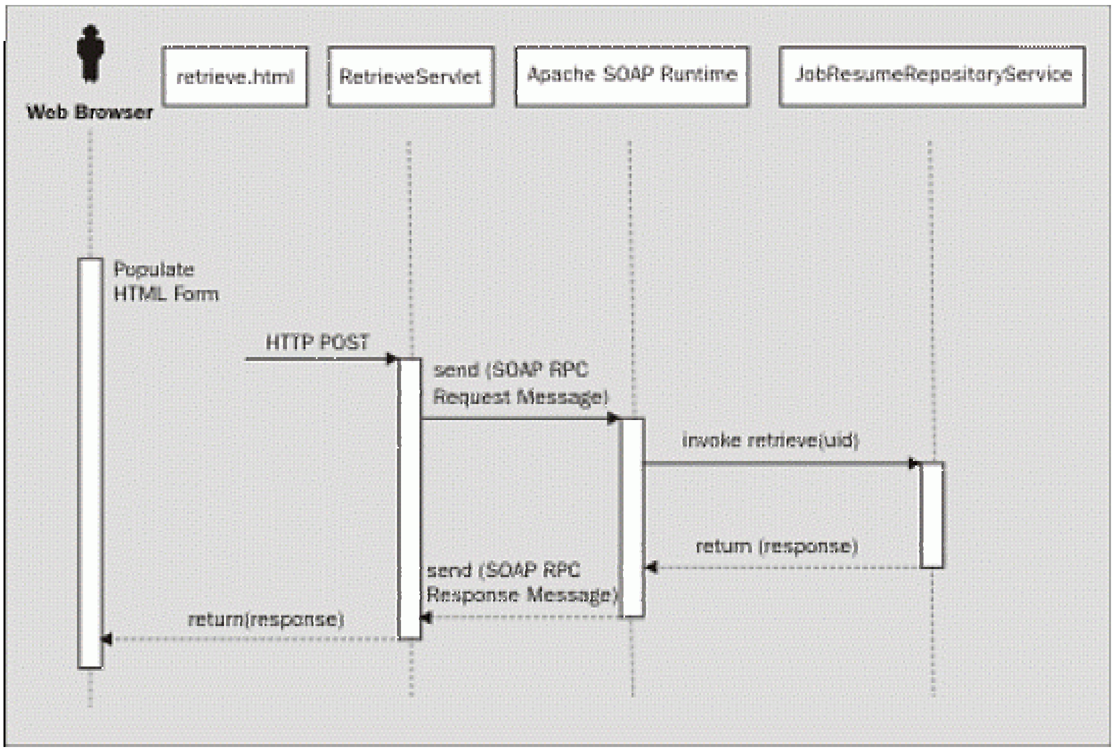

Web Browser
retrieve.html
RetrieveServlet
Apache SOAP Runtime
JobResumeRepositoryService
Populate HTML Form
HTTP POST
send (SOAP RPC Request Message)
invoke retrieve(uid)
return (response)
send (SOAP RPC Response Message)
return(response)

Figure x.x

3.2. System functionalities

In this chapter I'll describe the main functionalities grouped by the tree-tier layer design (client – web services app server – database server).

This table [*Table x.x*] represents the functionalities for **database server tier**:

Bean	Description	Properties
Magazie	Java class that represents database table "**Magazie**" - **load(ResultSer)** : Magazie – this function loads form the result set data into the bean - **CreateRecord(Connection, int)** : int – this function creates new record from data that is loaded into the bean with new primary key - **UpdateRecord(Connection)** : int – this function updates the record	**pk_magazie : int** nr_doc : int Index_nir : String data_receptie : java.util.Date Cantiate : float Prêt : float Valoare : float Dulap : String Raft : String Fk_produs : int Fk_furnizor : int
Produs	Java class that represents database table "**Produse**" - **load(ResultSer)** : Produs – this function loads form the result set data into the bean - **CreateRecord(Connection, int)** : int – this function creates new record from data that is loaded into the bean with new primary key - **UpdateRecord(Connection)** : int – this function updates the record	**pk_produs : int** nume : String cod_produs : String cod_bara_furnizor : String caracteristici_teh : byte[] stoc_minim : int Fk_clasaprodus : int Fk_producator : int
Producator	Java class that represents database table "**Producatori**" - **load(ResultSer)** : Producator – this function loads form the result set data into the bean - **CreateRecord(Connection, int)** : int – this function creates new record from data that is loaded into the bean with new primary key - **UpdateRecord(Connection)** : int – this function updates the record	**pk_producator : int** nume : String adresa : String oras : String tara : String telefon : String email : String url : String
ProdusFinal	Java class that represents database table "**ProduseFinale**" - **load(ResultSer)** : ProdusFinal – this function loads form the result set data into the bean - **CreateRecord(Connection, int)** : int – this function creates new record from data that is loaded into the bean with new primary key - **UpdateRecord(Connection)** : int – this function updates the record	**pk_produsfinal : int** nume : String data_finalizarii : java.util.Date cod_bara : String cod_bara_pic : byte[]
Inspectie	Java class that represents database table "**Inspectii**" - **load(ResultSer)** : Inspectie – this function loads form the result set data into the bean - **CreateRecord(Connection, int)** : int – this function creates new record from data that is loaded into the bean with new primary key - **UpdateRecord(Connection)** : int – this function updates the record	**pk_inspectie : int** data_expirarii : java.util.Date rezultatul_inspectiei : int Fk_magazie : int

Furnizor	Java class that represents database table "**Furnizori**" - **load(ResultSer)** : Furnizor – this function loads form the result set data into the bean - **CreateRecord(Connection, int)** : int – this function creates new record from data that is loaded into the bean with new primary key - **UpdateRecord(Connection)** : int – this function updates the record	**pk_furnizor : int** nume : String adresa : String oras : String tara : String telefon : String email : String url : String
CodBara	Java class that represents database table "**CodBare**" - **load(ResultSer)** : CodBara – this function loads form the result set data into the bean - **CreateRecord(Connection, int)** : int – this function creates new record from data that is loaded into the bean with new primary key - **UpdateRecord(Connection)** : int – this function updates the record	**pk_codbara : int** cod_bara : String cod_bara_pic : byte[] Fk_magazie : int
Client	Java class that represents database table "**Clieti**" - **load(ResultSer)** : Client – this function loads form the result set data into the bean - **CreateRecord(Connection, int)** : int – this function creates new record from data that is loaded into the bean with new primary key - **UpdateRecord(Connection)** : int – this function updates the record	**pk_client : int** nume : String adresa : String oras : String tara : String telefon : String email : String url : String
ClasaProdus	Java class that represents database table "**ClasaProduse**" - **load(ResultSer)** : ClasaProdus – this function loads form the result set data into the bean - **CreateRecord(Connection, int)** : int – this function creates new record from data that is loaded into the bean with new primary key - **UpdateRecord(Connection)** : int – this function updates the record	**pk_clasaprodus : int** nume_clasa : String cod_clasa : String
BonConsum	Java class that represents database table "**BonuriConsum**" - **load(ResultSer)** : BonConsum – this function loads form the result set data into the bean - **CreateRecord(Connection, int)** : int – this function creates new record from data that is loaded into the bean with new primary key - **UpdateRecord(Connection)** : int – this function updates the record	**pk_bonconsum : int** nr_doc : int data_eliberari : java.util.Date cantitate : float prêt : float valoare : float gestionar : String preluator : String Fk_magazie :int Fk_produsfinal : int

Table x.x

All this java beans represents tables from the database with functionalities for loading, creating and updating data.

The next tables [*Table x.x*] represents functionalities for **web services app server:**

Service name (magServ)	Description
getAllFromMagazie(int from,int to) : Magazie[]	Return from database table "Magazie" all records from point "from" to point "to" specified like arguments
getAllFromProduse(int from,int to) : Produse[]	Return from database table "Produse" all records from point "from" to point "to" specified like arguments
getAllFromProducatori(int from,int to) : Producator[]	Return from database table "Producatori" all records from point "from" to point "to" specified like arguments
getAllFromProduseFinale(int from,int to) : ProdusFinal[]	Return from database table "ProduseFinale" all records from

	point “from” to point “to” specified like arguments
getAllFromFurnizori(int from,int to) : Furnizor[]	Return from database table “Furnizori” all records from point “from” to point “to” specified like arguments
getAllFromClienti(int from,int to) : Client[]	Return from database table “clienti” all records from point “from” to point “to” specified like arguments
getAllFromClasaProduse(int from,int to) : ClasaProdus[]	Return from database table “ClasaProduse” all records from point “from” to point “to” specified like arguments
getAllFromBonuriConsum(int from,int to) : BonConsum[]	Return from database table “BonuriConsum” all records from point “from” to point “to” specified like arguments
createUpdateMagazie(Magazie arg) : int	Create or update record from database table. If the argument bean has no specified primary key then new record is created and if is specified then is update
createUpdateProdus(Produs arg) : int	- // -
createUpdateProducator(Producator arg) : int	- // -
createUpdateProdusFinal(ProdusFinal arg) : int	- // -
createUpdateFurnizor(Furnizor arg) : int	- // -
createUpdateClient(Client arg) : int	- // -
createUpdateClasaProdus(ClasaProdus arg) : int	- // -
createUpdateInspectie(Inspectie arg) : int	- // -
createUpdateCodBara(CodBara arg) : int	- // -
createUpdateBonConsum(BonConsum arg) : int	- // -
deleteMagazie(int pk) : Boolean	Delete from database table by primary key and return true if the is deleted with success
deleteProducator(int pk) : Boolean	- // -
deleteProdusFinal(int pk) : Boolean	- // -
deleteFurnizor(int pk) : Boolean	- // -
deleteClient(int pk) : Boolean	- // -
deleteInspectie(int pk) : Boolean	- // -
deleteCodBara(int pk) : Boolean	- // -
deleteClasaProdus(int pk) : Boolean	- // -
deleteBonConsum(int pk) : Boolean	- // -
deleteProdus(int pk) : Boolean	- // -

Service name (reportServ)	Description
createFisaDeMagazie(int pk_produs) : byte[]	Create detailed report that contains quantities of the specified product that has bean entered and left the warehouse and the current stock of that item
createBonDeConsumPartial(int pk_produsfinal,int nr_bon) : byte[]	Create detailed report that contains the list of items that has bean left the warehouse for a final product with registered number
createBonDeConsumTotal(int pk_produsfinal) : byte[]	Create detailed report that contains the list of items that has bean left the warehouse for a final product

Service name (webServ)	Description
getAllFromCatalog() : Catalog[]	Return list of all web catalogs
getAllProductsFromCatalog(int pk_catalog) : Product[]	Return list of products form some catalog
getProductDetails(int pk_product) : Product	Return product details
getAllClients() : Client[]	Return list of clients
createUpdateCatalog(Catalog arg) : int	Create or update the catalog
createUpdateProduct(Product arg) : int	Create or update the product
createUpdateClient(Client arg) : int	Create or update the client
deleteCatalog(int pk_catalog) : boolean	Delete catalog and return true if succeeded
deleteClient(int pk_client) : boolean	Delete client and return true if succeeded
deleteProduct(int pk_product) : boolean	Delete product and return true if succeeded

All this services are public for all clients that deserve and use these functionalities trough the axis web services engine. To see how to deploy the web service on Apache Tomcat see the appendix [Appendix A]

The next table represents the functionalities for client – tier:

Warehouse Administration

Interfaces name	Description
MagServ **Classes** / **Description** MagazieBean / Data from table "Magazie" ProdusBean / Data from table "Produse" ProducatorBean / Data from table "Producatori" ProdusFinalBean / Data from table "ProdusFinal" InspectieBean / Data from table "Inspectii" FurnizorBean / Data from table "Furnizori" CodBaraBean / Data from table "CodBare" ClientBean / Data from table "Clienti" ClasaProdusBean / Data from table "ClasaProduse" BonConsumBean / Data from table "BonConsum"	*This interface contains all the functions and java beans serialized and de-serialized used for this web service and are described in a WSDL (Web Service Description Language) that is published to Web Services server*.
ReportServ - CreateWarehouseInventory(int) : byte[] - ItemTracking(int) : byte[] - GetFinalProductsItems(int) : byte[]	*This interface contains all the functions and java beans serialized and de-serialized used for this web service and are described in a WSDL (Web Service Description Language) that is published to Web Services server.*

Table x.x

Marketing and Management

Interfaces name	Description
WebAppServ **Classes** / **Description** ClientBean / Data from table "Clienti" CatalogBean / Data from table "Catalog" WebProduseBean / Data from table "Web_Produse"	*This interface contains all the functions and java beans serialized and de-serialized used for this web service and are described in a WSDL (Web Service Description Language) that is published to Web Services server.*
ReportServ - CreateWarehouseInventory(int) : byte[] - ItemTracking(int) : byte[] - GetFinalProductsItems(int) : byte[]	*This interface contains all the functions and java beans serialized and de-serialized used for this web service and are described in a WSDL (Web Service Description Language) that is published to Web Services server.*

Table x.x

3.3 User Interfaces

The user interfaces are categorized in:

a) Warehouse Visual Basic Client

b) Web Client Interfaces

a). Warehouse Visual Basic Client

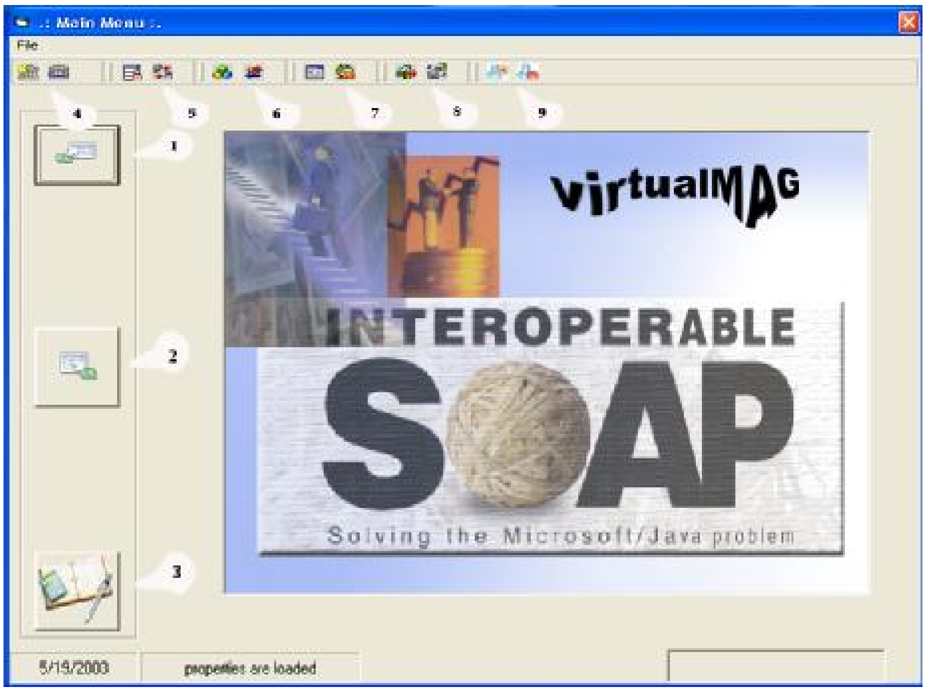


Figure x.x

This form represents the Main Menu for administration of warehouse program designed in visual basic language. From here we can access all other functionalities related to warehouse administration using:

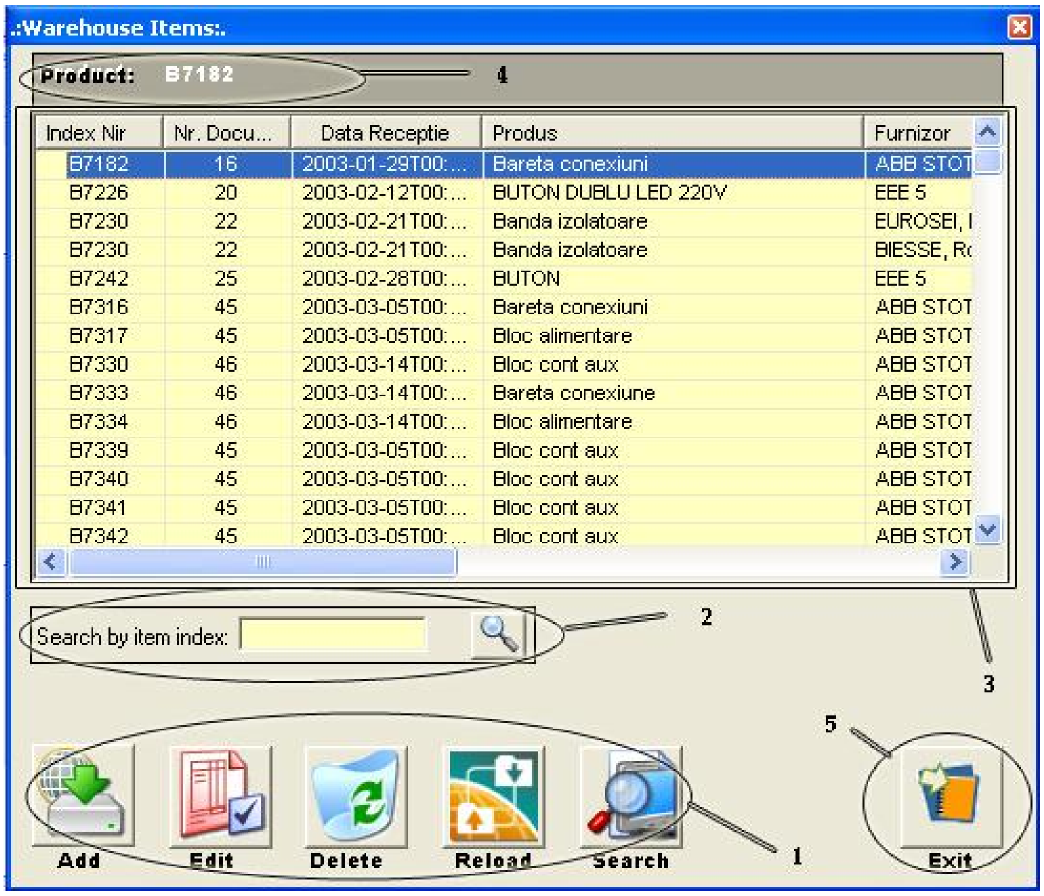
Fast access buttons:

1. Open "Warehouse Items" dialog (*Figure x.x*)
2. Open "Consumed Items" dialog
3. Open "Reports" dialog

Toolbar:

4. Group of two buttons, first adds new item into warehouse and second displays existing items for editing or deleting (*Figure x.x, Figure x.x*)
5. Group of two buttons, first adds new furnisher and second displays existing one for editing or deleting (*Figure x.x)*

6. Group of two buttons, first adds new item into database and second displays existing one for editing and deleting
7. Group of two buttons, first adds new item class into database and second displays existing one for editing and deleting
8. Group of two buttons, first adds new final product into database and second displays existing one for editing and deleting
9. Group of two buttons, first adds new client into database and second displays existing one for editing and deleting



Product: B7182

Index Nir	Nr. Docu...	Data Receptie	Produs	Furnizor
B7182	16	2003-01-29T00:...	Bareta conexiuni	ABB STOT
B7226	20	2003-02-12T00:...	BUTON DUBLU LED 220V	EEE 5
B7230	22	2003-02-21T00:...	Banda izolatoare	EUROSEI, I
B7230	22	2003-02-21T00:...	Banda izolatoare	BIESSE, Rc
B7242	25	2003-02-28T00:...	BUTON	EEE 5
B7316	45	2003-03-05T00:...	Bareta conexiuni	ABB STOT
B7317	45	2003-03-05T00:...	Bloc alimentare	ABB STOT
B7330	46	2003-03-14T00:...	Bloc cont aux	ABB STOT
B7333	46	2003-03-14T00:...	Bareta conexiune	ABB STOT
B7334	46	2003-03-14T00:...	Bloc alimentare	ABB STOT
B7339	45	2003-03-05T00:...	Bloc cont aux	ABB STOT
B7340	45	2003-03-05T00:...	Bloc cont aux	ABB STOT
B7341	45	2003-03-05T00:...	Bloc cont aux	ABB STOT
B7342	45	2003-03-05T00:...	Bloc cont aux	ABB STOT

Figure. 2

This form contains items from warehouse with following controls:

1. *Controls for adding, editing, deleting, reloading and searching items*
2. *Fast search from list*
3. *List that contains items from warehouse*
4. *Selected item index*
5. *Exit from this form*

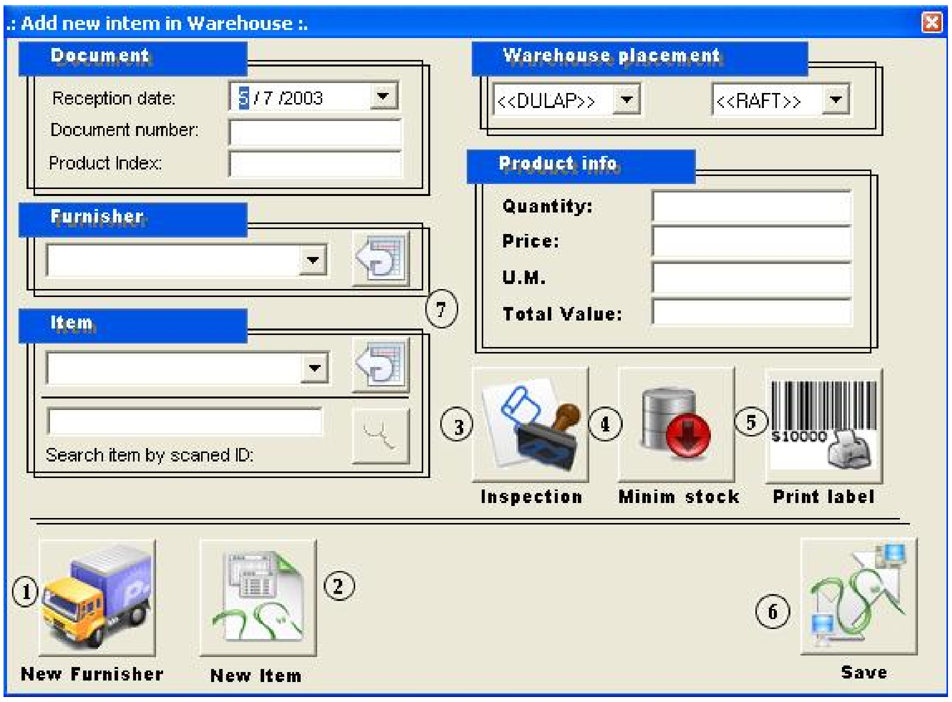


Figure x.x

This form adds new item into warehouse that contains data about document, furnisher, item, and warehouse placement, product info.

1. *Add new furnisher (Fig.4)*
2. *Add new item into database (Fig.5)*
3. *Set item inspection (Fig.6)*
4. *Set items minim stock (Fig.7)*
5. *Print bar code label (Fig.8)*
6. *Save new item into warehouse database*
7. *Fast search button*

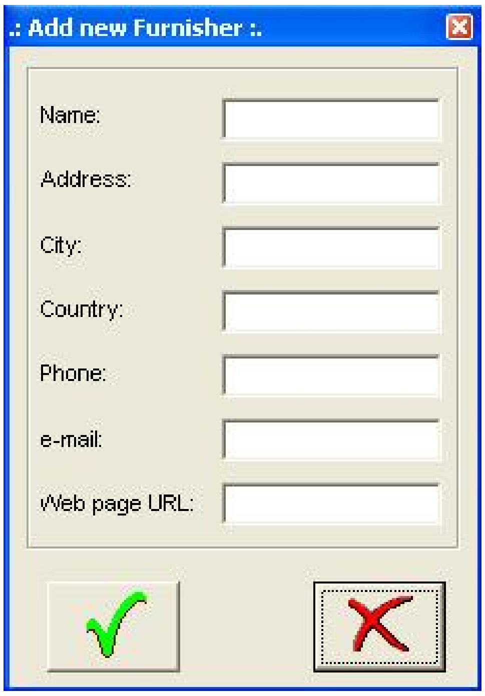


Figure x.x

This form adds new furnisher into database.

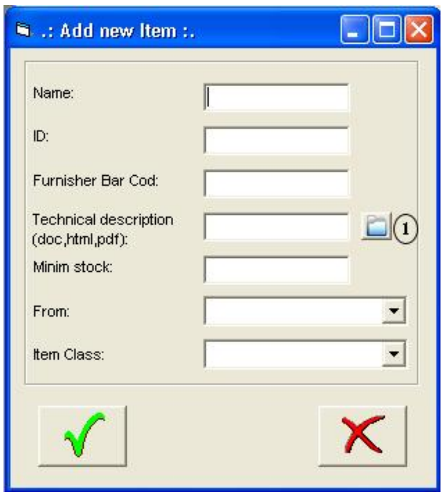
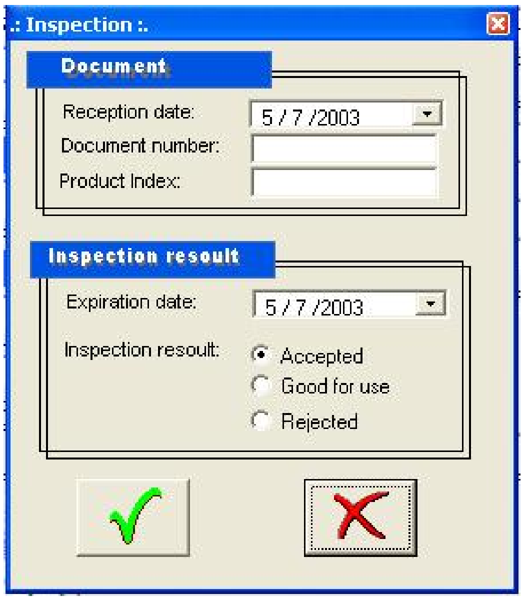


Figure x.x

This form adds new item into database.



.: Inspection :.

Document

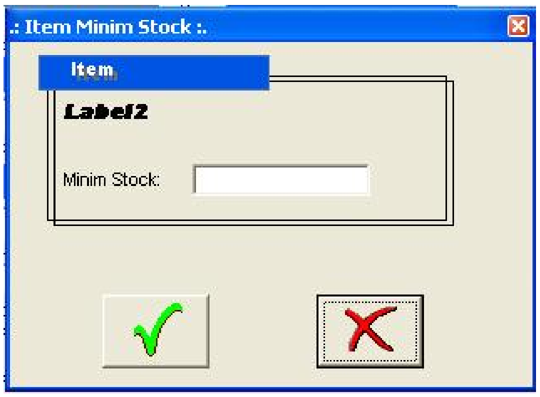
Reception date:	5 / 7 /2003
Document number:	
Product Index:	

Inspection resoult

Expiration date:	5 / 7 /2003
Inspection resoult:	[✓ Accepted] Good for use Rejected

Fig.6 -

This form sets inspection for item added in warehouse.



.: Item Minim Stock :.

Item

Label2

Minim Stock:

Fig.7 – Item Minim

This form sets item minim stock in warehouse.

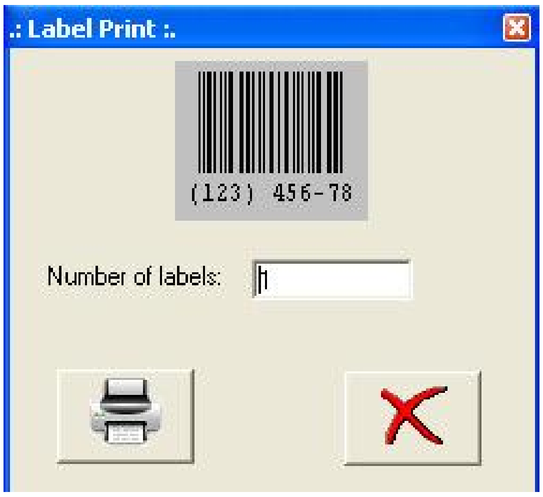
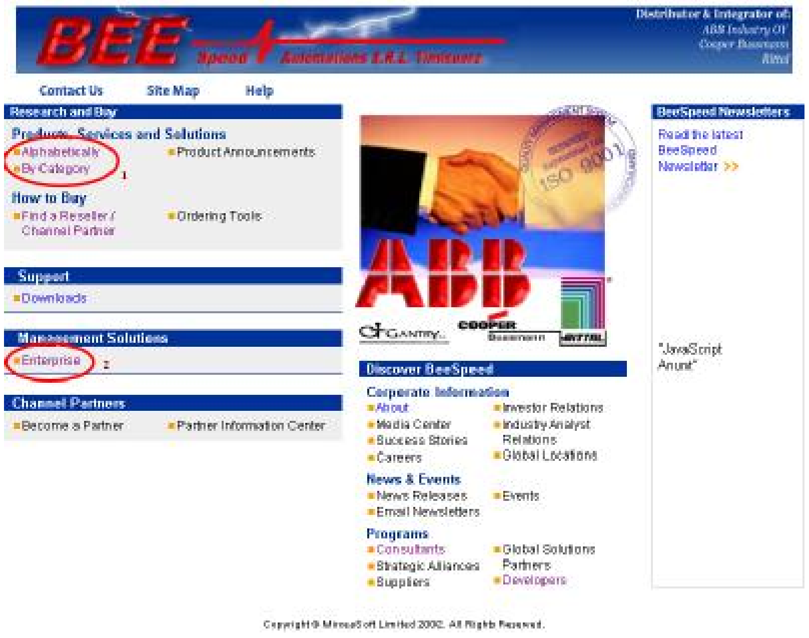


Fig.8 – Label

This form prints bar code label for item that has been introduced into warehouse.

b). Web Client interfaces


BEE Speed Automations S.R.L. Timisoara

Distributor & Integrator of:
ABB Industry OY
Cooper Bussmann
Rittal

Contact Us Site Map Help

Research and Buy

Products, Services and Solutions
- Alphabetically
- By Category
- Product Announcements

1

How to Buy
- Find a Reseller / Channel Partner
- Ordering Tools

Support
- Downloads

Management Solutions
- Enterprise

2

Channel Partners
- Become a Partner
- Partner Information Center

Discover BeeSpeed

Corporate Information
- About
- Media Center
- Success Stories
- Careers
- Investor Relations
- Industry Analyst Relations
- Global Locations

