

MERMAID

EU Project No: IST-1999-10637

Broker Interface Applet Data Protocols Design/Documentation

Deliverable Nos: D5 and D6
(Rev 3)

October 2001



Report Title:	Broker Interface Applet and Data Protocols Design/Documentation
Customer:	European Commission, Directorate-General Information Society, IST Programme
TXT Report no:	23401/D/05
Deliverable nos:	D5, D6
Report status:	Rev 3
Date:	October 2001
Contact details:	TXT e-Solutions S.p.A via Frigia 27, 20126 Milano, Italy e-mail: matteo.villa@txt.it web: www.txt.it

	Name	Signature	Date
Author:	Matteo Villa		
Approved by:	Paul Taylor		

This report is commercial-in-confidence. Any information contained herein should not be communicated to any third party without prior permission of TXT e-Solutions S.p.A.



Contributors

Company	Name
TXT	Matteo Villa
CEDRE	Vincent Gouriou
BMT	Paul Taylor
	Chris Rawlings
	Paul Goddard
IMGW	Bogusz Piliczewski
MO	Jack Hopkins

Distribution List

Company	Name	Number of Copies
EU Commission (DG-XIII)	Guy Weets	2
BMT	Paul Taylor	1
TXT	Matteo Villa	1
CEDRE	Vincent Gouriou	1
Met. Office	Paul Lancaster	1
IMGW-OM	Wlodek Kryzminski	1
NC	Koen DeHulsters	1



Revision

Revision Number	Date	Author	Purpose of Revision
1	31/08/2001	Matteo Villa	Initial Release
2	11/09/2001	Matteo Villa	Updated with IMGW and BMT contribution (description of their end applications)
3	26/09/01	Paul Taylor	Addition of search and results parameters, and re-formatting



Table of Contents

1. Introduction	6
1.1. Mermaid System Architecture.....	6
2. Broker Interface Component	9
2.1. Role of the Interface Component.....	9
2.1.1. Utilisation scenarios.....	9
2.2. Detailed Specifications.....	10
2.2.1. Public Functions (class “Consumer”)	11
2.2.2. Public Properties (class “Consumer”):.....	14
2.2.3. Other Classes:	14
2.3. How to Use the Interface Component.....	16
2.4. Testing	16
2.5. Future Integration.....	20
2.5.1. BMT application	20
2.5.2. IMGW application	21
3. Communication Protocols.....	22
3.1. The Communication Scenario	22
3.1. The SOAP Approach	23
3.2. Implementing SOAP in MERMAID	25
3.2.1. IF Component to / from broker Gateway.....	26
3.2.2. DAE to/from Broker Gateway	26
3.2.3. RDAE to / from DAE	26
Appendices	
A1. Appendix 1: Example WSDL File for IF Component to / from broker Gateway communication.....	27
A2. Appendix 2: Example WSDL File for the DAE to/from Broker Gateway communication.....	31



1. Introduction

This document details the following two deliverables developed during Workpackage 4 (WP4):

- D5 – Design, code and documentation for the Broker Interface Applet
- D6 – Design, code and documentation for the communication protocols and data despatch systems

It should be noted that the Interface has been renamed as the Broker Interface Component, as this will not be an applet in its true sense.

As the Broker Interface Component and the Communication Protocols are so closely linked, it is logical to describe them together, rather than in two separate documents.

In addition, it should be noted that the components developed within this workpackage will subsequently be used in WP7 “Data Broker Applications Prototyping”, which will be embedded into end user’s applications and within WP8 “Performance Assessment and Exploitation Evaluation”.

1.1. Mermaid System Architecture

There are two main products developed in WP4, namely;

- Broker Interface Component (formerly Applet)
- Communication Protocols

Generally speaking, the “Broker Interface Component” is a component that allows a “Computer-to-Computer” interaction between a generic application and the Mermaid Data Broker, while the “Communication Protocols” are those protocols that define what and how the two applications should communicate.

In order to have a better view of the scenario, it is necessary to consider the High Level Architecture of Mermaid (which is fully described in document “D2 – Mermaid System HLD”).

Figure 1 gives an overview of the various components that form the system. A brief description of them follows:

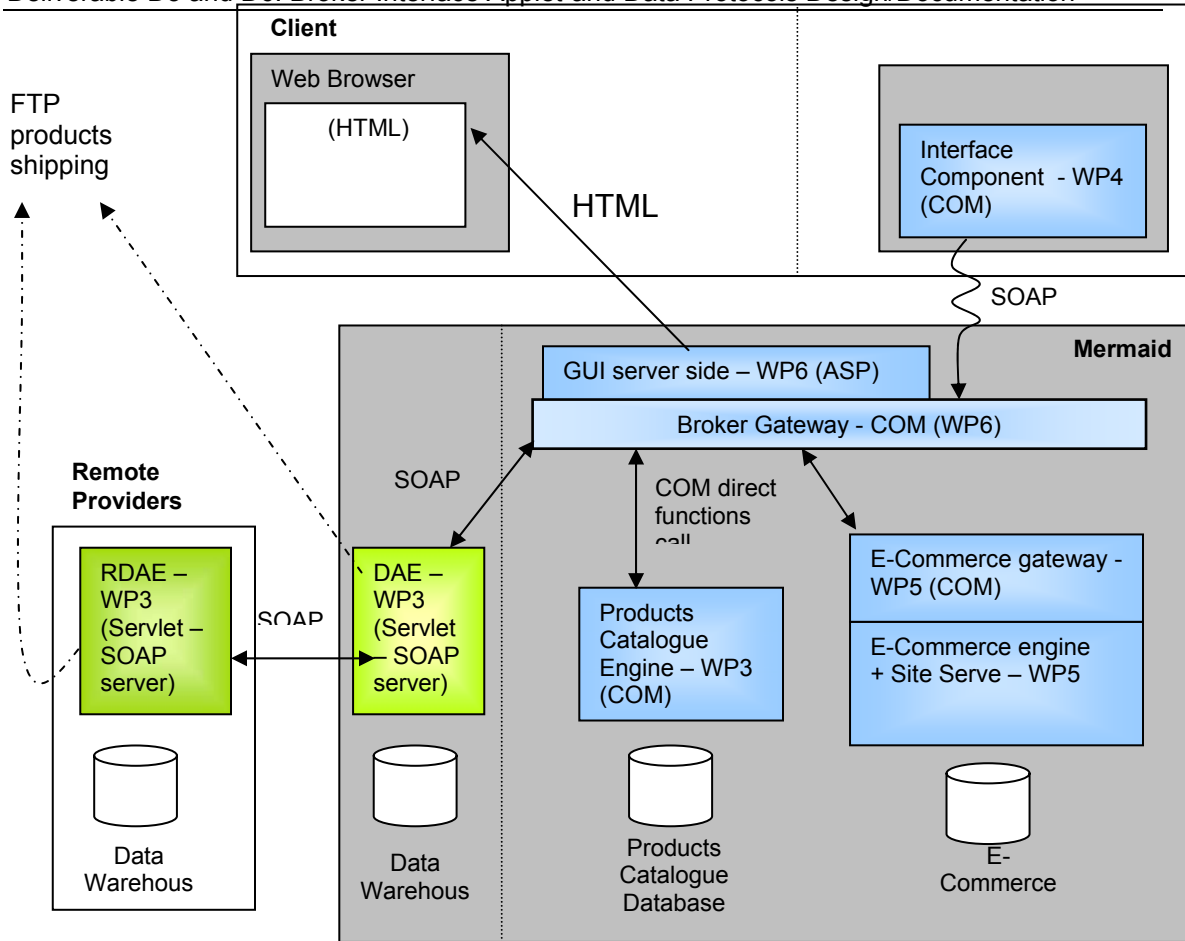


Figure 1: the Mermaid System HLD

- **Clients:** these can be either a web browser or an End Application
- **E-Commerce Gateway (WP5):** in charge of computing the prices, discounts, on-line purchasing, etc.. It is implemented with MS COM technology
- **Catalogue Engine (WP3):** managing the Metadata for products description and search. It is implemented with MS COM technology
- **DAE / RDAE (WP3):** managing files shipping and sub-sets extraction. Implemented with Java technology
- **GUI Server-side (WP6):** the user interfaces for the Web Browser client. Implemented with MS ASP technology
- **Broker Gateway (WP6):** integrating all the components. Implemented with MS COM technology
- **Interface Component (WP4):** the component to be plugged into an End Application. Implemented with MS COM technology
- **Databases:** all implemented with MS SQL server, except the providers datawarehouses, that may vary according providers choices
- **Communication (WP4):** represented with the arrows in figure 1.



This should clarify the role of WP4: the Interface Component (as it is now called) is a component that, when plugged into an End Application, allows the application to communicate with Mermaid via the Broker Gateway, using a specific communication protocol.

The same protocol is then used (clearly exchanging different messages) for the communication between:

- Broker Gateway vs Data Access Engine (DAE)
- DAE vs Remote Data Access Engine (RDAE)

It is important to emphasize the fact that between the Broker Gateway, the E-Commerce Engine and the Products Catalogue the communication is made by Microsoft COM/DCOM technology, as these components are all implemented with the same COM technology.

This is not the case for the DAE/RDAE and the Interface Component because they are either:

- implemented with different technologies (i.e. Java vs VB)
- distributed on different machines outside the intranet
- both the above two conditions apply

2. Broker Interface Component

2.1. Role of the Interface Component

As explained in the previous section, the interface component will allow a generic application to directly interface with Mermaid.

Within the Mermaid project this will be tested with two applications:

- BMT application: oil spill simulator
- IMGW application: on-line map builder

Whatever the application, the Interface (IF) Component will offer a set of API's able to communicate with the Mermaid Data Broker (via the Broker Gateway) and to execute its functionalities.

In general these applications will have, as a common requirement, the need of receiving datasets from the Data Broker. We can imagine that these applications will have a set of data (i.e. static data) that are always available and a set of data that are time-variant, and that need to be updated, or purchased. Figure 2 clarifies the scenario:

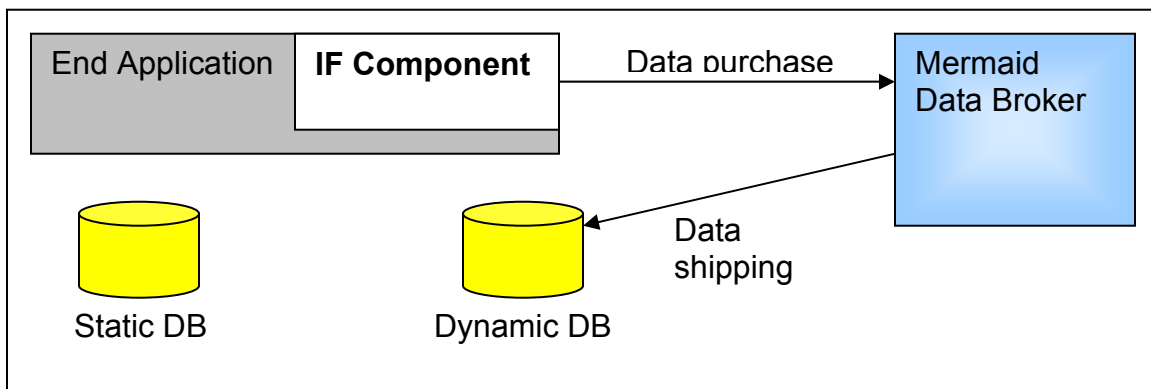


Figure 2: the utilisation scenario

2.1.1. Utilisation scenarios

It is necessary to clearly define what processes of those that will be available in the data broker the component should support. To answer this question it is necessary to consider the scenarios in which this component will be used:



1. the component will be plugged into an end application for a computer-to-computer communication
2. the goal of a computer-to-computer interaction is to speed up certain operations, so that the user does not have to leave the application they are working with
3. for a generic dataset search/purchase operations the Web interface of MERMAID gives the possibility of executing all the functionalities
4. a user working with an application that needs to receive data, wants to receive this data as soon as they are working with the application
5. to create/edit a dataset in the MERMAID site is not a time-critical operation, and therefore can be executed via the Web interface

Starting from these points we can conclude the following about all the processes supported by the Web interface:

Process	Need
Dataset find/purchase, shipped online (via FTP)	yes, as the data are received as soon as they are paid for.
Dataset find/purchase, shipped off-line (including e-mail shipping)	no, as the user cannot wait for an extended period in front of their application, that is stopped while waiting for the data. If they can wait such a long time, then they can do such a purchase operation via the Web interface
Dataset creation / edit	no, as the creation of a dataset in the MERMAID data broker is not an operation that may be required by an end application
Site administration	no

2.2. Detailed Specifications

This section provides the detailed implementation specifications of the Interface Component.

The component has been developed with MS Visual Basic 6 as a COM DLL. It has been developed in a way that it is 100% compatible with MS Visual C++ and can be used within ASP pages (all data types are VARIANT)

Therefore the Interface Component can be plugged into:

- MS VB/VC++ applications
- MS ASP pages



Note about Java: it is also possible to use the component in a Java application through the JNI interface, by using the following procedure:

1. write a JNI compliant wrapper class in C++, to encapsulate the DLL.
2. use the wrapper class from the Java application, thanks to the JNI interface

The following classes are implemented:

“Mermaidifproject.Consumer”
 “Mermaidifproject.CCoordinates”
 “Mermaidifproject.CSearchResults”
 “Mermaidifproject.CPayment”
 “Mermaidifproject.CBillingAddress”

Class “Consumer” implements functions, while classes “CCoordinates”, “CSearchResults”, “CPayment” and “CBillingAddress” are used to store data structures.

Finally, the component is able to communicate via the SOAP protocol (explained in the next section), thus it implements a MS SOAP client (that is another COM DLL).

MS SOAP 2.0 Gold Edition was used to build the SOAP client.

The methods and properties available in the classes are detailed below:

2.2.1. Public Functions (class “Consumer”)

The following tables explain all the functions available in class “Consumer”. All functions return 1 for OK and 0 for error.

Public Function Login(Username As Variant, Password As Variant, WSDL As Variant) As Variant
Log consumers in. Must be the first function called.
PARAMS:
'Username', 'Password' = user name and password
'WSDL' the address of the Broker Gateway SOAP Server WSDL file

Public Function SearchProduct(Coordinates() As Variant, DataTheme As Variant, Use As Variant, ProductName As Variant, byRef ProductID() As Variant) As Variant



Search for products, returning list of product ID. Use function 'ProductDetails' to see the details about the retrieved products

NOTE: there are two ID to describe products in MERMAID. 'ProductID' is unique for each product, while 'pf_id' (used in function product details for example) refers to the ID of the metadata description. This is because datastreams have the same metadata description but may have different purchase options (i.e. 1 week subscription, 2 weeks subscription, etc...)

PARAMS:

'Coordinates()': search coordinates. Object of type "CCoordinates" (see end of documet) – this includes box/circle coordinates, time start/end and altitude/depth

'DataTheme': defined in the Metadata Specification – D3.1, Appendix A

'Use' type of use the consumer is claiming to perform (allowed values are: 'Any', 'Commercial', 'Research', 'Governative').

'ProductName': name of the products to be retrieved or similar string

'ProductID()': returned products ID (SKU). Object of type "CSearchResults" (see end of documet)

NOTE: only return items that have the "OnLine" shipping method

Public Function ProductDetails(ProductID As Variant, byRef pf_id As Variant, byRef ProductName As Variant, byRef ProductProvider as Variant, byRef ProductPrice as Variant, byRef DatastreamDescription As Variant) As Variant

This function returns all the relevant details about a certain product, starting from its product ID.