News & Events
- News Releases
- Email Newsletters
- Events

Programs
- Consultants
- Strategic Alliances
- Suppliers
- Global Solutions Partners
- Developers

BeeSpeed Newsletters

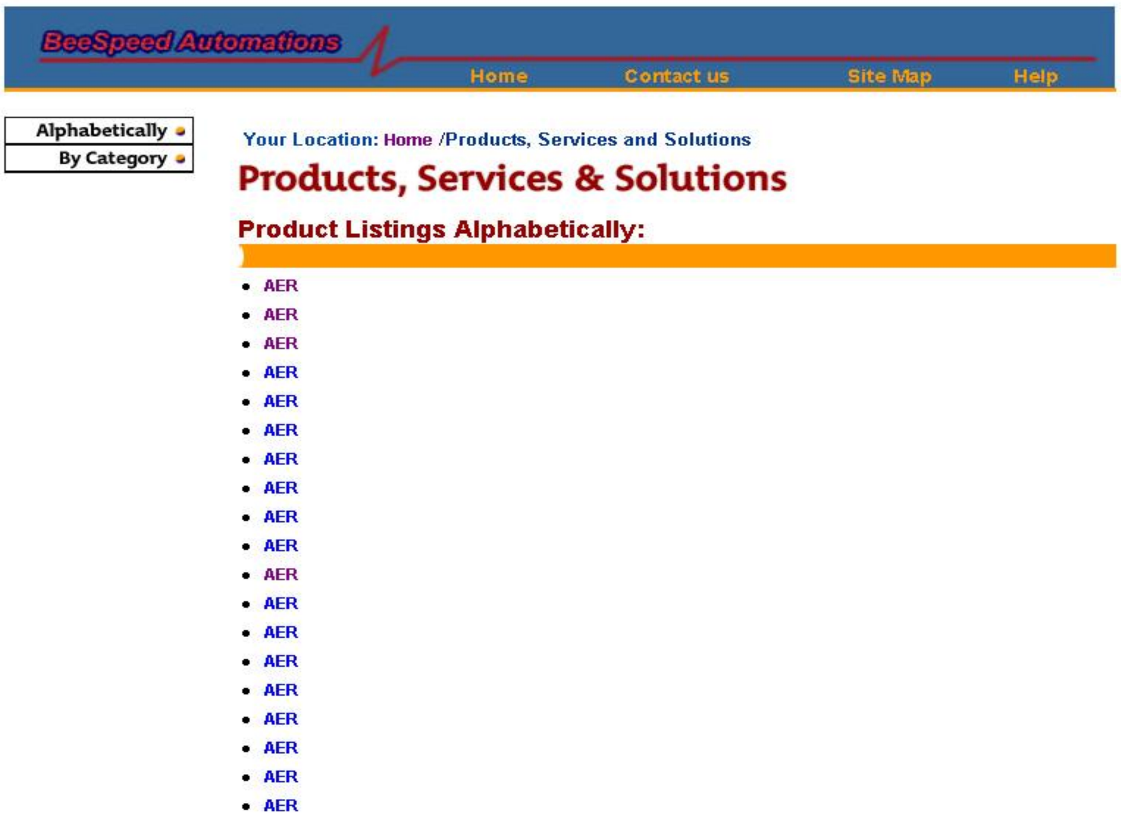
Read the latest BeeSpeed Newsletter >>

"JavaScript Anunt"

Copyright © MirceaSoft Limited 2002. All Rights Reserved.

Fig.9 – Web main page

This page represents main page for web client. From here you can see two links for clients in this project that access web catalog of the final products, displayed alphabetically or by category, and a link that is used for management solutions.



Your Location: Home /Products, Services and Solutions

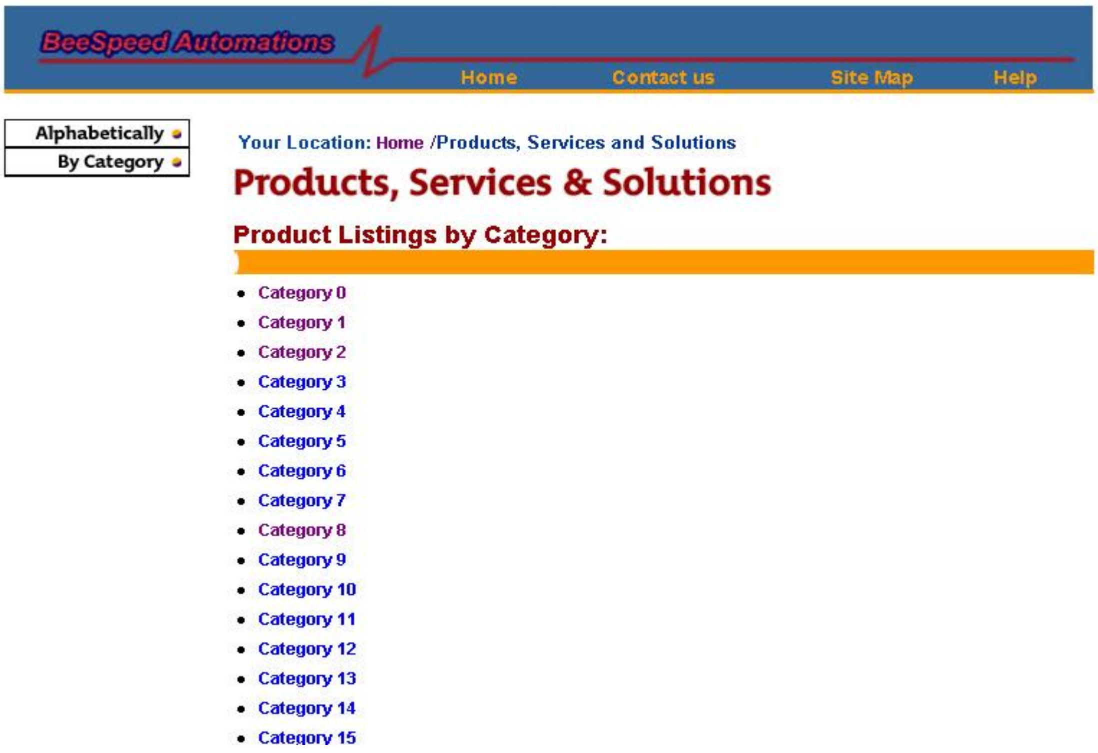
Products, Services & Solutions

Product Listings Alphabetically:

- AER
- AER
- AER
- AER
- AER
- AER
- AER
- AER
- AER
- AER
- AER
- AER
- AER
- AER
- AER
- AER
- AER
- AER
- AER

Fig.10 Products displayed

This page displays products ordered alphabetically. If select some of the product from list the product details is displayed (*Fig.13*).



Your Location: Home /Products, Services and Solutions

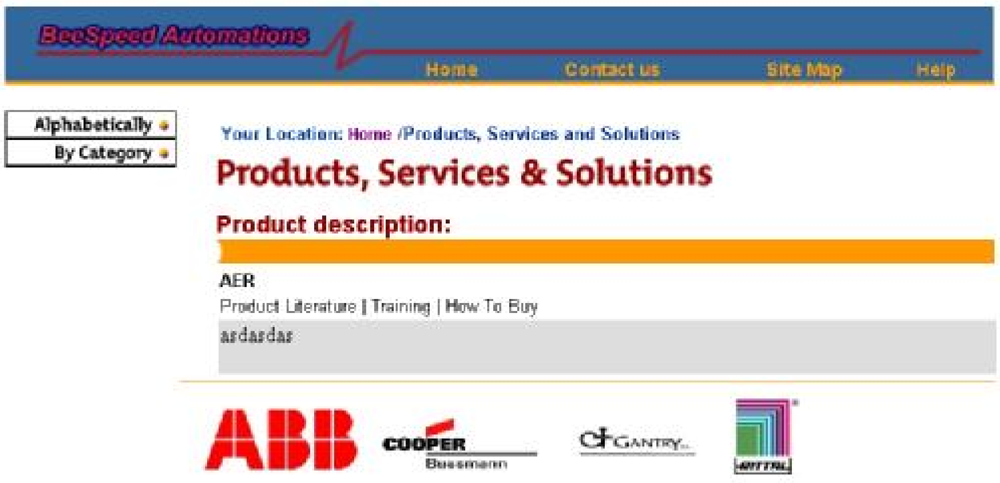
Products, Services & Solutions

Product Listings by Category:

- Category 0
- Category 1
- Category 2
- Category 3
- Category 4
- Category 5
- Category 6
- Category 7
- Category 8
- Category 9
- Category 10
- Category 11
- Category 12
- Category 13
- Category 14
- Category 15

Fig.11 – Products displayed by

This page displays categories of products. If some of the category is selected from list the products from that category are displayed.



Your Location: Home /Products, Services and Solutions

Products, Services & Solutions

Product description:

AER

Product Literature | Training | How To Buy

asdasdas

Fig.12 – Product

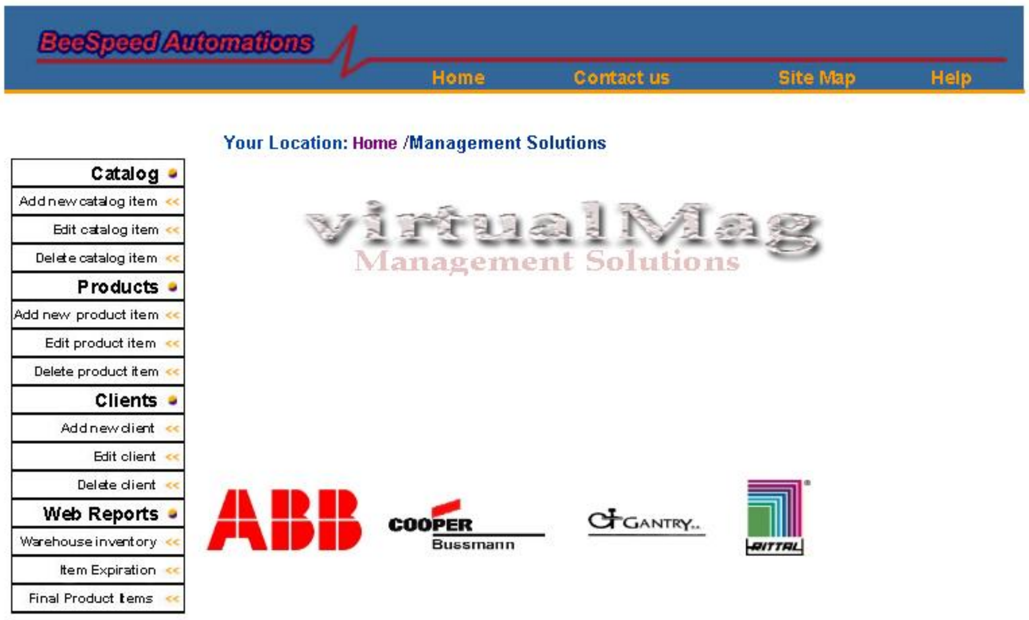


Fig.13 – Management Solution

This page represents main page for management solutions. From here the manager can administrate catalogs, products and clients. Also he can create reports in different formats like HTML, PDF or XLS.

3.4 Database structure

Like a database server we have used the MySQL[***1995,c] because is free and is very efficient for our needs. We need a database where we can store the data for warehouse administration and for marketing and management needs.

In the next figure you can see the database structure and relations between the tables.

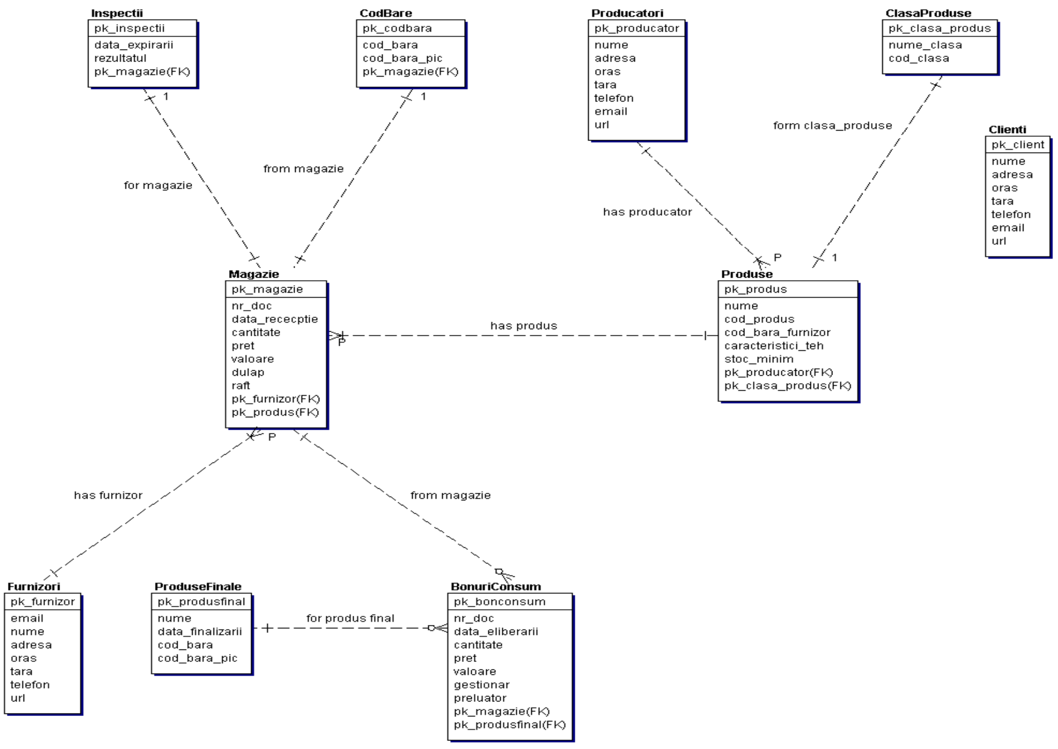


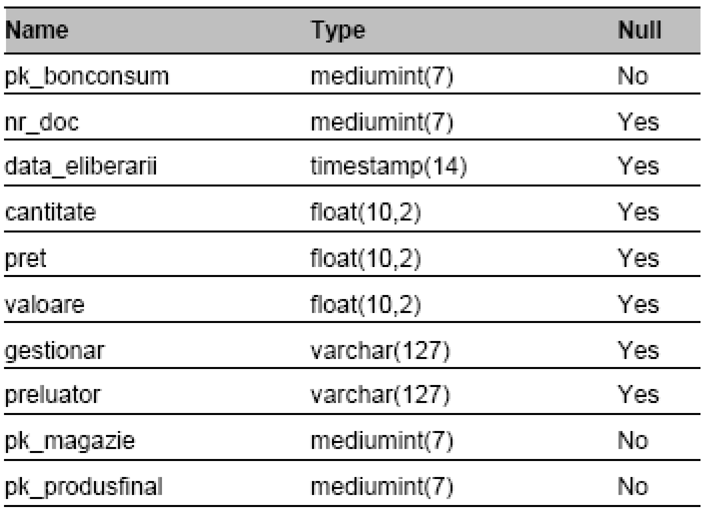
Figure x.x

Table list

Table	Records	Size	Created	Type	Comment
bonuriconsum	0	2 KB	6/12/2003 4:33:44 PM	MyISAM	
clasaproduse	1	3 KB	6/12/2003 4:33:44 PM	MyISAM	
clienti	2	3 KB	6/12/2003 4:33:44 PM	MyISAM	
codbare	1	3 KB	6/12/2003 4:33:44 PM	MyISAM	
furnizori	3	3 KB	6/12/2003 4:33:44 PM	MyISAM	
inspectii	1	3 KB	6/12/2003 4:33:44 PM	MyISAM	
magazie	4	3 KB	6/12/2003 4:33:44 PM	MyISAM	
producatori	1	3 KB	6/12/2003 4:33:44 PM	MyISAM	
produse	1	3 KB	6/12/2003 4:33:44 PM	MyISAM	
produsefinale	0	2 KB	6/12/2003 4:33:45 PM	MyISAM	