PARAMS:

'ProductID': MERMAID unique product ID (SKU)

'pf_id': Returned MERMAID Dataset ID

'ProductName' : returned name of the product

'ProductProvider' : returned name of the product provider

'ProductPrice': returned price of the product

'DatastreamDescription': returned datastream description

Boundary Co-ordinates: (co-ordinates of the set or subset, as Top, Left, Right, Bottom) returned from product metadata description

Time Band (Start and End Time of the dataset or subset) returned from product metadata description

Depth / Altitude: (of the dataset or subset) :returned from product metadata description

Data Theme: returned from product metadata description

Accuracy : returned from product metadata description

Data-Source : returned from product metadata description

Storage-Medium : returned from product metadata description

Data-Form : returned from product metadata description



Summary : returned from product metadata description
--

Public Function AddItem(ProductID As Variant) As Variant

Add an Item to the consumer's shopping basket. Only products supporting "OnLine" shipping methods are allowed (other shipping methods would make no sense for the interface component)
--

PARAMS:

'ProductID': MERMAID unique product ID
--

<i>NOTE: only allow items that have the "OnLine" shipping method</i>
--

Public Function CompleteOrder(FTP_Address As Variant, FTP_Dir As Variant, FTP_Login As Variant, FTP_Pwd As Variant, Default_Values As Variant) As Variant
--

Add online shipping info to the order (NB only online shipping is supported)
--

PARAMS:

'FTP_...': FTP details. Not mandatory if parameter 'Default_Values' is set to 'Y'

'Default_Values': indicator stating whether the shopper's default FTP address shall be used ('Y') or the addresses provided with the parameters 'FTP_...' ('N')

<i>NOTE: add shipping details for ALL the items in the Order Form</i>

Public Function PayOrder(Billing_address() As Variant, Payment_method As Variant, Payment_params() As Variant, byRef Receipt_ID as Variant, Default_Values As Variant) As Variant
--

Pay the order and start the shipping process
--

PARAMS:

'Billing_address' = billing address. Object of type "CBillingAddress" (see end of documet). May be left empty if parameter "Default_Values" is set to 'Y'

'Payment_method' = for the moment, can only be 'CC' (for Credit Card). In future, we'll see

'Payment_params' = Payment method specific parameters. Object of type "CPayment" (see end of this document)

'ReceiptID': the returned receipt ID

'Default_Values': indicator stating whether the shopper default billing address shall be used ('Y') or the address provided with parameter "Billing_address" ('N')
--



Public Function GetDownloadStatus(byRef Completed as Double) As Variant
--

Returns the download status of the purchased items
--

PARAMS:

'Completed': percentage of completed download (between 0 and 100)

NOTE: this function should be called iteratively. An alternative can be an event (fired when download finishes) but this couldn't then be used in ASP pages.

Public Function GetLastErrors() As Variant

Return an error string relative to last function called (note: use this function when some other function returns 0)
--

2.2.2. Public Properties (class "Consumer"):

Public UserID As String

This is the MERMAID Shopper unique ID. It is set by function 'Login'. It is just for information, but if its value is modified, it will cause the Broker Gateway to reject all further connections.

Public pUse As String

Type of use the consumer is claiming to perform.

This value can be set directly, but it is also set in function 'ProductSearch', so after calling that function it is no longer necessary to set pUse.

2.2.3. Other Classes:

Public CBilling_address

Stores the shopped billing address values

Members:

- | |
|--|
| • Public Full_name As Variant |
| • Public Organisation_legal_name As Variant |
| • Public Street As Variant |
| • Public City As Variant |
| • Public Country As Variant |
| • Public ZIP_code As Variant |
| • Public Phone As Variant |



Public CCoordinates
Stores the user's search coordinates. These can be either a rectangular box (x1, x2, y1, y2) or a circle (x1, x2, r). For both of them it is possible to specify an altitude/depth (z) and in a specific time slice (t1, t2)
<u>Members:</u>
<ul style="list-style-type: none"> Public SearchType As Variant: type of search - "Box" (default) or "Circle" Public x1 As Variant: top-left corner x coordinate (case of "Box" search) or circle centre (case of "Circle" search) Public x2 As Variant: bottom right corner x coordinate Public y1 As Variant: top-left corner y coordinate (case of "Box" search) or circle centre (case of "Circle" search) Public y2 As Variant: bottom right corner y coordinate Public Radius As Variant: radius length (case of "Circle" search) Public z As Variant: altitude (if >0) or depth (if <0) Public t1 As Variant: time start Public t2 As Variant: time end
Public CSearchResults
Stores the results of a search
<u>Members:</u>
<ul style="list-style-type: none"> Public NumRes As Integer : total number of search results Public ID As Collection : collection of retrieved products ID (SKU)
Public CPayment
Stores the payment parameters.
<u>Members:</u>
<ul style="list-style-type: none"> Public Param As Collection : collection of payment parameters. These may vary according to each payment method. So far, only the Credit Card payment method has been defined, thus there are 6 parameters defined, in this order: cc_firstname, cc_lastname, cc_number, cc_type, cc_expiration_year, cc_expiration_month



2.3. How to Use the Interface Component

This section describes how a generic application should use the methods described in the previous section.

The functions described can be used in this order:

1. First you have to call "**Login**": this will establish the communication with the Broker and log the user into MERMAID.
2. Then you have to **add products** to the shopping basket. This can be done either directly (if you already know the products ID) with function "**AddItem**" or by first searching for products. In this last case, it is possible to **search** with function "**SearchProduct**", returning a list of products ID, and then "**ProductsDetails**" to see more **details** about a specific product (i.e. its name, provider, etc..)
Note that before adding an item to the basket, it is necessary to declare what kind of use it is intended to perform with the product. This can be done either by specifying the parameter "use" in function "search" or by directly setting the public property "pUse".
3. When all the items have been added to the shopping basket, it is necessary to complete the order by specifying **the shipping address**, with function "**CompleteOrder**". All the fields of the shipping address are mandatory
4. Finally, it is possible to **pay on-line** with function "**PayOrder**". This will also start the on-line shipping process.
5. Function "**GetDownloadStatus**" will inform about the completion of the online shipping operation

NOTES:

- a) note that only those products supporting online shipping method can be managed
- b) every time a function returns an error code (something different than '1') it is possible to have more details about the error by calling function "**GetLastErrors**"

2.4. Testing

As the component is a library, it has no user interface. Therefore there are two different kinds of testing that are possible with the IF Component:

- 1) test the component integrated in the end users applications: this will be done in WP 7



- 2) test the component in itself: this means test the functionalities of the system with a temporary GUI

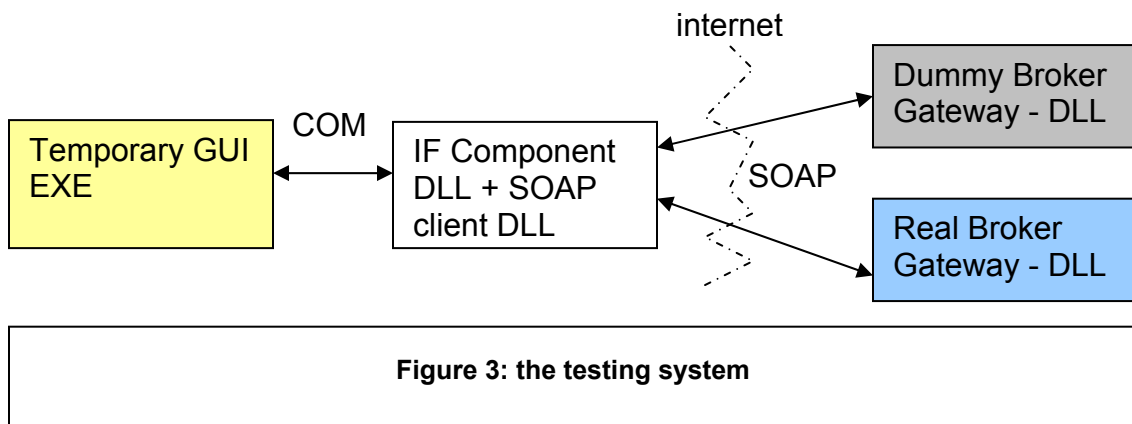
The testing carried out in this workpackage is therefore about option number 2. For this purpose two components have been developed:

- **a temporary user interface:** this is to allow the testing of all the functionalities of the IF component
- **a “dummy” broker gateway:** before interfacing with the real broker gateway, we developed a “dummy” one, i.e. a broker gateway with the right methods names but always returning a success (or error) code

The testing followed this path:

- test with the temporary interface connecting to the dummy broker gateway
- test with the temporary interface connecting to the real broker gateway: in this way complex error situations could be tested

The architecture of the test system was as shown in Figure 3:



The temporary GUI was a Windows EXE developed with MS Visual Basic 6.0, while the IF Component was connected to either the dummy or real broker gateway through the SOAP protocol (explained in the next section)

A screenshot of the temporary GUI is shown in Figure 4:

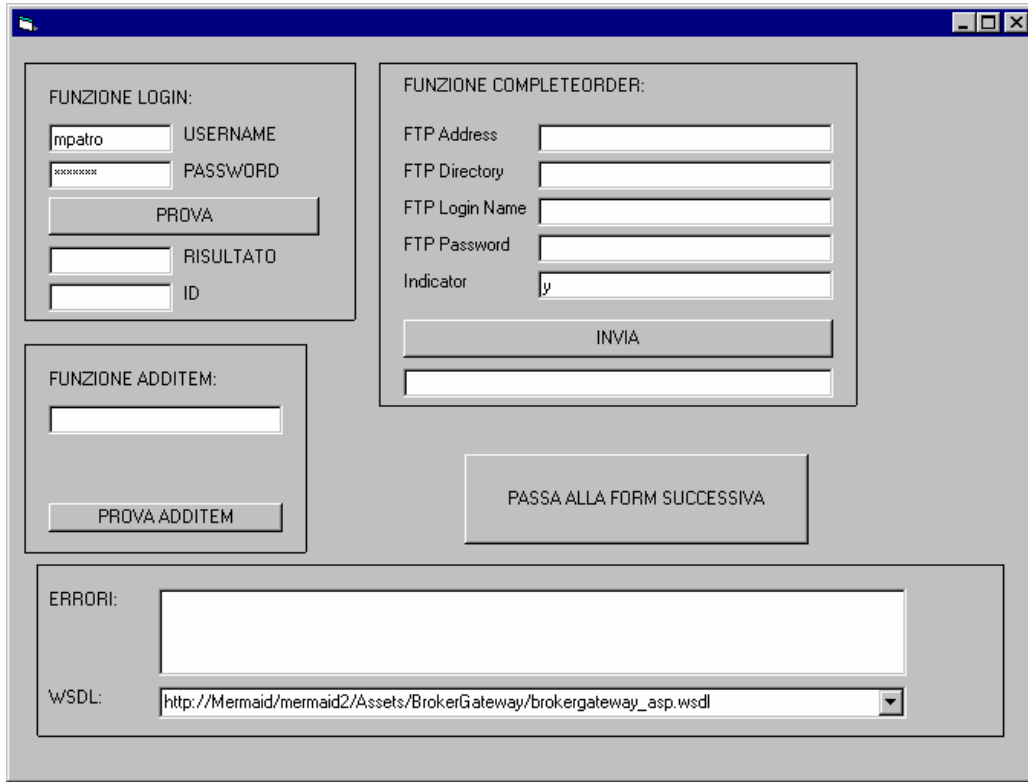


Figure 4: the temporary GUI

With such a system, the tests executed are reported in the next table. Note that testing has been divided into two parts: single methods executions (executed in the right order, but with wrong parameters) and methods executed in the wrong order (with right parameters)

Single methods:

Test	Description	Result
Login	username and/or password empty and/or wrong	returns 0, does not connect to the Data Broker
SearchProduct	any value of the search parameters	always success code (1), search results can be 0 products found as well as the whole catalog
ProductDetails	product not supporting On Line shipping method	returns 0 and the error message that product does not support on-line shipping
AddItem	product not existing or not supporting On Line shipping	returns 0, saying either that the product does not exists or that it does not support on-line



		shipping
CompleteOrder	shipping address not complete (some field missing) while the 'default address' indicator is set to 'N'	returns 0 with a message stating the field that is missing
PayOrder	some payment detail field is missing	returns 0 with a message stating the field that is missing

Wrong methods chaining:

Please, note that the "right" method chaining is defined in paragraph 2.3 of this document. Moreover, in case of a success, method 'GetLastErrors' will always return "OK"

Test	Description	Result
1	Run 'AddItem' immediately after 'Login'	if public method 'pUse' was set to a correct value, this works, otherwise 'AddItem' fails stating that 'use can not be null' (note that 'Use' is usually set when a product is searched)
2	Run 'CompleteOrder' immediately after 'Login'	if the basket was filled with products all supporting On line shipping method, it goes well, otherwise there could be either the message 'your basket is empty' or the message 'some product in your basket does not support on-line shipping method'
3	Run 'ProductDetails' immediately after 'Login'	if public method 'pUse' was set to a correct value, this works (use is necessary in order to compute the price), otherwise 'ProductDetails' fails stating that 'use can not be null' (note that 'Use' is usually set when a product is searched)
4	Run 'PayOrder' immediately after 'Login'	if the basket was filled with products all supporting On line shipping method, and for all of them the shipping address was already set, it goes well, otherwise there could be either the message 'your basket is empty' or the message 'some product in your basket does not support on-line shipping method' or 'some shipping address is missing in your basket'
5	Run any method before 'Login'	always fails, saying "User not logged in"
6	Run any function with an	always fails, reporting "Invalid User ID"



	invalid User ID	
--	-----------------	--

2.5. Future Integration

As already explained, the IF Component will be integrated with two end applications within the project: this will be carried out in WP7

For a future exploitation, we could say that the component could be integrated into whatever application that may need to receive online the datasets available in the MERMAID Data Broker. In fact, the component can be easily integrated with almost all the existing technologies:

- Microsoft technologies (as VC++ or VB): full compatibility
- MS ASP web pages: full compatibility
- Java technology: easy integration using the JNI technology
- Unix, Linux technologies: it is still possible to integrate the component as the communication to and from the Broker protocol is published in this document and it is based on the SOAP protocol

The following sections provide a brief overview of the two end applications. Detailed descriptions of these applications will be provided within Work Package 7.

2.5.1. BMT application

This application will be a simplified web-enabled version of BMT's existing stand-alone Oil Spill Information System (OSIS). The concept is that in the event of an oil spill, an end user will be able to enter the time and position of the spill, together with the type of oil and volume spilt in the application model set-up. The application will then automatically interface with MERMAID to obtain the required metocean data for the region of interest from the broker (i.e. the hydrodynamics and meteorology that drive the model). This data will be in the [2D Modelled data](#) format, as specified in Deliverable D2.2 (Supported Data Structures). The application will then determine the fate and trajectory of the oil spill, and provide results to the end user in the form of a simple graphical output. This will likely show the position of the spill over time on a map, together with a simple graph showing the spill volume over time.

This application will be developed using an integrated solution of existing 3rd Party Web GIS solutions with proprietary model and data serving technology, probably developed using SOAP, C#, SQL Server and C++/COM.



2.5.2. IMGW application

The IMGW application will be developed to demonstrate ways of communicating with the Data Broker and viewing of data. The prototype will be designed to process the data formats specified in Deliverable D2.2 (Supported Data Structures). A special focus will be put on Observed Irregular Timeseries Data, and Point Location Measured Timeseries, which are not used by the BMT emergency response application. The interface of the application prototype will consist of the following components: Communication Manager and Viewer.

The Interface Component will be integrated into the Communication Manager, and it will be responsible for establishing the connection with the Data Broker, searching, buying and downloading of the data.

Data for viewing will be selected from the list of datasets transferred from the Data Broker referring to a particular environmental parameter. Basic information about the data will be displayed from the file header. A user will be able to make displays of data, e.g. spatial distribution of the parameter, time dependence diagram for the parameter at given points, etc. The utilisation of other professional mapping software for presentation of the data will be considered. The prototype will be developed for the WIN32 platform using Visual C++ or Visual Basic.



3. Communication Protocols

3.1. *The Communication Scenario*

Communication within the MERMAID project is required in various parts of the architecture, in particular to let those components that are distributed over the Internet exchange information.

There can be different types of “communication”, and it is important to remember that for many cases proven market technologies and standards already exist.

Summarising the most important of them:

- **Component communication on the same machine:** that means that an application is distributed amongst various components, and these components are all on the same machine. Existing technologies COM (MS), JavaBeans (Java). Note that these technologies do not work in the case of a MS component communicating with a Java component. In this last case more complex solutions should be adopted, such as encapsulating the MS DLL into a JNI-compliant wrapper class, but these are solutions that do not have a generic validity, and they will therefore need to be investigated in each case.
- **Component communication on different machines, within the Intranet:** in this case the components are distributed on more than one machine, but they are all on the same side of the firewall. Existing technologies are: DCOM (MS), CORBA (Java, MS), Enterprise JavaBeans (Java). In the case of communication for components of different technologies, the same considerations explained in the previous point are valid.
- **Component communication across the Internet:** in this case the components are distributed over the internet. Java offers the Servlet to Servlet or Applet to Servlet communication model, while MS does not have a precise solution.

Considering the specific case of MERMAID, we have the following components that need to communicate:

- **Broker Gateway – ECommerce Engine:** in this case both components are developed with MS technology, so MS COM guarantees the communications. There is no need to go into the implementative details of COM/DCOM, that can be found on the Microsoft web site (www.microsoft.com)



- **Broker Gateway – Catalogue Engine:** this case is exactly the same as the previous one: COM technology let the two components communicate
- **Broker Gateway – DAE:** this case is a bit more complex, as the Data Access Engine will be a Java Servlet. The solution adopted will be discussed in the next paragraph
- **DAE – RDAE:** this is a case of a Servlet-to-Servlet communication over the Internet
- **IF Component – Broker Gateway:** this is another complicated case, being a MS to MS communication over the Internet

Summarising the situation, a communication technology and protocol shall be found for a:

- MS to MS over the Internet
- Java to MS on the same machine
- Java to Java over the Internet

It is clear that amongst the various options available, the choice must tend towards those that can solve all the three problems, rather than choosing different technologies for each.

3.1. The SOAP Approach

The Mermaid consortium chose the SOAP protocol as a valid solution to the communications issue.

The following quote from the W3C proposal submission (<http://www.w3.org/tr/soap>) gives a definition of SOAP:

“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 datatypes, and a convention for representing remote procedure calls and responses. SOAP can potentially be used in combination with a variety of other protocols; however, the only bindings defined in this document describe how to use SOAP in combination with HTTP and HTTP Extension Framework.”

The proposal was submitted, and thus it is supported by important vendors on the market such as:

[DevelopMentor](#), [International Business Machines Corporation](#), [Lotus Development Corporation](#), [Microsoft](#), [UserLand Software](#)



It is important to underline the fact that SOAP is vendor-independent, as it is XML-based and it defines the rules to exchange messages, not the technology about how to do it.

In fact, various implementations can be found on the market, in particular:

MS Implementation: Microsoft first release a SOAP Toolkit 2.0 gold Edition (early 2001), then embedded SOAP into its new .NET technology. Therefore SOAP is and will be fully supported by Microsoft (more at: <http://msdn.microsoft.com/soap/>)

Java Implementation: amongst the various options existing, the Apache approach has a very good degree of maturity (<http://xml.apache.org/soap/>)

It is also important to remember two other standards linked with SOAP:

WSDL: this defines HOW to describe a Web service (<http://www.w3.org/TR/wsdl/>):

“WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate, however, the only bindings described in this document describe how to use WSDL in conjunction with SOAP 1.1, HTTP GET/POST, and MIME.”

UDDI: a sort of yellow pages of all the Web services that can be found over the Internet (<http://www.uddi.org/>)

“The Universal Description, Discovery and Integration (UDDI) project is a sweeping industry initiative. The project creates a platform-independent, open framework for describing services, discovering businesses, and integrating business services using the Internet, as well as an operational registry that is available today.

UDDI is the first truly cross-industry effort driven by all major platform and software providers, as well as marketplace operators and e-business leaders. These technology and business pioneers are acting as the initial catalysts to quickly develop UDDI and related technologies.

The UDDI project takes advantage of WorldWide Web Consortium (W3C) and Internet Engineering Task Force (IETF) standards such as Extensible Markup Language (XML), and HTTP and Domain Name System (DNS) protocols. Additionally, cross platform programming features are addressed by adopting



early versions of the proposed Simple Object Access Protocol (SOAP) known as XML Protocol messaging specifications found at the [W3C Web site](#). The UDDI protocol is the building block that will enable businesses to quickly, easily and dynamically find and transact with one another using their preferred applications. Over 220 companies are members of the UDDI [community](#)"

It should be noted that other more well-established options may well have provided alternatives to the SOAP solution. However, it was felt that not only did SOAP provide the most optimum solution, it was also a relatively new technology that the MERMAID consortium felt should be explored in this technological research project.

So how can SOAP / WSDL help MERMAID ?

The answer is very simple. SOAP provides an easy way to let whatever component (both MS or Java) communicate over the Internet, and in particular, thanks to the various implementations, a straight forward way to implement, in effect, a Remote Methods Invocation.

To summarise in detail how SOAP will be implemented:

IF Component to Broker Gateway: communicate with MS SOAP Toolkit. IF Component being a SOAP client and Broker Gateway being a SOAP server.

Broker Gateway to DAE: communicate with MS SOAP Toolkit client on the Broker Gateway, and Apache SOAP Server on the Data Access Engine

DAE to RDAE: communicate with Apache SOAP client (DAE) and server (RDAE)

All the scenarios described have been tested by TXT through the implementation of simple test components implementing the 4 cases:

1. Java SOAP client to Java SOAP Server
2. Java SOAP client to MS SOAP Server
3. MS SOAP client to MS SOAP Server
4. MS SOAP client to Java SOAP Server

As the tests gave excellent results, the decision to adopt SOAP for MERMAID was agreed by the project partners and the implementation began.

3.2. Implementing SOAP in MERMAID

In this section we will analyse in detail how SOAP is now integrated into MERMAID.



3.2.1. IF Component to / from broker Gateway

The IF Component is a SOAP client, as it needs to execute some services that are available on the MERMAID Data Broker.

As explained in the previous section, these services are about the dataset purchasing operations. For each one of the methods described in paragraph 2.2, there is a SOAP message going to the Data Broker, where a SOAP server is in charge to receive it, understand it, execute the required service and then send back the answer to the IF Component.

As we saw before, to describe a Web service, it is necessary to provide a WSDL file, therefore all the services and messages exchanged between these two components can be summarised by the example file detailed in Appendix 1.

3.2.2. DAE to/from Broker Gateway

The DAE and the Broker Gateway have to communicate about dataset shipping: when a user purchases something, the Broker Gateway has to tell the DAE to send (via FTP or e-mail) the purchased files to the shipping addresses selected. In this way, the DAE acts as a SOAP server, while the Broker Gateway is a SOAP client.

As said before, the WSDL entirely describes the Web service offered. Appendix 2 provides an example of the WSDL file required for the DAE communication.

3.2.3. RDAE to / from DAE

The RDAE to DAE communication protocol is Java to Java technology via SOAP on two Java compatible web servers. The XML SOAP messages are decoded into remote procedure calls that facilitate the Initiation of transfers plus the retrieval of status information.