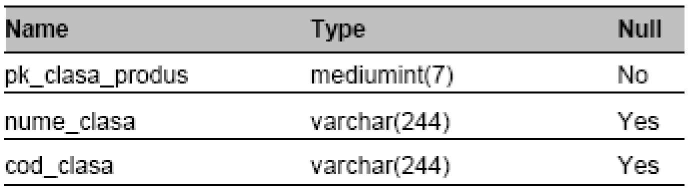
Table's description:

BonuriConsum



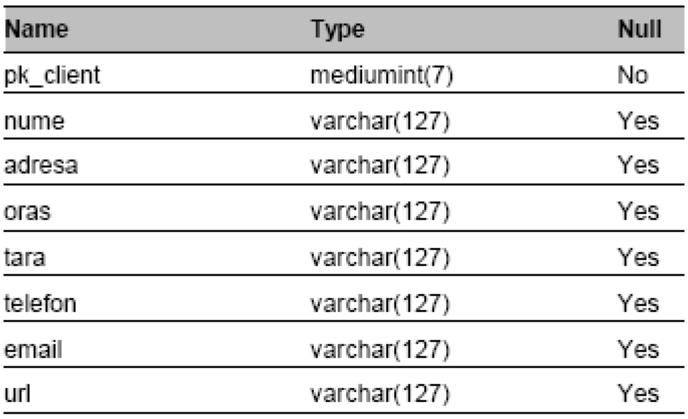
Name	Type	Null
pk_bonconsum	mediumint(7)	No
nr_doc	mediumint(7)	Yes
data_eliberarii	timestamp(14)	Yes
cantitate	float(10,2)	Yes
pret	float(10,2)	Yes
valoare	float(10,2)	Yes
gestionar	varchar(127)	Yes
preluator	varchar(127)	Yes
pk_magazie	mediumint(7)	No
pk_produsfinal	mediumint(7)	No

ClasaProduse



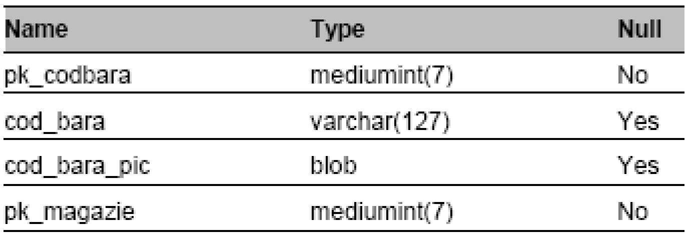
Name	Type	Null
pk_clasa_produs	mediumint(7)	No
nume_clasa	varchar(244)	Yes
cod_clasa	varchar(244)	Yes

Clienti



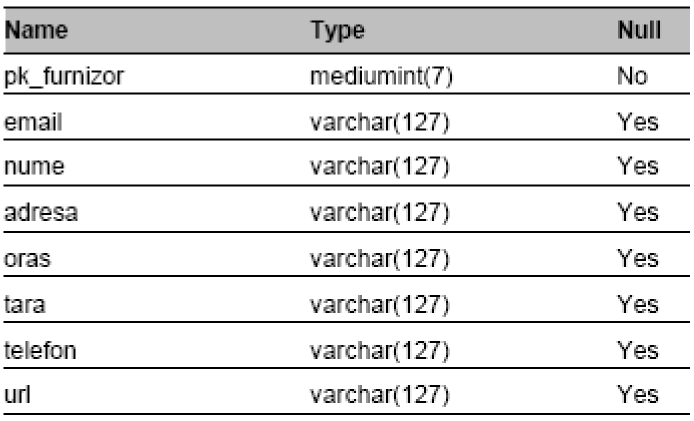
Name	Type	Null
pk_client	mediumint(7)	No
nume	varchar(127)	Yes
adresa	varchar(127)	Yes
oras	varchar(127)	Yes
tara	varchar(127)	Yes
telefon	varchar(127)	Yes
email	varchar(127)	Yes
url	varchar(127)	Yes

CodBare



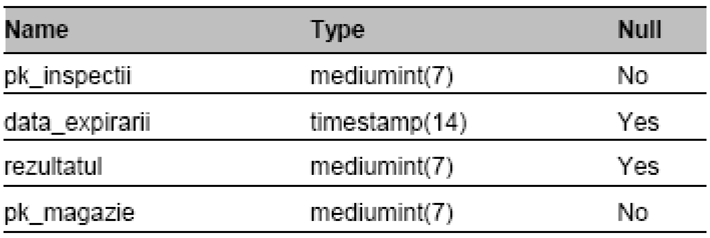
Name	Type	Null
pk_codbara	mediumint(7)	No
cod_bara	varchar(127)	Yes
cod_bara_pic	blob	Yes
pk_magazie	mediumint(7)	No

Furnizori



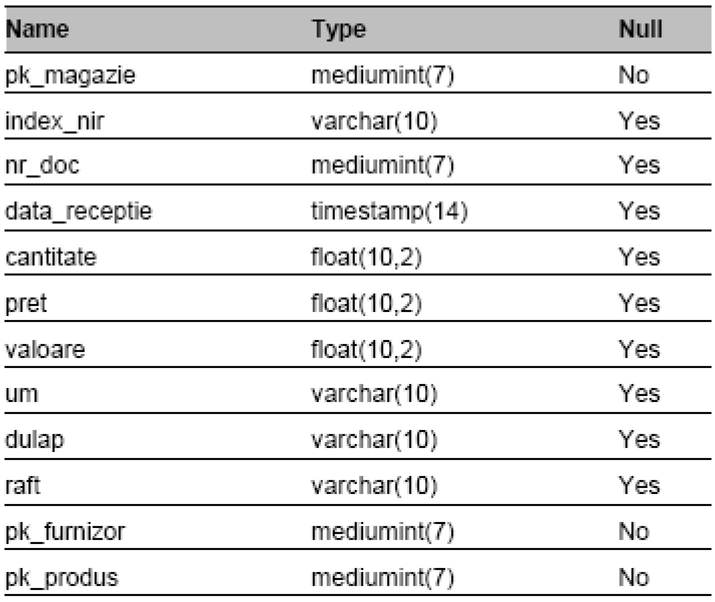
Name	Type	Null
pk_furnizor	mediumint(7)	No
email	varchar(127)	Yes
nume	varchar(127)	Yes
adresa	varchar(127)	Yes
oras	varchar(127)	Yes
tara	varchar(127)	Yes
telefon	varchar(127)	Yes
url	varchar(127)	Yes

Inspectii



Name	Type	Null
pk_inspectii	mediumint(7)	No
data_expirarii	timestamp(14)	Yes
rezultatul	mediumint(7)	Yes
pk_magazie	mediumint(7)	No

Magazie



Name	Type	Null
pk_magazie	mediumint(7)	No
index_nir	varchar(10)	Yes
nr_doc	mediumint(7)	Yes
data_receptie	timestamp(14)	Yes
cantitate	float(10,2)	Yes
pret	float(10,2)	Yes
valoare	float(10,2)	Yes
um	varchar(10)	Yes
dulap	varchar(10)	Yes
raft	varchar(10)	Yes
pk_furnizor	mediumint(7)	No
pk_produs	mediumint(7)	No

Producatori

Name	Type	Null
pk_producator	mediumint(7)	No
nume	varchar(127)	Yes
adresa	varchar(127)	Yes
oras	varchar(127)	Yes
tara	varchar(127)	Yes
telefon	varchar(127)	Yes
email	varchar(127)	Yes
url	varchar(127)	Yes

Produse

Name	Type	Null
pk_produs	mediumint(7)	No
nume	varchar(244)	Yes
cod_produs	varchar(244)	Yes
cod_bara_furnizor	varchar(244)	Yes
caracteristici_teh	blob	Yes
stoc_minim	mediumint(7)	Yes
pk_producator	mediumint(7)	No
pk_clasa_produs	mediumint(7)	No

ProduseFinale

Name	Type	Null
pk_produsfinal	mediumint(7)	No
nume	varchar(127)	Yes
data_finalizarii	timestamp(14)	Yes
cod_bara	varchar(127)	Yes
cod_bara_pic	blob	Yes

3.5 System communication

In our system we have two types of communications between modules and those are Http communication and object communications.
Communication between the system modules (tiers) is categorized by:

a) SOAP communication
b) Database communication
c) Printer communication

a) SOAP communication

What is SOAP?

SOAP is an XML-based communication protocol and encoding format for inter-application communication. Originally conceived by Microsoft and Userland software, it has evolved through several generations and the current spec, SOAP 1.1, is fast growing in popularity and usage. The W3C's XML Protocol working group [***2000,b] is in the process of turning SOAP into a true open standard, and as of this writing has released a working draft of SOAP 1.2, which cleans up some of the more confusing areas of the 1.1 spec.

SOAP is widely viewed as the backbone to a new generation of cross-platform cross-language distributed computing applications, termed Web Services.

What is Axis?

Axis is essentially a *SOAP engine* – a framework for constructing SOAP processors such as clients, servers, gateways, etc. The current version of Axis is written in Java, but a C++ implementation of the client side of Axis is being developed.

But Axis isn't just a SOAP engine – it also includes:

- a simple stand-alone server,
- a server which plugs into servlet engines such as Tomcat,
- extensive support for the *Web Service Description Language (WSDL)*,

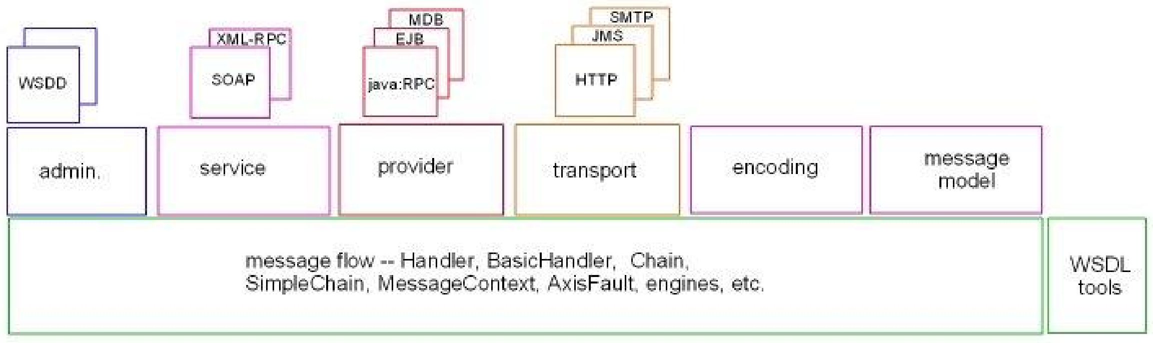
- emitter tooling that generates Java classes from WSDL.
- some sample programs, and
- a tool for monitoring TCP/IP packets.

Axis is the third generation of Apache SOAP (which began at IBM as “SOAP4J”). In late 2000, the committers of Apache SOAP v2 began discussing how to make the engine much more flexible, configurable, and able to handle both SOAP and the upcoming XML Protocol specification from the W3C.

After a little while, it became clear that a ground-up re-architecture was required. Several of the v2 committers proposed very similar designs, all based around configurable “chains” of message “handlers” which would implement small bits of functionality in a very flexible and compassable manner.

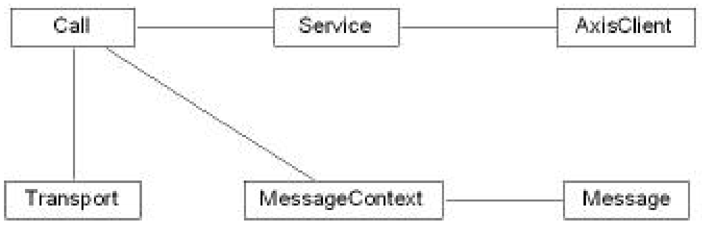
Axis comprises several subsystems working together with the aim of separating responsibilities cleanly and making Axis modular. Subsystems, which are properly layered, enable parts of a system to be used without having to use the whole of it (or hack the code).

The following diagram shows the layering of subsystems. The lower layers are independent of the higher layers. The ‘stacked’ boxes represent mutually independent, although not necessary mutually exclusive, alternatives. For example, the HTTP, SMTP, and JMS transports are independent of each other but may be used together.



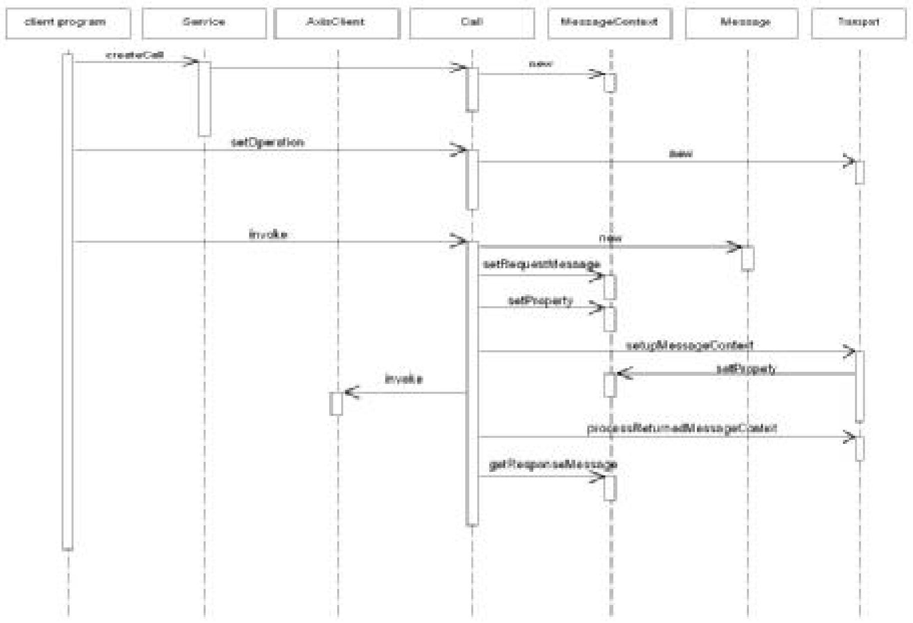
Client Side Processing

The client side Axis processing constructs a Call object with associated Service, MessageContext, and request Message as shown below before invoking the AxisClient engine.