A1. Appendix 1: Example WSDL File for IF Component to / from broker Gateway communication

```

<?xml version='1.0' encoding='UTF-8' ?>
<!-- Generated 09/04/01 by Microsoft SOAP Toolkit WSDL File Generator, Version 1.00.623.1 -->
<definitions name='brokergateway_asp' targetNamespace='http://tempuri.org/wsdl/'
  xmlns:wsdl='http://tempuri.org/wsdl/'
  xmlns:typens='http://tempuri.org/type'
  xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
  xmlns:xsd='http://www.w3.org/2001/XMLSchema'
  xmlns:stk='http://schemas.microsoft.com/soap-toolkit/wsdl-extension'
  xmlns='http://schemas.xmlsoap.org/wsdl/'>
  <types>
    <schema targetNamespace='http://tempuri.org/type'
      xmlns='http://www.w3.org/2001/XMLSchema'
      xmlns:SOAP-ENC='http://schemas.xmlsoap.org/soap/encoding/'
      xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
      elementFormDefault='qualified'>
      <complexType name='ArrayOfstring'>
        <complexContent>
          <restriction base='SOAP-ENC:Array'>
            <attribute ref='SOAP-ENC:arrayType' wsdl:arrayType='string[]' />
          </restriction>
        </complexContent>
      </complexType>
    </schema>
  </types>
  <message name='BrokerGatewayClass.PayOrder'>
    <part name='Billing_address' type='typens:ArrayOfstring' />
    <part name='Payment_method' type='xsd:string' />
    <part name='Payment_param' type='typens:ArrayOfstring' />
    <part name='Receipt_ID' type='xsd:string' />
    <part name='Default_Values' type='xsd:string' />
    <part name='UserID' type='xsd:string' />
    <part name='last_error' type='xsd:string' />
  </message>
  <message name='BrokerGatewayClass.PayOrderResponse'>
    <part name='Result' type='xsd:double' />
    <part name='Billing_address' type='typens:ArrayOfstring' />
    <part name='Payment_method' type='xsd:string' />
    <part name='Payment_param' type='typens:ArrayOfstring' />
    <part name='Receipt_ID' type='xsd:string' />
    <part name='Default_Values' type='xsd:string' />
    <part name='UserID' type='xsd:string' />
    <part name='last_error' type='xsd:string' />
  </message>
  <message name='BrokerGatewayClass.ProductDetails'>
    <part name='ProductSku' type='xsd:string' />
    <part name='pf_id' type='xsd:string' />
    <part name='ProductName' type='xsd:string' />
    <part name='ProductProvider' type='xsd:string' />
    <part name='ProductPrice' type='xsd:string' />
    <part name='DatastreamDescription' type='xsd:string' />
    <part name='pUse' type='xsd:string' />
    <part name='pCoordinates' type='typens:ArrayOfstring' />
    <part name='UserID' type='xsd:string' />
    <part name='last_error' type='xsd:string' />
  </message>
  <message name='BrokerGatewayClass.ProductDetailsResponse'>
    <part name='Result' type='xsd:double' />
    <part name='ProductSku' type='xsd:string' />
    <part name='pf_id' type='xsd:string' />
    <part name='ProductName' type='xsd:string' />
  </message>

```



```

    <part name='ProductProvider' type='xsd:string'/>
    <part name='ProductPrice' type='xsd:string'/>
    <part name='DatastreamDescription' type='xsd:string'/>
    <part name='pUse' type='xsd:string'/>
    <part name='pCoordinates' type='typens:ArrayOfstring'/>
    <part name='UserID' type='xsd:string'/>
    <part name='last_error' type='xsd:string'/>
  </message>
<message name='BrokerGatewayClass.SearchProduct'>
  <part name='Coordinates' type='typens:ArrayOfstring'/>
  <part name='Product_Name' type='xsd:string'/>
  <part name='ProductSku' type='typens:ArrayOfstring'/>
  <part name='UserID' type='xsd:string'/>
  <part name='last_error' type='xsd:string'/>
</message>
<message name='BrokerGatewayClass.SearchProductResponse'>
  <part name='Result' type='xsd:double'/>
  <part name='Coordinates' type='typens:ArrayOfstring'/>
  <part name='Product_Name' type='xsd:string'/>
  <part name='ProductSku' type='typens:ArrayOfstring'/>
  <part name='UserID' type='xsd:string'/>
  <part name='last_error' type='xsd:string'/>
</message>
<message name='BrokerGatewayClass.CompleteOrder'>
  <part name='FTP_Address' type='xsd:string'/>
  <part name='FTP_Dir' type='xsd:string'/>
  <part name='FTP_Login' type='xsd:string'/>
  <part name='FTP_Pwd' type='xsd:string'/>
  <part name='Default_Values' type='xsd:string'/>
  <part name='UserID' type='xsd:string'/>
  <part name='last_error' type='xsd:string'/>
</message>
<message name='BrokerGatewayClass.CompleteOrderResponse'>
  <part name='Result' type='xsd:double'/>
  <part name='FTP_Address' type='xsd:string'/>
  <part name='FTP_Dir' type='xsd:string'/>
  <part name='FTP_Login' type='xsd:string'/>
  <part name='FTP_Pwd' type='xsd:string'/>
  <part name='Default_Values' type='xsd:string'/>
  <part name='UserID' type='xsd:string'/>
  <part name='last_error' type='xsd:string'/>
</message>
<message name='BrokerGatewayClass.AddItem'>
  <part name='ProductSku' type='xsd:string'/>
  <part name='pCoordinates' type='typens:ArrayOfstring'/>
  <part name='pUse' type='xsd:string'/>
  <part name='UserID' type='xsd:string'/>
  <part name='last_error' type='xsd:string'/>
</message>
<message name='BrokerGatewayClass.AddItemResponse'>
  <part name='Result' type='xsd:double'/>
  <part name='ProductSku' type='xsd:string'/>
  <part name='pCoordinates' type='typens:ArrayOfstring'/>
  <part name='pUse' type='xsd:string'/>
  <part name='UserID' type='xsd:string'/>
  <part name='last_error' type='xsd:string'/>
</message>
<message name='BrokerGatewayClass.Login'>
  <part name='usern' type='xsd:string'/>
  <part name='pwd' type='xsd:string'/>
  <part name='UserID' type='xsd:string'/>
  <part name='last_error' type='xsd:string'/>
</message>
<message name='BrokerGatewayClass.LoginResponse'>
  <part name='Result' type='xsd:double'/>
  <part name='usern' type='xsd:string'/>
  <part name='pwd' type='xsd:string'/>
  <part name='UserID' type='xsd:string'/>
  <part name='last_error' type='xsd:string'/>

```



```

</message>
<portType name='BrokerGatewayClassSoapPort'>
  <operation name='PayOrder' parameterOrder='Billing_address Payment_method Payment_param Receipt_ID
Default_Values UserID last_error'>
    <input message='wsdlIns:BrokerGatewayClass.PayOrder' />
    <output message='wsdlIns:BrokerGatewayClass.PayOrderResponse' />
  </operation>
  <operation name='ProductDetails' parameterOrder='ProductSku pf_id ProductName ProductProvider ProductPrice
DataStreamDescription pUse pCoordinates UserID last_error'>
    <input message='wsdlIns:BrokerGatewayClass.ProductDetails' />
    <output message='wsdlIns:BrokerGatewayClass.ProductDetailsResponse' />
  </operation>
  <operation name='SearchProduct' parameterOrder='Coordinates Product_Name ProductSku UserID last_error'>
    <input message='wsdlIns:BrokerGatewayClass.SearchProduct' />
    <output message='wsdlIns:BrokerGatewayClass.SearchProductResponse' />
  </operation>
  <operation name='CompleteOrder' parameterOrder='FTP_Address FTP_Dir FTP_Login FTP_Pwd Default_Values
UserID last_error'>
    <input message='wsdlIns:BrokerGatewayClass.CompleteOrder' />
    <output message='wsdlIns:BrokerGatewayClass.CompleteOrderResponse' />
  </operation>
  <operation name='AddItem' parameterOrder='ProductSku pCoordinates pUse UserID last_error'>
    <input message='wsdlIns:BrokerGatewayClass.AddItem' />
    <output message='wsdlIns:BrokerGatewayClass.AddItemResponse' />
  </operation>
  <operation name='Login' parameterOrder='usern pwd UserID last_error'>
    <input message='wsdlIns:BrokerGatewayClass.Login' />
    <output message='wsdlIns:BrokerGatewayClass.LoginResponse' />
  </operation>
</portType>
<binding name='BrokerGatewayClassSoapBinding' type='wsdlIns:BrokerGatewayClassSoapPort' >
  <stk:binding preferredEncoding='UTF-8'/>
  <soap:binding style='rpc' transport='http://schemas.xmlsoap.org/soap/http' />
  <operation name='PayOrder' >
    <soap:operation soapAction='http://tempuri.org/action/BrokerGatewayClass.PayOrder' />
    <input>
      <soap:body use='encoded' namespace='http://tempuri.org/message/'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
    </input>
    <output>
      <soap:body use='encoded' namespace='http://tempuri.org/message/'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
    </output>
  </operation>
  <operation name='ProductDetails' >
    <soap:operation soapAction='http://tempuri.org/action/BrokerGatewayClass.ProductDetails' />
    <input>
      <soap:body use='encoded' namespace='http://tempuri.org/message/'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
    </input>
    <output>
      <soap:body use='encoded' namespace='http://tempuri.org/message/'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
    </output>
  </operation>
  <operation name='SearchProduct' >
    <soap:operation soapAction='http://tempuri.org/action/BrokerGatewayClass.SearchProduct' />
    <input>
      <soap:body use='encoded' namespace='http://tempuri.org/message/'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
    </input>
    <output>
      <soap:body use='encoded' namespace='http://tempuri.org/message/'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
    </output>
  </operation>
  <operation name='CompleteOrder' >
    <soap:operation soapAction='http://tempuri.org/action/BrokerGatewayClass.CompleteOrder' />
    <input>

```



```
<soap:body use='encoded' namespace='http://tempuri.org/message/'
      encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
</input>
<output>
  <soap:body use='encoded' namespace='http://tempuri.org/message/'
        encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
</output>
</operation>
<operation name='AddItem' >
  <soap:operation soapAction='http://tempuri.org/action/BrokerGatewayClass.AddItem' />
  <input>
    <soap:body use='encoded' namespace='http://tempuri.org/message/'
          encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
  </input>
  <output>
    <soap:body use='encoded' namespace='http://tempuri.org/message/'
          encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
  </output>
</operation>
<operation name='Login' >
  <soap:operation soapAction='http://tempuri.org/action/BrokerGatewayClass.Login' />
  <input>
    <soap:body use='encoded' namespace='http://tempuri.org/message/'
          encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
  </input>
  <output>
    <soap:body use='encoded' namespace='http://tempuri.org/message/'
          encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
  </output>
</operation>
</binding>
<service name='brokergateway_asp' >
  <port name='BrokerGatewayClassSoapPort' binding='wsdl:ns:BrokerGatewayClassSoapBinding' >
    <soap:address location='http://Mermaid/mermaid2/Assets/BrokerGateway/brokergateway_asp.ASP' />
  </port>
</service>
</definitions>
```



A2. Appendix 2: Example WSDL File for the DAE to/from Broker Gateway communication

```
<?xml version='1.0' encoding='UTF-8' ?>
<!-- Generated 09/05/01 by Microsoft SOAP Toolkit WSDL File Generator, Version 1.00.623.1 -->
<definitions name='DAE' targetNamespace='http://tempuri.org/wsdll'
  xmlns:wsdlns='http://tempuri.org/wsdll'
  xmlns:typens='http://tempuri.org/type'
  xmlns:soap='http://schemas.xmlsoap.org/wsdll/soap'
  xmlns:xsd='http://www.w3.org/2001/XMLSchema'
  xmlns:stk='http://schemas.microsoft.com/soap-toolkit/wsdll-extension'
  xmlns='http://schemas.xmlsoap.org/wsdll/'>
  <types>
    <schema targetNamespace='http://tempuri.org/type'
      xmlns='http://www.w3.org/2001/XMLSchema'
      xmlns:SOAP-ENC='http://schemas.xmlsoap.org/soap/encoding'
      xmlns:wsdll='http://schemas.xmlsoap.org/wsdll/'
      elementFormDefault='qualified'>
    </schema>
  </types>
  <message name='DAESimulator.NotifyTransferComplete'>
  </message>
  <message name='DAESimulator.NotifyTransferCompleteResponse'>
    <part name='Result' type='xsd:double'>
  </message>
  <message name='DAESimulator.ReturnExtractionStatus'>
  </message>
  <message name='DAESimulator.ReturnExtractionStatusResponse'>
    <part name='Result' type='xsd:double'>
  </message>
  <message name='DAESimulator.ReturnDeleteStatus'>
  </message>
  <message name='DAESimulator.ReturnDeleteStatusResponse'>
    <part name='Result' type='xsd:double'>
  </message>
  <message name='DAESimulator.ReturnUploadStatus'>
  </message>
  <message name='DAESimulator.ReturnUploadStatusResponse'>
    <part name='Result' type='xsd:double'>
  </message>
  <message name='DAESimulator.ReturnTransferStatus'>
  </message>
  <message name='DAESimulator.ReturnTransferStatusResponse'>
    <part name='Result' type='xsd:double'>
  </message>
  <message name='DAESimulator.ReturnUsedSpace'>
    <part name='providerID' type='xsd:anyType'>
  </message>
  <message name='DAESimulator.ReturnUsedSpaceResponse'>
    <part name='Result' type='xsd:double'>
    <part name='providerID' type='xsd:anyType'>
  </message>
  <message name='DAESimulator.DeleteDataset'>
    <part name='datasetID' type='xsd:anyType'>
  </message>
  <message name='DAESimulator.DeleteDatasetResponse'>
    <part name='Result' type='xsd:double'>
    <part name='datasetID' type='xsd:anyType'>
  </message>
  <message name='DAESimulator.EmailDataset'>
    <part name='datasetID' type='xsd:anyType'>
    <part name='emailAddress' type='xsd:anyType'>
  </message>
  <message name='DAESimulator.EmailDatasetResponse'>
    <part name='Result' type='xsd:double'>
```



```

    <part name='datasetID' type='xsd:anyType'/>
    <part name='emailAddress' type='xsd:anyType'/>
  </message>
<message name='DAESimulator.EmailSubset'>
  <part name='datasetID' type='xsd:anyType'/>
  <part name='emailAddress' type='xsd:anyType'/>
  <part name='subsetParameters' type='xsd:anyType'/>
</message>
<message name='DAESimulator.EmailSubsetResponse'>
  <part name='Result' type='xsd:double'/>
  <part name='datasetID' type='xsd:anyType'/>
  <part name='emailAddress' type='xsd:anyType'/>
  <part name='subsetParameters' type='xsd:anyType'/>
</message>
<message name='DAESimulator.UploadDataset'>
  <part name='providerID' type='xsd:anyType'/>
  <part name='directoryLocation' type='xsd:anyType'/>
  <part name='filename' type='xsd:anyType'/>
</message>
<message name='DAESimulator.UploadDatasetResponse'>
  <part name='Result' type='xsd:double'/>
  <part name='providerID' type='xsd:anyType'/>
  <part name='directoryLocation' type='xsd:anyType'/>
  <part name='filename' type='xsd:anyType'/>
</message>
<message name='DAESimulator.TransferSubset'>
  <part name='datasetID' type='xsd:anyType'/>
  <part name='DeliveryIP' type='xsd:anyType'/>
  <part name='DeliveryDirectory' type='xsd:anyType'/>
  <part name='subsetParameters' type='xsd:anyType'/>
</message>
<message name='DAESimulator.TransferSubsetResponse'>
  <part name='Result' type='xsd:double'/>
  <part name='datasetID' type='xsd:anyType'/>
  <part name='DeliveryIP' type='xsd:anyType'/>
  <part name='DeliveryDirectory' type='xsd:anyType'/>
  <part name='subsetParameters' type='xsd:anyType'/>
</message>
<message name='DAESimulator.TransferDataset'>
  <part name='datasetID' type='xsd:anyType'/>
  <part name='DeliveryIP' type='xsd:anyType'/>
  <part name='DeliveryDirectory' type='xsd:anyType'/>
</message>
<message name='DAESimulator.TransferDatasetResponse'>
  <part name='Result' type='xsd:double'/>
  <part name='datasetID' type='xsd:anyType'/>
  <part name='DeliveryIP' type='xsd:anyType'/>
  <part name='DeliveryDirectory' type='xsd:anyType'/>
</message>
<portType name='DAESimulatorSoapPort'>
  <operation name='NotifyTransferComplete' parameterOrder="">
    <input message='wsdlIns:DAESimulator.NotifyTransferComplete' />
    <output message='wsdlIns:DAESimulator.NotifyTransferCompleteResponse' />
  </operation>
  <operation name='ReturnExtractionStatus' parameterOrder="">
    <input message='wsdlIns:DAESimulator.ReturnExtractionStatus' />
    <output message='wsdlIns:DAESimulator.ReturnExtractionStatusResponse' />
  </operation>
  <operation name='ReturnDeleteStatus' parameterOrder="">
    <input message='wsdlIns:DAESimulator.ReturnDeleteStatus' />
    <output message='wsdlIns:DAESimulator.ReturnDeleteStatusResponse' />
  </operation>
  <operation name='ReturnUploadStatus' parameterOrder="">
    <input message='wsdlIns:DAESimulator.ReturnUploadStatus' />
    <output message='wsdlIns:DAESimulator.ReturnUploadStatusResponse' />
  </operation>
  <operation name='ReturnTransferStatus' parameterOrder="">
    <input message='wsdlIns:DAESimulator.ReturnTransferStatus' />
    <output message='wsdlIns:DAESimulator.ReturnTransferStatusResponse' />
  </operation>