An instance of Service and its related AxisClient instance are created before the Call object. Invoking the Service.createCall factory method then creates the Call object. Call.setOperation creates a Transport instance, if a suitable one is not already associated with the Call instance. Then Call.invoke creates a MessageContext and associated request Message, drives AxisClient.invoke, and processes the resultant MessageContext.

This significant method calls in this sequence are shown in the following interaction diagram.



b) Database connections

Database connection is realized using java mysql driver and represents the low level communication between the java virtual machine and mysql server trough http sockets. For this communication I don't have to say nothing more because is so popular and is not the issue of discussion.

c) Printer communication

This communication is done trough parallel port LPT 1 the printer port and for this application that administrate the warehouse we'll use the specialized bar code printer from Intermec EasyCoder C4.

The EasyCoder C4 printers from Intermec are provided with a built-in protocol (Esim) by which you can use any computer, terminal, scanner or keyboard that can produce ASCII characters, to control the printer. This is a useful alternative to the Intermec InterDriver, which requires a PC operating under Microsoft Windows. With the Esim protocol, you can use any editor to control the printer, either by means of the serial RS-232 channel or the parallel Centronics channel.

3.6 Type of reports

The warehouse administrator and the marketing and management manager generate the reports. Reports are data from the database represented and selected in different forms and formats. Using the Jasper Reports we can display and print the report in a PDF, HTML and XLS file format.

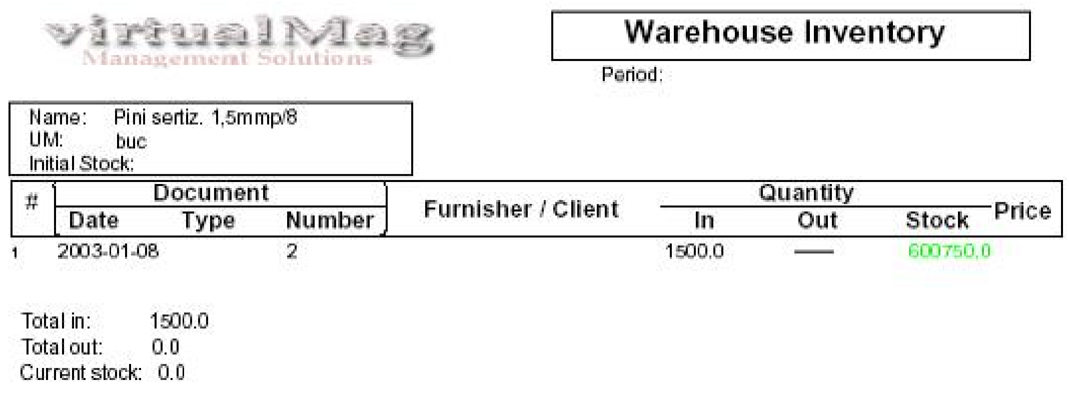
For this application we'll be able to create tree types of reports:

1. Warehouse inventory
2. Item tracking
3. Final product items

1. Warehouse inventory

This type of report represents the inventory for a particular item so we'll see the amount of items that has been entered into warehouse and the amount of items that has been left the warehouse and the total items that are currently in the warehouse on the stack.

In the next figure [*Figure x.x*] you'll see an example of this type of report in PDF file format.



Warehouse Inventory
Period:

Name: Pini sertiz. 1,5mmp/8
UM: buc
Initial Stock:

#	Document			Furnisher / Client	Quantity			Price
	Date	Type	Number		In	Out	Stock	
1	2003-01-08		2		1500.0	—	600750.0	

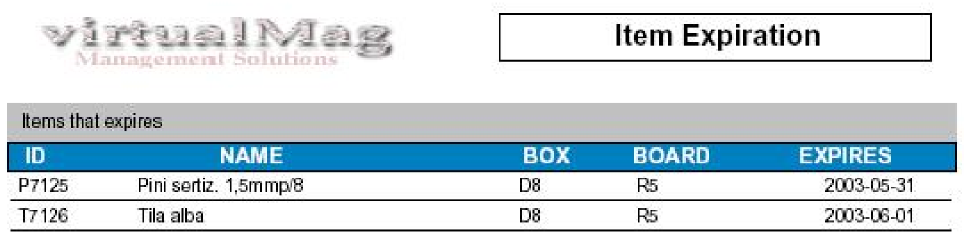
Total in: 1500.0
Total out: 0.0
Current stock: 0.0

Figure x.x

2. Item tracking

This type of report represents the tracking items that has been expired or expires in a near period.

In the next figure [*Figure x.x*] you'll see an example of this type of report in PDF file format.



Item Expiration

Items that expires

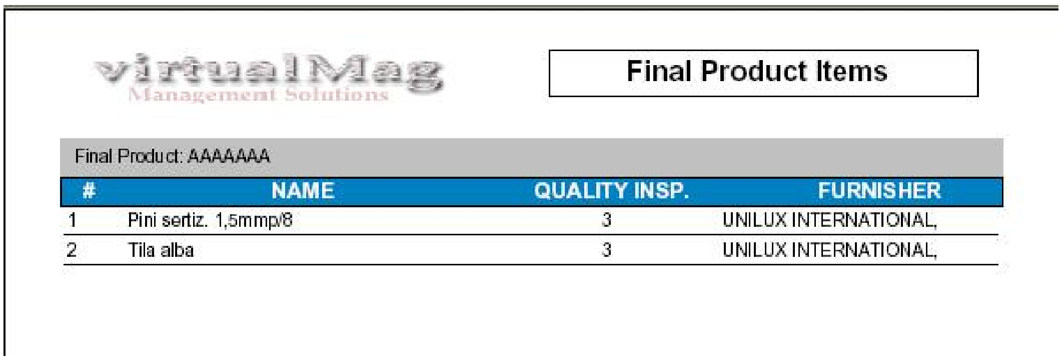
ID	NAME	BOX	BOARD	EXPIRES
P7125	Pini sertiz. 1,5mmp/8	D8	R5	2003-05-31
T7126	Tila alba	D8	R5	2003-06-01

Figure x.x

3. Final product Items

This type of report represents the tracking items that has been used for a particular final product or has bean sold to the client.

In the next figure [*Figure x.x*] you'll see an example of this type of report in PDF file format.


Final Product Items

Final Product: AAAAAAA

#	NAME	QUALITY INSP.	FURNISHER
1	Pini sertiz. 1,5mmp/8	3	UNILUX INTERNATIONAL,
2	Tila alba	3	UNILUX INTERNATIONAL,

Figure x.x

3.7 Security and risk analysis

A standard attack on a web site is usually that of identifying and abusing badly written CGI scripts. Anything that gives read access to the file system is a security hole, letting people get at the code behind the site, often including database passwords and other sensitive data, plus of course there are the core parts of the underlying platform, which may contain important information: passwords, credit card lists, user-private information, and the like. Unauthorized access to this data can be embarrassing and expensive.

Having write access to the system leads to even greater abuses; defaced web sites may be created, spoof endpoints written to capture caller's data, or the database directly manipulated.

XML messages have a few intrinsic weaknesses that Web Service creators should know about. None of these problems are unique to SOAP; anyone processing incoming XML needs to know and resist these.

1. Large XML Docukents
   . Have a client post an XML doc of extreme length/depth <ha><ha><ha>....</ha></ha></ha>. This does bad things to DOM parsers and memory consumption on the server: a DoS attack. The issue here is that the costs of handling a large XML document are much greater than the cost of generating one.
2. Entity Expansion Attacks. If an XML doc header declares some recursive entity declarations, and the file refers to them, then bad things happen. Axis became immune to this between versions 1.0 and 1.1.
3. Entities referring to the file system.
   Here you declare an entity referring to a local file, then expand it. Result: you may be able to probe for files, perhaps even get a copy of it in the error response. As Axis does not support entities any more, it resists this. If your code has any way of resolving URLs from incoming messages, you may recreate this problem.

The other thing to know about XML is that string matching is not enough to be sure that the content is safe, because of the many ways to reformat the same XML.

A core philosophy is ‘defend in depth’, with monitoring for trouble.

Disguise

One tactic here is to hide the fact that you are running Axis...look at all the headers that we send back to describe the service, and if any identify Axis, edit that constant in the source. While obscurity on its own is inadequate; it can slow down attacks or make you seem less vulnerable to known holes.

Cut down the build

Rebuild Axis without bits of it you • irce need. This is a very paranoid solution, but keeps the number of potential attack points down. One area to consider is the 'instant SOAP service' feature of JWS pages. They, along with JSP pages, provide anyone who can get text files onto the web application with the ability to run arbitrary Java code.

Rename things

The AxisServlet, the AdminService, even happyaxis.jsp are all in well known locations under the webapp, which is called 'axis' by default. Rename all of these, by editing web.xml for the servlet, server-config.wsdd for the AdminService; the others are just JSP and WAR files you can rename. You may not need the AdminService once you have generated the server config on a development machine.

Stop AxisServlet listing services

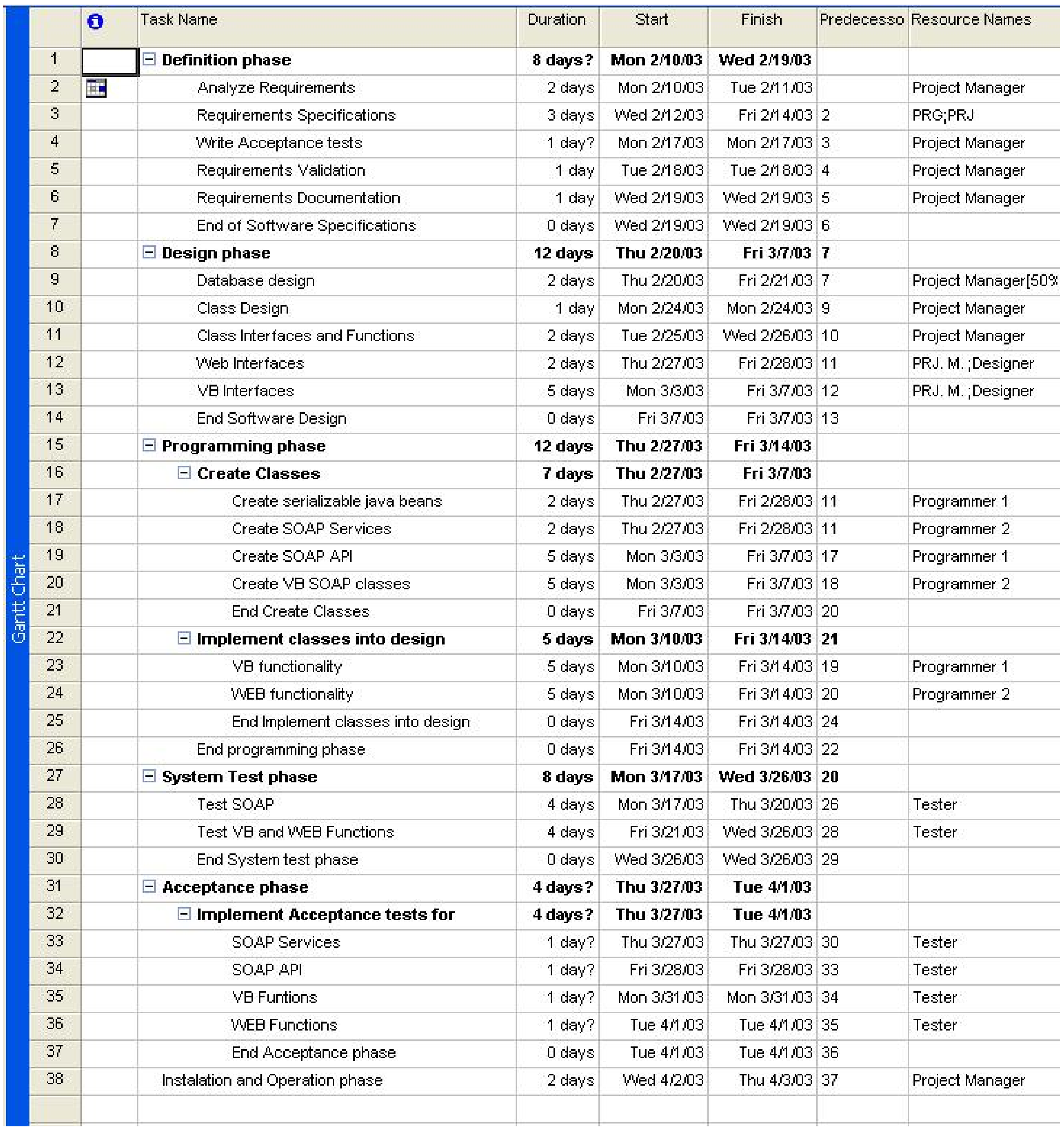
To do this, set the Axis global configuration property axis.enableListQuery to false.

Keep stack traces out of the responses

By default, Axis ships in *production* mode; stack traces do not get sent back to the caller. If you set axis.development.system to true in the configuration, stack traces get sent over the wire in faults. This exposes internal information about the implementation that may be used in finding weaknesses.

6.1 Project Management

To accomplish this project I've decided for a "water-flow" life cycle like project management style because it's easy to monitor and at any time I can have a picture of what is done and what's not, and at the same time I can be test the finished work.
I'll present in the next step the list of tasks and the "GANTT-CHART" for those tasks designed in Microsoft Project [*Figure x.x, Figure x.x*].