```



```

</operation>
<operation name='ReturnUsedSpace' parameterOrder='providerID'>
  <input message='wsdlIns:DAESimulator.ReturnUsedSpace' />
  <output message='wsdlIns:DAESimulator.ReturnUsedSpaceResponse' />
</operation>
<operation name='DeleteDataset' parameterOrder='datasetID'>
  <input message='wsdlIns:DAESimulator.DeleteDataset' />
  <output message='wsdlIns:DAESimulator.DeleteDatasetResponse' />
</operation>
<operation name='EmailDataset' parameterOrder='datasetID emailAddress'>
  <input message='wsdlIns:DAESimulator.EmailDataset' />
  <output message='wsdlIns:DAESimulator.EmailDatasetResponse' />
</operation>
<operation name='EmailSubset' parameterOrder='datasetID emailAddress subsetParameters'>
  <input message='wsdlIns:DAESimulator.EmailSubset' />
  <output message='wsdlIns:DAESimulator.EmailSubsetResponse' />
</operation>
<operation name='UploadDataset' parameterOrder='providerID directoryLocation filename'>
  <input message='wsdlIns:DAESimulator.UploadDataset' />
  <output message='wsdlIns:DAESimulator.UploadDatasetResponse' />
</operation>
<operation name='TransferSubset' parameterOrder='datasetID DeliveryIP DeliveryDirectory subsetParameters'>
  <input message='wsdlIns:DAESimulator.TransferSubset' />
  <output message='wsdlIns:DAESimulator.TransferSubsetResponse' />
</operation>
<operation name='TransferDataset' parameterOrder='datasetID DeliveryIP DeliveryDirectory'>
  <input message='wsdlIns:DAESimulator.TransferDataset' />
  <output message='wsdlIns:DAESimulator.TransferDatasetResponse' />
</operation>
</portType>
<binding name='DAESimulatorSoapBinding' type='wsdlIns:DAESimulatorSoapPort' >
  <stk:binding preferredEncoding='UTF-8'>
  <soap:binding style='rpc' transport='http://schemas.xmlsoap.org/soap/http' />
  <operation name='NotifyTransferComplete' >
    <soap:operation soapAction='http://tempuri.org/action/DAESimulator.NotifyTransferComplete' />
    <input>
      <soap:body use='encoded' namespace='http://tempuri.org/message/'
        encodingStyle='http://schemas.xmlsoap.org/soap/encoding' />
    </input>
    <output>
      <soap:body use='encoded' namespace='http://tempuri.org/message/'
        encodingStyle='http://schemas.xmlsoap.org/soap/encoding' />
    </output>
  </operation>
  <operation name='ReturnExtractionStatus' >
    <soap:operation soapAction='http://tempuri.org/action/DAESimulator.ReturnExtractionStatus' />
    <input>
      <soap:body use='encoded' namespace='http://tempuri.org/message/'
        encodingStyle='http://schemas.xmlsoap.org/soap/encoding' />
    </input>
    <output>
      <soap:body use='encoded' namespace='http://tempuri.org/message/'
        encodingStyle='http://schemas.xmlsoap.org/soap/encoding' />
    </output>
  </operation>
  <operation name='ReturnDeleteStatus' >
    <soap:operation soapAction='http://tempuri.org/action/DAESimulator.ReturnDeleteStatus' />
    <input>
      <soap:body use='encoded' namespace='http://tempuri.org/message/'
        encodingStyle='http://schemas.xmlsoap.org/soap/encoding' />
    </input>
    <output>
      <soap:body use='encoded' namespace='http://tempuri.org/message/'
        encodingStyle='http://schemas.xmlsoap.org/soap/encoding' />
    </output>
  </operation>
  <operation name='ReturnUploadStatus' >
    <soap:operation soapAction='http://tempuri.org/action/DAESimulator.ReturnUploadStatus' />
    <input>

```



```

    <soap:body use='encoded' namespace='http://tempuri.org/message/'
      encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
  </input>
  <output>
    <soap:body use='encoded' namespace='http://tempuri.org/message/'
      encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
  </output>
</operation>
<operation name='ReturnTransferStatus' >
  <soap:operation soapAction='http://tempuri.org/action/DAESimulator.ReturnTransferStatus' />
  <input>
    <soap:body use='encoded' namespace='http://tempuri.org/message/'
      encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
  </input>
  <output>
    <soap:body use='encoded' namespace='http://tempuri.org/message/'
      encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
  </output>
</operation>
<operation name='ReturnUsedSpace' >
  <soap:operation soapAction='http://tempuri.org/action/DAESimulator.ReturnUsedSpace' />
  <input>
    <soap:body use='encoded' namespace='http://tempuri.org/message/'
      encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
  </input>
  <output>
    <soap:body use='encoded' namespace='http://tempuri.org/message/'
      encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
  </output>
</operation>
<operation name='DeleteDataset' >
  <soap:operation soapAction='http://tempuri.org/action/DAESimulator.DeleteDataset' />
  <input>
    <soap:body use='encoded' namespace='http://tempuri.org/message/'
      encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
  </input>
  <output>
    <soap:body use='encoded' namespace='http://tempuri.org/message/'
      encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
  </output>
</operation>
<operation name='EmailDataset' >
  <soap:operation soapAction='http://tempuri.org/action/DAESimulator.EmailDataset' />
  <input>
    <soap:body use='encoded' namespace='http://tempuri.org/message/'
      encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
  </input>
  <output>
    <soap:body use='encoded' namespace='http://tempuri.org/message/'
      encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
  </output>
</operation>
<operation name='EmailSubset' >
  <soap:operation soapAction='http://tempuri.org/action/DAESimulator.EmailSubset' />
  <input>
    <soap:body use='encoded' namespace='http://tempuri.org/message/'
      encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
  </input>
  <output>
    <soap:body use='encoded' namespace='http://tempuri.org/message/'
      encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
  </output>
</operation>
<operation name='UploadDataset' >
  <soap:operation soapAction='http://tempuri.org/action/DAESimulator.UploadDataset' />
  <input>
    <soap:body use='encoded' namespace='http://tempuri.org/message/'
      encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
  </input>

```



```
<output>
  <soap:body use='encoded' namespace='http://tempuri.org/message/'
    encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
</output>
</operation>
<operation name='TransferSubset' >
  <soap:operation soapAction='http://tempuri.org/action/DAESimulator.TransferSubset' />
  <input>
    <soap:body use='encoded' namespace='http://tempuri.org/message/'
      encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
  </input>
  <output>
    <soap:body use='encoded' namespace='http://tempuri.org/message/'
      encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
  </output>
</operation>
<operation name='TransferDataset' >
  <soap:operation soapAction='http://tempuri.org/action/DAESimulator.TransferDataset' />
  <input>
    <soap:body use='encoded' namespace='http://tempuri.org/message/'
      encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
  </input>
  <output>
    <soap:body use='encoded' namespace='http://tempuri.org/message/'
      encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
  </output>
</operation>
</binding>
<service name='DAE' >
  <port name='DAESimulatorSoapPort' binding='wsdl:DAESimulatorSoapBinding' >
    <soap:address location='http://Mermaid/mermaid2/Assets/Simulatore/DAE.ASP' />
  </port>
</service>
</definitions>
```