		Task Name	Duration	Start	Finish	Predecesso	Resource Names
1		**⊟ Definition phase**	**8 days?**	**Mon 2/10/03**	**Wed 2/19/03**		
2		Analyze Requirements	2 days	Mon 2/10/03	Tue 2/11/03		Project Manager
3		Requirements Specifications	3 days	Wed 2/12/03	Fri 2/14/03	2	PRG;PRJ
4		Write Acceptance tests	1 day?	Mon 2/17/03	Mon 2/17/03	3	Project Manager
5		Requirements Validation	1 day	Tue 2/18/03	Tue 2/18/03	4	Project Manager
6		Requirements Documentation	1 day	Wed 2/19/03	Wed 2/19/03	5	Project Manager
7		End of Software Specifications	0 days	Wed 2/19/03	Wed 2/19/03	6	
8		**⊟ Design phase**	**12 days**	**Thu 2/20/03**	**Fri 3/7/03**	**7**	
9		Database design	2 days	Thu 2/20/03	Fri 2/21/03	7	Project Manager[50%
10		Class Design	1 day	Mon 2/24/03	Mon 2/24/03	9	Project Manager
11		Class Interfaces and Functions	2 days	Tue 2/25/03	Wed 2/26/03	10	Project Manager
12		Web Interfaces	2 days	Thu 2/27/03	Fri 2/28/03	11	PRJ. M. ;Designer
13		VB Interfaces	5 days	Mon 3/3/03	Fri 3/7/03	12	PRJ. M. ;Designer
14		End Software Design	0 days	Fri 3/7/03	Fri 3/7/03	13	
15		**⊟ Programming phase**	**12 days**	**Thu 2/27/03**	**Fri 3/14/03**		
16		**⊟ Create Classes**	**7 days**	**Thu 2/27/03**	**Fri 3/7/03**		
17		Create serializable java beans	2 days	Thu 2/27/03	Fri 2/28/03	11	Programmer 1
18		Create SOAP Services	2 days	Thu 2/27/03	Fri 2/28/03	11	Programmer 2
19		Create SOAP API	5 days	Mon 3/3/03	Fri 3/7/03	17	Programmer 1
20		Create VB SOAP classes	5 days	Mon 3/3/03	Fri 3/7/03	18	Programmer 2
21		End Create Classes	0 days	Fri 3/7/03	Fri 3/7/03	20	
22		**⊟ Implement classes into design**	**5 days**	**Mon 3/10/03**	**Fri 3/14/03**	**21**	
23		VB functionality	5 days	Mon 3/10/03	Fri 3/14/03	19	Programmer 1
24		WEB functionality	5 days	Mon 3/10/03	Fri 3/14/03	20	Programmer 2
25		End Implement classes into design	0 days	Fri 3/14/03	Fri 3/14/03	24	
26		End programming phase	0 days	Fri 3/14/03	Fri 3/14/03	22	
27		**⊟ System Test phase**	**8 days**	**Mon 3/17/03**	**Wed 3/26/03**	**20**	
28		Test SOAP	4 days	Mon 3/17/03	Thu 3/20/03	26	Tester
29		Test VB and WEB Functions	4 days	Fri 3/21/03	Wed 3/26/03	28	Tester
30		End System test phase	0 days	Wed 3/26/03	Wed 3/26/03	29	
31		**⊟ Acceptance phase**	**4 days?**	**Thu 3/27/03**	**Tue 4/1/03**		
32		**⊟ Implement Acceptance tests for**	**4 days?**	**Thu 3/27/03**	**Tue 4/1/03**		
33		SOAP Services	1 day?	Thu 3/27/03	Thu 3/27/03	30	Tester
34		SOAP API	1 day?	Fri 3/28/03	Fri 3/28/03	33	Tester
35		VB Funtions	1 day?	Mon 3/31/03	Mon 3/31/03	34	Tester
36		WEB Functions	1 day?	Tue 4/1/03	Tue 4/1/03	35	Tester
37		End Acceptance phase	0 days	Tue 4/1/03	Tue 4/1/03	36	
38		Instalation and Operation phase	2 days	Wed 4/2/03	Thu 4/3/03	37	Project Manager

Figure x.x

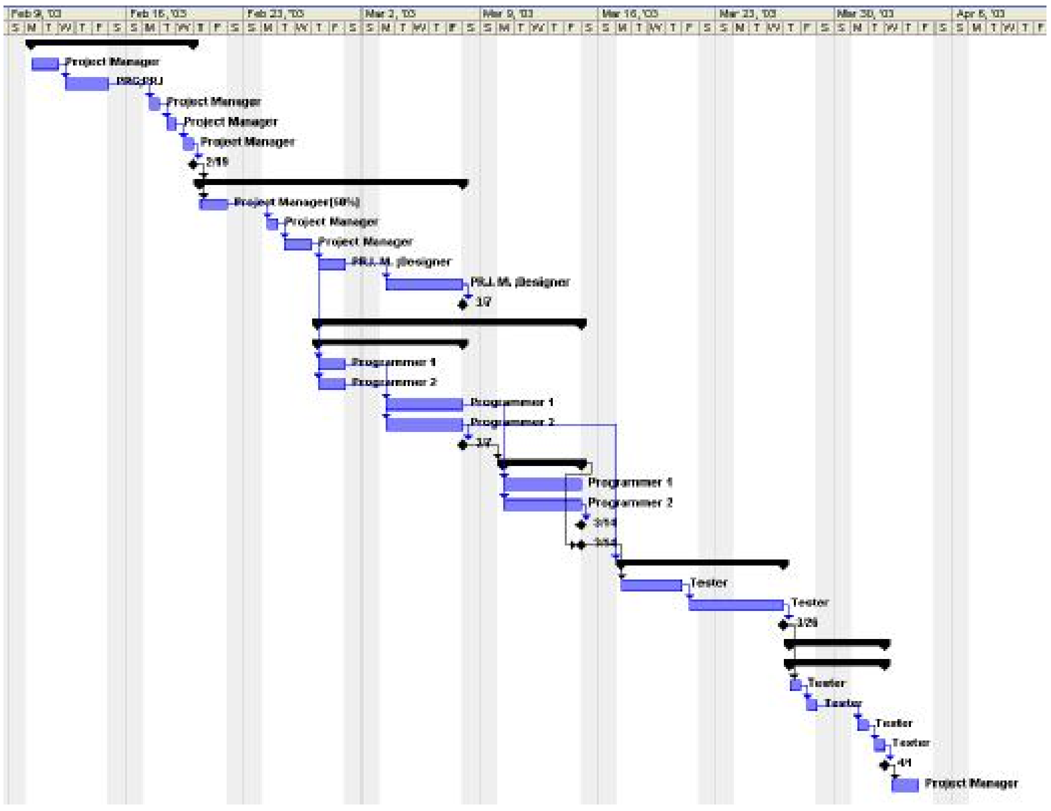

Feb 9, '03
Feb 16, '03
Feb 23, '03
Mar 2, '03
Mar 9, '03
Mar 16, '03
Mar 23, '03
Mar 30, '03
Apr 6, '03
Project Manager
PRG;PRJ
Project Manager
Project Manager
Project Manager
2/19
Project Manager[50%]
Project Manager
Project Manager
PRJ.M. ;Designer
PRJ.M. ;Designer
3/7
Programmer 1
Programmer 2
Programmer 1
Programmer 2
3/7
Programmer 1
Programmer 2
3/14
3/14
Tester
Tester
3/26
Tester
Tester
Tester
Tester
4/1
Project Manager

Figure x.x

4

Project design

4.1 Programming structure

The main problem in this project is to integrate different types of applications to communicate to each other trough a unique language. This is possible to realize using the XML and SOAP web service. But this is not all, this represents only the manner of connecting different applications to communicate trough a bridge, we need also to be able to interpret this kind of language so for this I decided for this technologies:

a) Java language for the end-server side
b) Axis web services middle tier responsible for bridging the front and end side
c) Visual basic language with SOAP toolkit for a client
d) Jsp for web browser application for a end user client
e) Apache Tomcat for web application server
f) MySql database server

Now let's get the big picture of what we have to do.
In this figures [*Figure x.x, Figure x.x, Figure x.x*] we can see the class diagrams witch represents java beans that correspond to each table from database database services and a web services. These are reused by public functions from web services.

4.2 Component description

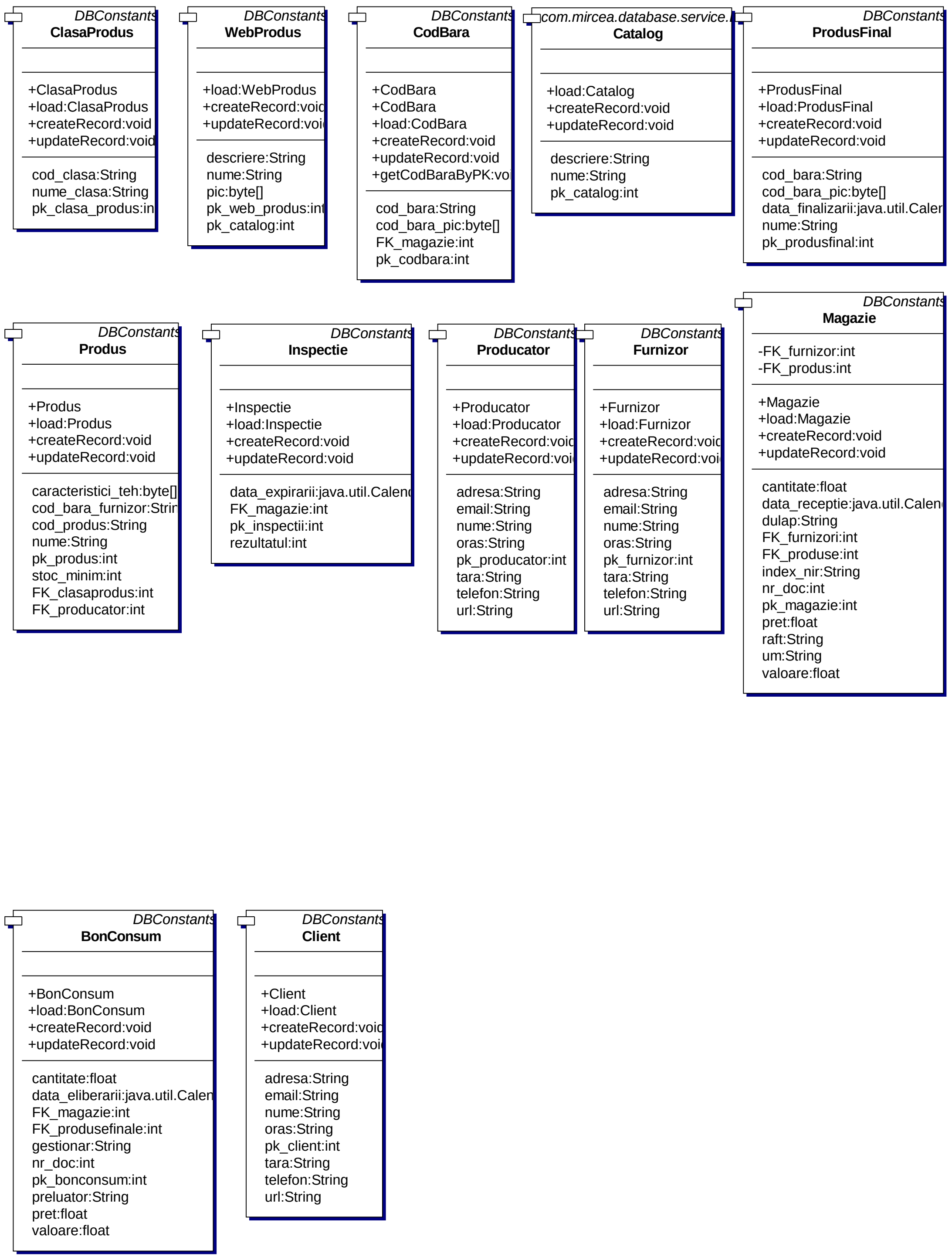

DBConstants
ClasaProdus
+ClasaProdus
+load:ClasaProdus
+createRecord:void
+updateRecord:void
cod_clasa:String
nume_clasa:String
pk_clasa_produs:in
DBConstants
WebProdus
+load:WebProdus
+createRecord:voic
+updateRecord:voi
descriere:String
nume:String
pic:byte[]
pk_web_produs:int
pk_catalog:int
DBConstants
CodBara
+CodBara
+CodBara
+load:CodBara
+createRecord:void
+updateRecord:void
+getCodBaraByPK:vo
cod_bara:String
cod_bara_pic:byte[]
FK_magazie:int
pk_codbara:int
com.mircea.database.service.
Catalog
+load:Catalog
+createRecord:void
+updateRecord:void
descriere:String
nume:String
pk_catalog:int
DBConstants
ProdusFinal
+ProdusFinal
+load:ProdusFinal
+createRecord:void
+updateRecord:void
cod_bara:String
cod_bara_pic:byte[]
data_finalizarii:java.util.Caler
nume:String
pk_produsfinal:int
DBConstants
Produs
+Produs
+load:Produs
+createRecord:void
+updateRecord:void
caracteristici_teh:byte[]
cod_bara_furnizor:Strir
cod_produs:String
nume:String
pk_produs:int
stoc_minim:int
FK_clasaprodus:int
FK_producator:int
DBConstants
Inspectie
+Inspectie
+load:Inspectie
+createRecord:void
+updateRecord:void
data_expirarii:java.util.Calen
FK_magazie:int
pk_inspectii:int
rezultatul:int
DBConstants
Producator
+Producator
+load:Producator
+createRecord:voic
+updateRecord:voi
adresa:String
email:String
nume:String
oras:String
pk_producator:int
tara:String
telefon:String
url:String
DBConstants
Furnizor
+Furnizor
+load:Furnizor
+createRecord:voic
+updateRecord:voi
adresa:String
email:String
nume:String
oras:String
pk_furnizor:int
tara:String
telefon:String
url:String
DBConstants
Magazie
-FK_furnizor:int
-FK_produs:int
+Magazie
+load:Magazie
+createRecord:void
+updateRecord:void
cantitate:float
data_receptie:java.util.Calen
dulap:String
FK_furnizori:int
FK_produse:int
index_nir:String
nr_doc:int
pk_magazie:int
pret:float
raft:String
um:String
valoare:float
DBConstants
BonConsum
+BonConsum
+load:BonConsum
+createRecord:void
+updateRecord:void
cantitate:float
data_eliberarii:java.util.Calen
FK_magazie:int
FK_produsefinale:int
gestionar:String
nr_doc:int
pk_bonconsum:int
preluator:String
pret:float
valoare:float
DBConstants
Client
+Client
+load:Client
+createRecord:voic
+updateRecord:voi
adresa:String
email:String
nume:String
oras:String
pk_client:int
tara:String
telefon:String
url:String

Figure x.x

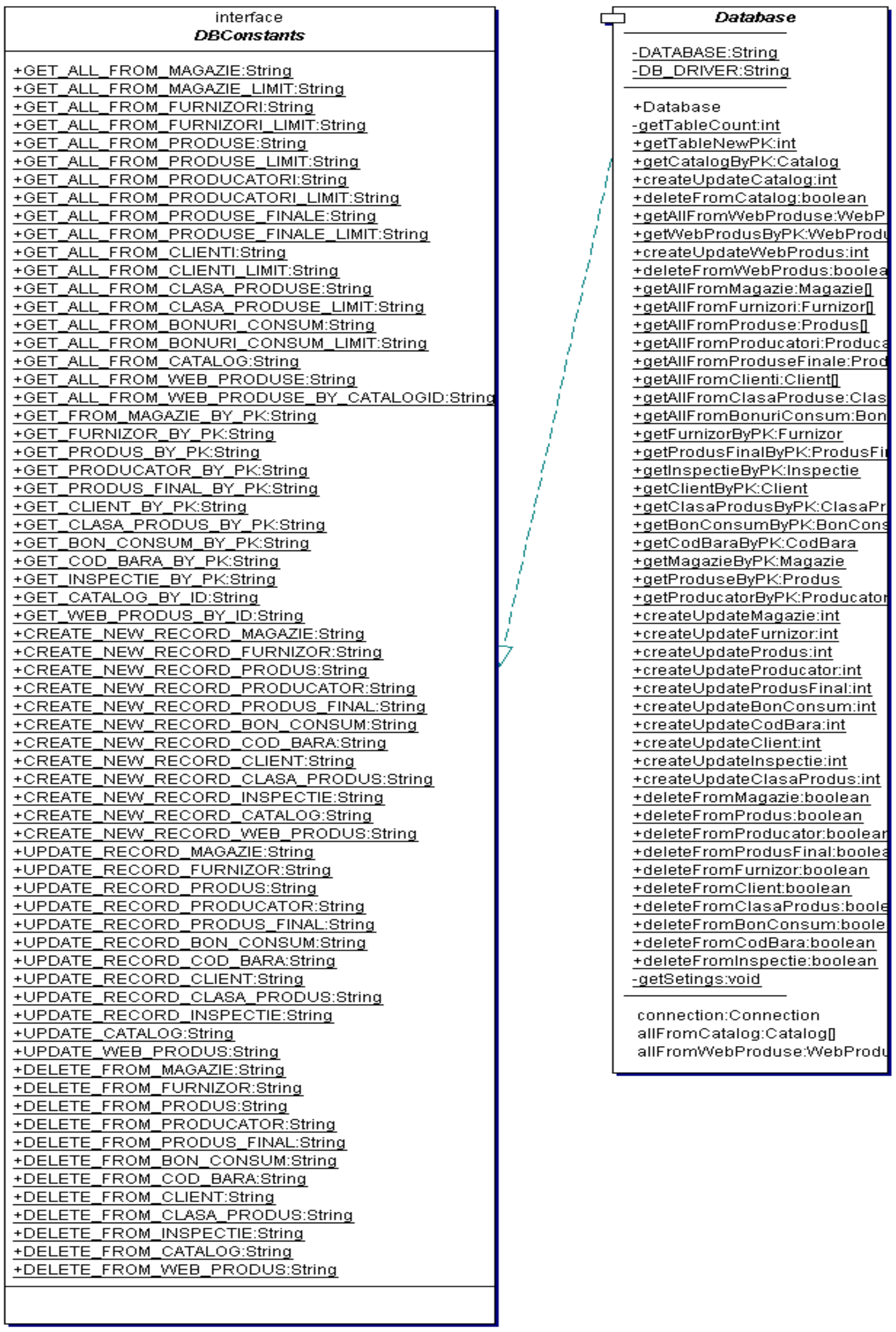

interface
DBConstants
+GET_ALL_FROM_MAGAZIE:String
+GET_ALL_FROM_MAGAZIE_LIMIT:String
+GET_ALL_FROM_FURNIZORI:String
+GET_ALL_FROM_FURNIZORI_LIMIT:String
+GET_ALL_FROM_PRODUSE:String
+GET_ALL_FROM_PRODUSE_LIMIT:String
+GET_ALL_FROM_PRODUCATORI:String
+GET_ALL_FROM_PRODUCATORI_LIMIT:String
+GET_ALL_FROM_PRODUSE_FINALE:String
+GET_ALL_FROM_PRODUSE_FINALE_LIMIT:String
+GET_ALL_FROM_CLIENTI:String
+GET_ALL_FROM_CLIENTI_LIMIT:String
+GET_ALL_FROM_CLASA_PRODUSE:String
+GET_ALL_FROM_CLASA_PRODUSE_LIMIT:String
+GET_ALL_FROM_BONURI_CONSUM:String
+GET_ALL_FROM_BONURI_CONSUM_LIMIT:String
+GET_ALL_FROM_CATALOG:String
+GET_ALL_FROM_WEB_PRODUSE:String
+GET_ALL_FROM_WEB_PRODUSE_BY_CATALOGID:String
+GET_FROM_MAGAZIE_BY_PK:String
+GET_FURNIZOR_BY_PK:String
+GET_PRODUS_BY_PK:String
+GET_PRODUCATOR_BY_PK:String
+GET_PRODUS_FINAL_BY_PK:String
+GET_CLIENT_BY_PK:String
+GET_CLASA_PRODUS_BY_PK:String
+GET_BON_CONSUM_BY_PK:String
+GET_COD_BARA_BY_PK:String
+GET_INSPECTIE_BY_PK:String
+GET_CATALOG_BY_ID:String
+GET_WEB_PRODUS_BY_ID:String
+CREATE_NEW_RECORD_MAGAZIE:String
+CREATE_NEW_RECORD_FURNIZOR:String
+CREATE_NEW_RECORD_PRODUS:String
+CREATE_NEW_RECORD_PRODUCATOR:String
+CREATE_NEW_RECORD_PRODUS_FINAL:String
+CREATE_NEW_RECORD_BON_CONSUM:String
+CREATE_NEW_RECORD_COD_BARA:String
+CREATE_NEW_RECORD_CLIENT:String
+CREATE_NEW_RECORD_CLASA_PRODUS:String
+CREATE_NEW_RECORD_INSPECTIE:String
+CREATE_NEW_RECORD_CATALOG:String
+CREATE_NEW_RECORD_WEB_PRODUS:String
+UPDATE_RECORD_MAGAZIE:String
+UPDATE_RECORD_FURNIZOR:String
+UPDATE_RECORD_PRODUS:String
+UPDATE_RECORD_PRODUCATOR:String
+UPDATE_RECORD_PRODUS_FINAL:String
+UPDATE_RECORD_BON_CONSUM:String
+UPDATE_RECORD_COD_BARA:String
+UPDATE_RECORD_CLIENT:String
+UPDATE_RECORD_CLASA_PRODUS:String
+UPDATE_RECORD_INSPECTIE:String
+UPDATE_CATALOG:String
+UPDATE_WEB_PRODUS:String
+DELETE_FROM_MAGAZIE:String
+DELETE_FROM_FURNIZOR:String
+DELETE_FROM_PRODUS:String
+DELETE_FROM_PRODUCATOR:String
+DELETE_FROM_PRODUS_FINAL:String
+DELETE_FROM_BON_CONSUM:String
+DELETE_FROM_COD_BARA:String
+DELETE_FROM_CLIENT:String
+DELETE_FROM_CLASA_PRODUS:String
+DELETE_FROM_INSPECTIE:String
+DELETE_FROM_CATALOG:String
+DELETE_FROM_WEB_PRODUS:String
Database
-DATABASE:String
-DB_DRIVER:String
+Database
-getTableCount:int
+getTableNewPK:int
+getCatalogByPK:Catalog
+createUpdateCatalog:int
+deleteFromCatalog:boolean
+getAllFromWebProduse:WebP
+getWebProdusByPK:WebProdu
+createUpdateWebProdus:int
+deleteFromWebProdus:boolea
+getAllFromMagazie:Magazie[]
+getAllFromFurnizori:Furnizor[]
+getAllFromProduse:Produs[]
+getAllFromProducatori:Produca
+getAllFromProduseFinale:Prod
+getAllFromClienti:Client[]
+getAllFromClasaProduse:Clas
+getAllFromBonuriConsum:Bon
+getFurnizorByPK:Furnizor
+getProdusFinalByPK:ProdusFi
+getInspectieByPK:Inspectie
+getClientByPK:Client
+getClasaProdusByPK:ClasaPr
+getBonConsumByPK:BonCons
+getCodBaraByPK:CodBara
+getMagazieByPK:Magazie
+getProduseByPK:Produs
+getProducatorByPK:Producator
+createUpdateMagazie:int
+createUpdateFurnizor:int
+createUpdateProdus:int
+createUpdateProducator:int
+createUpdateProdusFinal:int
+createUpdateBonConsum:int
+createUpdateCodBara:int
+createUpdateClient:int
+createUpdateInspectie:int
+createUpdateClasaProdus:int
+deleteFromMagazie:boolean
+deleteFromProdus:boolean
+deleteFromProducator:boolean
+deleteFromProdusFinal:boolea
+deleteFromFurnizor:boolean
+deleteFromClient:boolean
+deleteFromClasaProdus:boole
+deleteFromBonConsum:boole
+deleteFromCodBara:boolean
+deleteFromInspectie:boolean
-getSetings:void
connection:Connection
allFromCatalog:Catalog[]
allFromWebProduse:WebProdu

Figure x.x

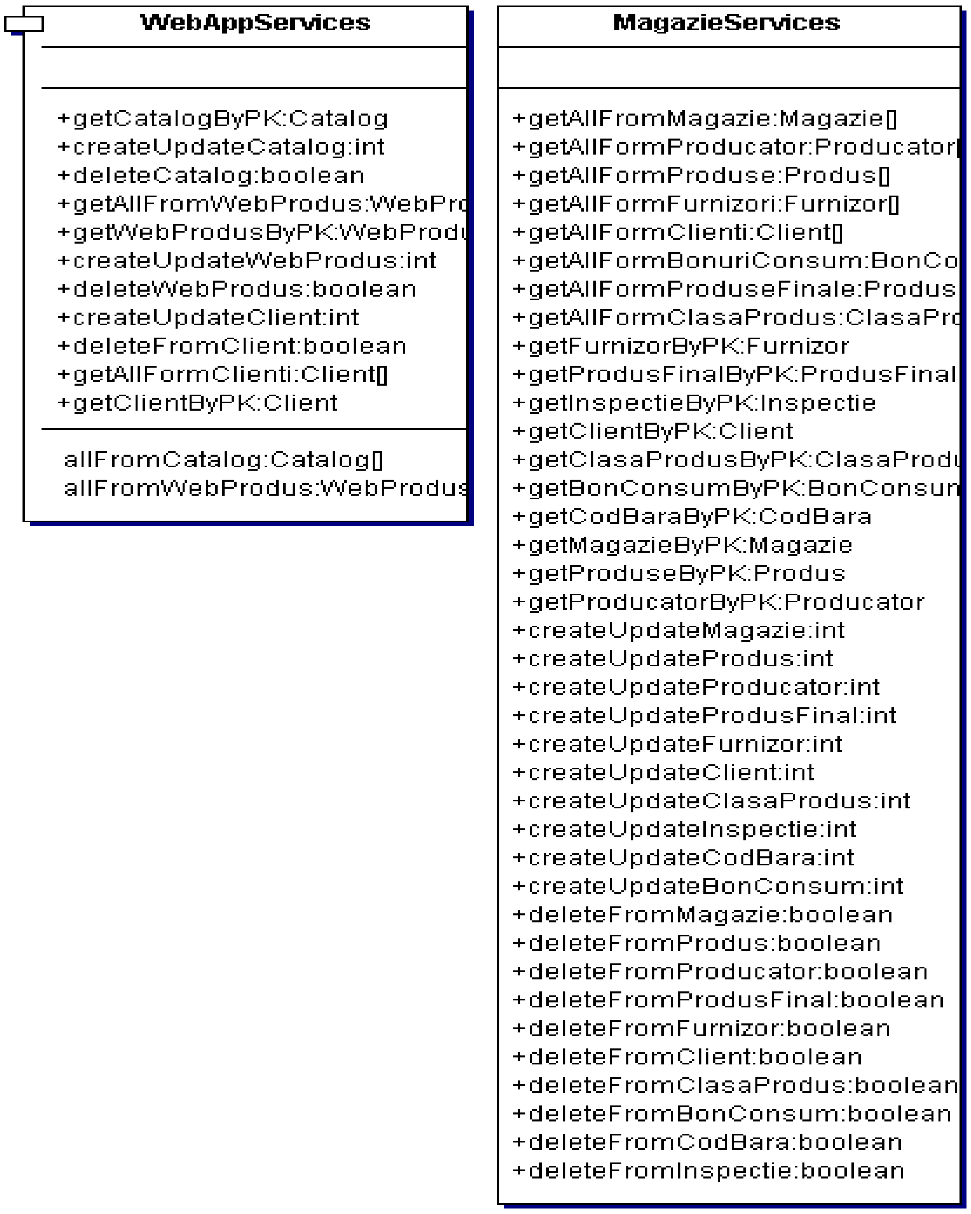

WebAppServices
+getCatalogByPK:Catalog
+createUpdateCatalog:int
+deleteCatalog:boolean
+getAllFromWebProdus:WebPro
+getWebProdusByPK:WebProdu
+createUpdateWebProdus:int
+deleteWebProdus:boolean
+createUpdateClient:int
+deleteFromClient:boolean
+getAllFormClienti:Client[]
+getClientByPK:Client
allFromCatalog:Catalog[]
allFromWebProdus:WebProdus
MagazieServices
+getAllFromMagazie:Magazie[]
+getAllFormProducator:Producator
+getAllFormProduse:Produs[]
+getAllFormFurnizori:Furnizor[]
+getAllFormClienti:Client[]
+getAllFormBonuriConsum:BonCo
+getAllFormProduseFinale:Produs
+getAllFormClasaProdus:ClasaPro
+getFurnizorByPK:Furnizor
+getProdusFinalByPK:ProdusFinal
+getInspectieByPK:Inspectie
+getClientByPK:Client
+getClasaProdusByPK:ClasaProdu
+getBonConsumByPK:BonConsum
+getCodBaraByPK:CodBara
+getMagazieByPK:Magazie
+getProduseByPK:Produs
+getProducatorByPK:Producator
+createUpdateMagazie:int
+createUpdateProdus:int
+createUpdateProducator:int
+createUpdateProdusFinal:int
+createUpdateFurnizor:int
+createUpdateClient:int
+createUpdateClasaProdus:int
+createUpdateInspectie:int
+createUpdateCodBara:int
+createUpdateBonConsum:int
+deleteFromMagazie:boolean
+deleteFromProdus:boolean
+deleteFromProducator:boolean
+deleteFromProdusFinal:boolean
+deleteFromFurnizor:boolean
+deleteFromClient:boolean
+deleteFromClasaProdus:boolean
+deleteFromBonConsum:boolean
+deleteFromCodBara:boolean
+deleteFromInspectie:boolean

Figure x.x

In the first figure [*Figure x.x*] we have the list of beans that performs and stores database data and actions.

Java Beans description

Package: com.• ircea.database.beans

All java beans have the same functionality so I'll present in general one.

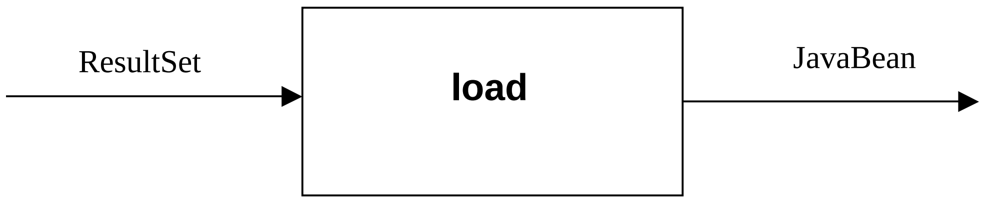
Implements: com.• ircea.database.service.DBConstants

This interface is used for defining SQL commands for select, update and delete actions.

Public setXX() and getXX() functions

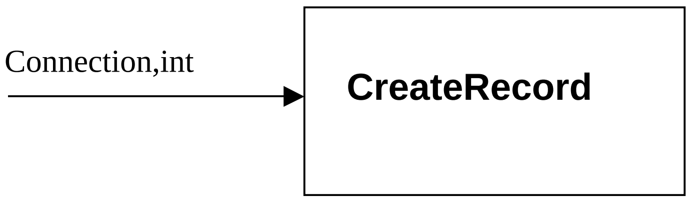
The next functions are the same for all other beans and are used for loading, creating and updating data into table from java bean and inverse.

Function: load(ResultSet) : JavaBean



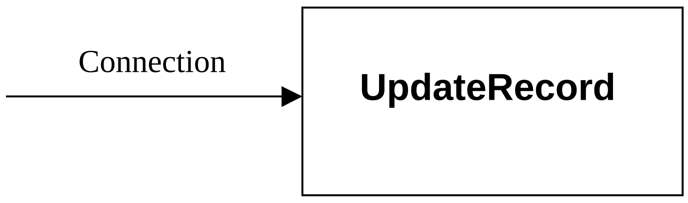
This function loads data from given ResultSet that is returned by selecting the database table into java bean.

Function: CreateRecord(Connection,int) : void



This function creates new record into database table with data from java bean, using given connection and new primary key for that record.

Function: UpdateRecord(Connection) : void



This function updates record into database table with data from java bean, using given connection for that record.

In the figure [*Figure x.x*] is represented the database services like database connection, and all functionalities for selecting, creating, updating and deleting a record. In this figure you can see all functionalities that has to be implemented and I'll discuss only how it has to be implemented the connection and how to use the connection for committing the SQL actions that has bean defined in DBConstants interface.

Database services description

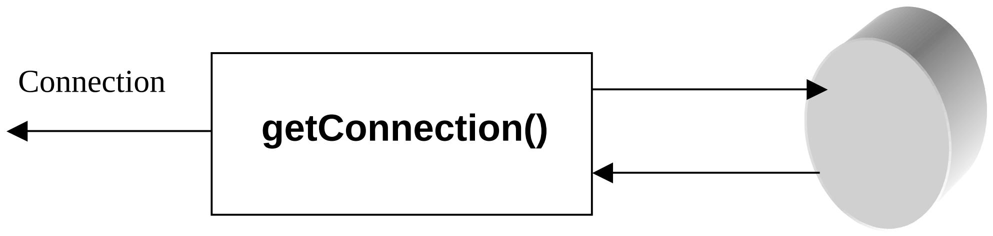
Package: com.mircea.database.service

Implements: com.mircea.database.service.DBConstants

Class: Database

This class performs all actions regarded to the database usage

The connection is done in function **getConnection()**, this function reads from a property file the database name and java connection driver. The database connection is returned after the connection is created with a database server.



Now using this connection we'll create all other functions for select, update, create and delete, here is an example for these actions:

```
public static Magazie[] getAllFromMagazie(int from,int to){
                Connection db_conn = getConnection();
                ResultSet res = null;
                Magazie mag[] = null;
                Magazie magazie = null;
                PreparedStatement ps = null;
        try{
                if(to == 0)
                  ps = db_conn.prepareStatement(GET_ALL_FROM_MAGAZIE);
                else{
                 ps = db_conn.prepareStatement(GET_ALL_FROM_MAGAZIE_LIMIT);
                 ps.setInt(1,from);
                 ps.setInt(2,to);
                }


                res = ps.executeQuery();
                int count = getTableCount(res);
                res.beforeFirst();
                mag = new Magazie[count];
                for(int i=0;i< count;i++){
                  res.next();
                  magazie = new Magazie();
                  mag[i] = magazie.load(res);
                }

          db_conn.close();
        }catch(SQLException ex){
                System.err.println("SQLException ");
                ex.printStackTrace();

        }
                return mag;
        }
```

Listing x.x

In this listing [*Listing x.x*] is an example for getting all records from a database table in a limit “from” – “to”.

```
public static Inspectie getInspectieByPK(int pk){
                        Connection conn = getConnection();
                        ResultSet res = null;
                        Inspectie insp = null;
            try{
                        PreparedStatement ps = conn.prepareStatement(GET_INSPECTIE_BY_PK);
                        ps.setInt(1,pk);
                        res = ps.executeQuery();
                        if(res.next()){
                                    insp = new Inspectie();
                                    insp.load(res);
                        }
               conn.close();
            }catch(SQLException ex){
                                    System.err.println("SQLException ");
                                    ex.printStackTrace();
            }
                        return insp;
            }
```

Listing x.x

In this listing [*Listing x.x*] is an example for getting a record from a database by primary key “pk”.

```
public static int createUpdateMagazie(Magazie mag){
            Connection conn = getConnection();
            int ret = 0;
            if(mag.getPk_magazie() > 0){ //Update
               ret = mag.getPk_magazie();
               mag.updateRecord(conn);

            }else{ //Create
              ret = getTableNewPK("pk_magazie","Magazie");
              mag.createRecord(conn,ret);

          }
    return ret;
}
```

Listing x.x

In this listing [*Listing x.x*] is an example of creating or updating a record by passing the object values and properties.

```
public static boolean deleteFromMagazie(int pk){
                Connection conn = getConnection();
                boolean res = false;
        try{
                PreparedStatement ps =
conn.prepareStatement(DELETE_FROM_MAGAZIE);
                ps.setInt(1,pk);
                ps.execute();
          res = true;
          conn.close();
        }catch(SQLException ex){
                        System.err.println("SQLException ");
                        ex.printStackTrace();
        }
                return res;
        }
```

Listing x.x

In this listing [*Listing x.x*] is an example of deleting a record by primary key “pk”.

Web Services public functions

Package: com.mircea.axis

Class: MagazieServices

Class: WebAppServices

These two classes are responsible for bridging the applications trough the SOAP protocol. To publish the web services using axis engine we need to copy the classes into “classes” directory where the axis is installed.

After we do that web have to deploy the functions that we need to be public, more about how to deploy see appendix [Appendix A].

4.3 Sequence diagram

In this section I'll present the interaction between a client – web services – database server. The next figure [*Figure x.x*] represents a sequence diagram between those components.

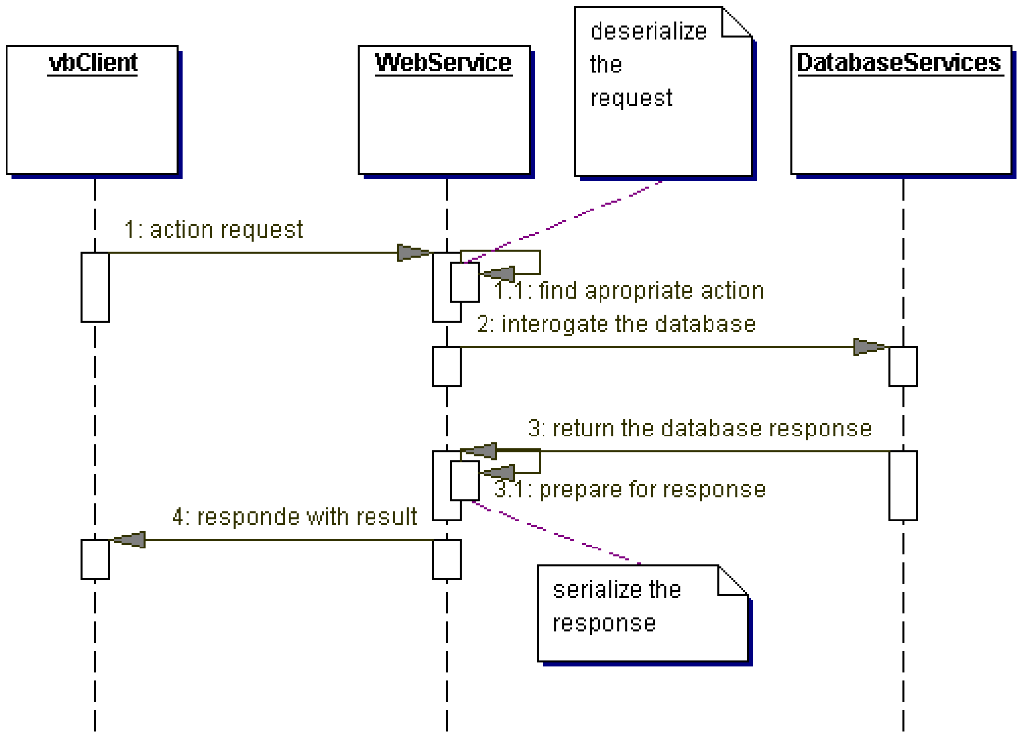


Figure x.x

Here we can see that when a client request an action from a web services, the web service first has to de-serialize that SOAP message request to find appropriate callable function and to perform that action, and at the end to serialize the response and to respond to the client with a SOAP message.

5

Installation

System requirements

For installing the application on a client server we need this requirements:

- java 1.4 or higher
- Apache Tomcat 4.0
- Axis 1.1 or higher
- MySql 3.21

Library dependencies

- ant libraries
- axis libraries
- jasperreports libraries
- webAppServicesAPI libraries

Installation

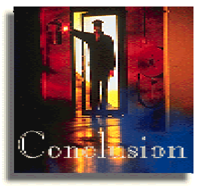
To install our application properly on a client server first of all it has to be installed the axis engine firs and after that to run the install ant application that will install all other libraries and will deploy the web service on that server.
For installing the Visual Basic application “virtualMag” will run the setup.exe file that will lunch a wizard for configuration of the network and other properties.

6

Final Results

6.1 Component tests

To assure that all components are working together properly we have to make some tests. We'll use the JUnit test tool from www.junit.org for testing java classes.



7

Conclusion

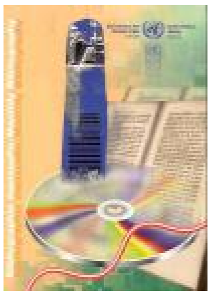
Web Services are one of the hottest technologies going right now but there are a number of pitfalls that any programmer must be aware of. Data Integrity is important for any object's proper functioning and must be assured or it could corrupt a larger program and generate a very difficult to trace error. By using the techniques from this project you can keep your objects safe and your data assured.

Today's business environment demands a new breed of Web applications that will accelerate your entry into new markets, help you reach and retain customers, and streamline operating efficiency. The widespread adoption of the Internet and the explosion of Web applications have provided the unprecedented ability to extend the value and impact of key enterprise functions with next generation solutions for customers, internal stakeholders, and partners. Now, Web Services are looming on the horizon of IT strategies and business plans, promising the next phase of business value via universal, distributed computing.

The real advantages by using the Web services are:

- Interoperability
- Ubiquity
- Loosely coupled applications
- Working applications
- Support of software industry leaders

For this project the real benefit is that we can administrate the warehouse at the single place and to have access to that database from other part of the globe.
Other interesting part is that this application can be easily extended into other domains.



8

Bibliography

[***2003,a]	Apache Software Foundation, Axis Web Service. http://ws.apache.org/axis/
[***2000,b]	Simple Object Access Protocol (SOAP) 1.1 http://www.w3.org/TR/SOAP/
[***1995,c]	MySQL® database server http://www.mysql.com/
[***2001,d]	Jasper Reports http://jasperreports.sourceforge.net/index.html
[***1999,e]	Jakarta Apache Tomcat http://jakarta.apache.org/tomcat/
[***2002,f]	Microsoft® SOAP Toolkit http://msdn.microsoft.com/library/default.asp?url=/nhp/default.asp?contentid=28000523
[***2003,g]	Web Services Journal http://www.sys-con.com/webservices/

Appendix A

Publishing Web Services with axis

Let's say we have a simple class like the following:

```
public class Calculator {
  public int add(int i1, int i2)
  {
    return i1 + i2;
  }

  public int subtract(int i1, int i2)
  {
    return i1 - i2;
  }
}
```

(You'll find this very class in samples/userguide/example2/Calculator.java.)

How do we go about making this class available via SOAP? There are a couple of answers to that question, but we'll start with the easiest way Axis provides to do this, which takes almost no effort at all!

JWS (Java Web Service) Files - Instant Deployment

OK, here's step 1 : copy the above .java file into your webapp directory, and rename it "Calculator.jws". So you might do something like this:
% copy Calculator.java *<your-webapp-root>*/axis/Calculator.jws
Now for step 2... hm, wait a minute. You're done! You should now be able to access the service at the following URL (assuming your Axis web application is on port 8080):

http://localhost:8080/axis/Calculator.jws

Axis automatically locates the file, compiles the class, and converts SOAP calls correctly into Java invocations of your service class. Try it out - there's a calculator client in samples/userguide/example2/CalcClient.java, which you can use like this:

```
% java samples.userguide.example2.CalcClient -p8080 add 2 5
Got result : 7
% java samples.userguide.example2.CalcClient -p8080 subtract 10 9
Got result : 1%
```

(note that you may need to replace the "-p8080" with whatever port your J2EE server is running on)

Important: JWS web services are intended for simple web services. You cannot use packages in the pages, and as the code is compiled at run time you can not find out about errors until after deployment. Production quality web services should use Java classes with custom deployment.

Custom Deployment - Introducing WSDD

JWS files are great quick ways to get your classes out there as Web Services, but they're not always the best choice. For one thing, you need the source code - there might be times when you want to expose a pre-existing class on your system without source. Also, the amount of configuration you can do as to how the service gets accessed is pretty limited - you can't specify custom type mappings, or control which Handlers get invoked when people are using your service. *(note for the future : the Axis team, and the Java SOAP community at large, are thinking about ways to be able to embed this sort of metadata into your source files if desired - stay tuned!)*

Deploying via descriptors

To really use the flexibility available to you in Axis, you should get familiar with the Axis **Web Service Deployment Descriptor (WSDD)** format. A deployment descriptor contains a bunch of things you want to "deploy" into Axis - i.e. make available to the Axis engine. The most common thing to deploy is a Web Service, so let's start by taking a look at a deployment descriptor for a basic service (this file is samples/userguide/example3/deploy.wsdd):

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
           xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
 <service name="MyService" provider="java:RPC">
  <parameter name="className"
value="samples.userguide.example3.MyService"/>
  <parameter name="allowedMethods" value="*"/>
 </service>
</deployment>
```

Pretty simple, really - the outermost element tells the engine that this is a WSDD deployment, and defines the "java" namespace. Then the service element actually defines the service for us. A service is a **targeted chain** (see the Architecture Guide), which means it may have any/all of: a request flow, a pivot Handler (which for a service is called a "provider"), and a response flow. In this case, our provider is "java:RPC", which is built into Axis, and indicates a Java RPC service. The actual class which handles this is **org.apache.axis.providers.java.RPCProvider**. We'll go into more detail later on the different styles of services and their providers.

We need to tell the RPCProvider that it should instantiate and call the correct class (e.g. samples.userguide.example3.MyService), and we do so by including <parameter> tags, giving the service one parameter to configure the class name, and another to tell the engine that any public method on that class may be called via SOAP (that's what the "*" means; we could also have restricted the SOAP-accessible methods by using a space or comma separated list of available method names).

Advanced WSDD - specifying more options

WSDD descriptors can also contain other information about services, and also other pieces of Axis called "Handlers" which we'll cover in a later section.

Scoped Services

Axis supports scoping service objects (the actual Java objects which implement your methods) three ways. "Request" scope, the default, will create a new object each time a SOAP request comes in for your service. "Application" scope will create a singleton shared object to service **all** requests. "Session" scope will create a new object for each session-enabled client who accesses your service. To specify the scope option, you add a <parameter> to your service like this (where "*value*" is request, session, or application):

```
<service name="MyService"...>
  <parameter name="scope" value="value"/>
  ...
</service>
```

Using the AdminClient

Once we have this file, we need to send it to an Axis server in order to actually deploy the described service. We do this with the AdminClient, or the "org.apache.axis.client.AdminClient" class. If you have deployed Axis on a server other than Tomcat, you may need to use the -p <*port*> argument. The default port is 8080. A typical invocation of the AdminClient looks like this:

```
% java org.apache.axis.client.AdminClient deploy.wsdd
<Admin>Done processing</Admin>
```

This command has now made our service accessible via SOAP. Check it out by running the Client class - it should look like this:

```
% java samples.userguide.example3.Client -
lhttp://localhost:8080/axis/services/MyService "test me!"
You typed : test me!
%
```