US 20100146396A1

(19) **United States**

(12) **Patent Application Publication**

(10) **Pub. No.: US 2010/0146396 A1**

(43) **Pub. Date: Jun. 10, 2010**

**Able et al.**

(54) **SYSTEMS AND METHODS FOR WEB SERVICE FUNCTION, DEFINITION, IMPLEMENTATION, AND/OR EXECUTION**

(75) Inventors: **Steve L. Able**, Canton, GA (US); **Roland S. Martin**, Marietta, GA (US)

Correspondence Address:
**BARNES & THORNBURG LLP**
**Suite 1150, 3343 Peachtree Road, N.E.**
**Atlanta, GA 30326-1428 (US)**

(73) Assignee: **GT Software, Inc.**, Atlanta, GA (US)

(21) Appl. No.: **12/704,124**

(22) Filed: **Feb. 11, 2010**

**Related U.S. Application Data**

(63) Continuation of application No. 11/129,597, filed on May 13, 2005, now Pat. No. 7,665,064.

(60) Provisional application No. 60/571,652, filed on May 14, 2004.

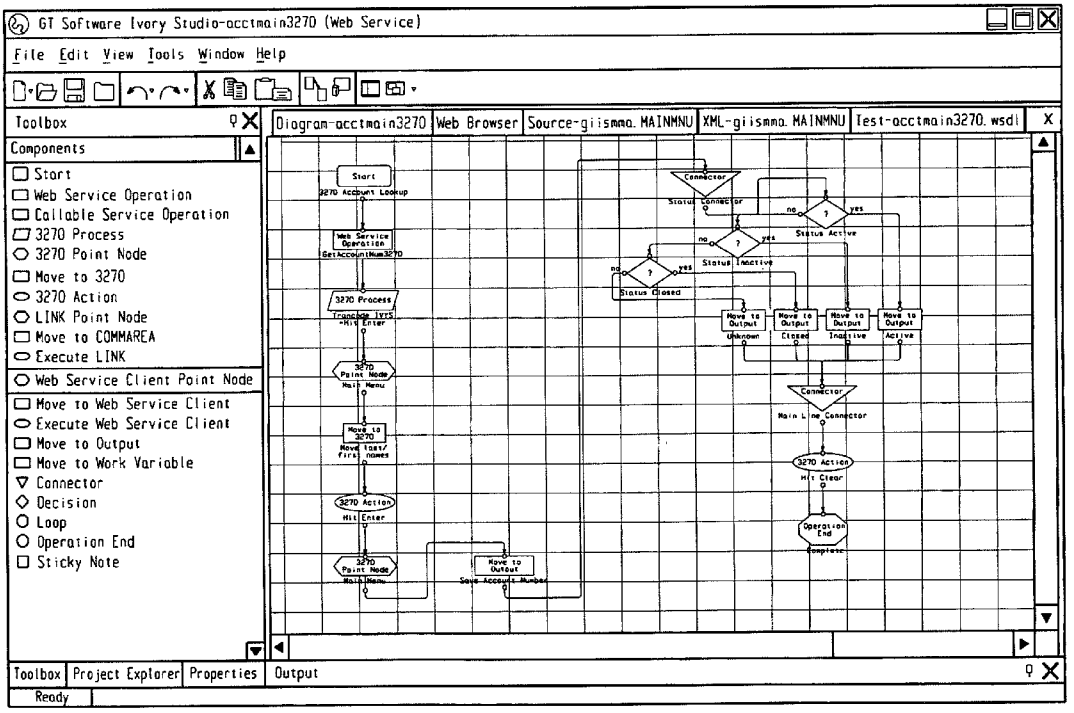
**Publication Classification**

(51) **Int. Cl.**
*G06F 15/16* (2006.01)
*G06F 3/048* (2006.01)

(52) **U.S. Cl.** ..... **715/735**

(57) **ABSTRACT**

Systems and methods for defining, implementing, deploying and/or executing Web services. A Web service is defined using a graphical interface that allows the application developer to specify the operation of the Web service through inclusion and configuration of graphical nodes representing particular operations and/or functions. The Web service definition can, in some instances, include standard functions and/or previously defined Web services. The environment for definition typically includes a user input device, a graphical display device, a system data store and a system processor. The system processor is in communication with the other elements. The user defines the Web service using an interface provided via the graphical display device. The user interacts with the interface using the user input device. The system processor receives the information from the user input device and updates the provided interface on the graphical display device accordingly. The graphical definition of the Web services is stored at least ephemerally in the system data store, but may, in some instances, be saved to the system data store. Once defined, the graphical definition is converted into a programmatic implementation executable by a Web services server. This programmatic implementation can then be transmitted to an appropriate Web services server accessible by an intended user community.



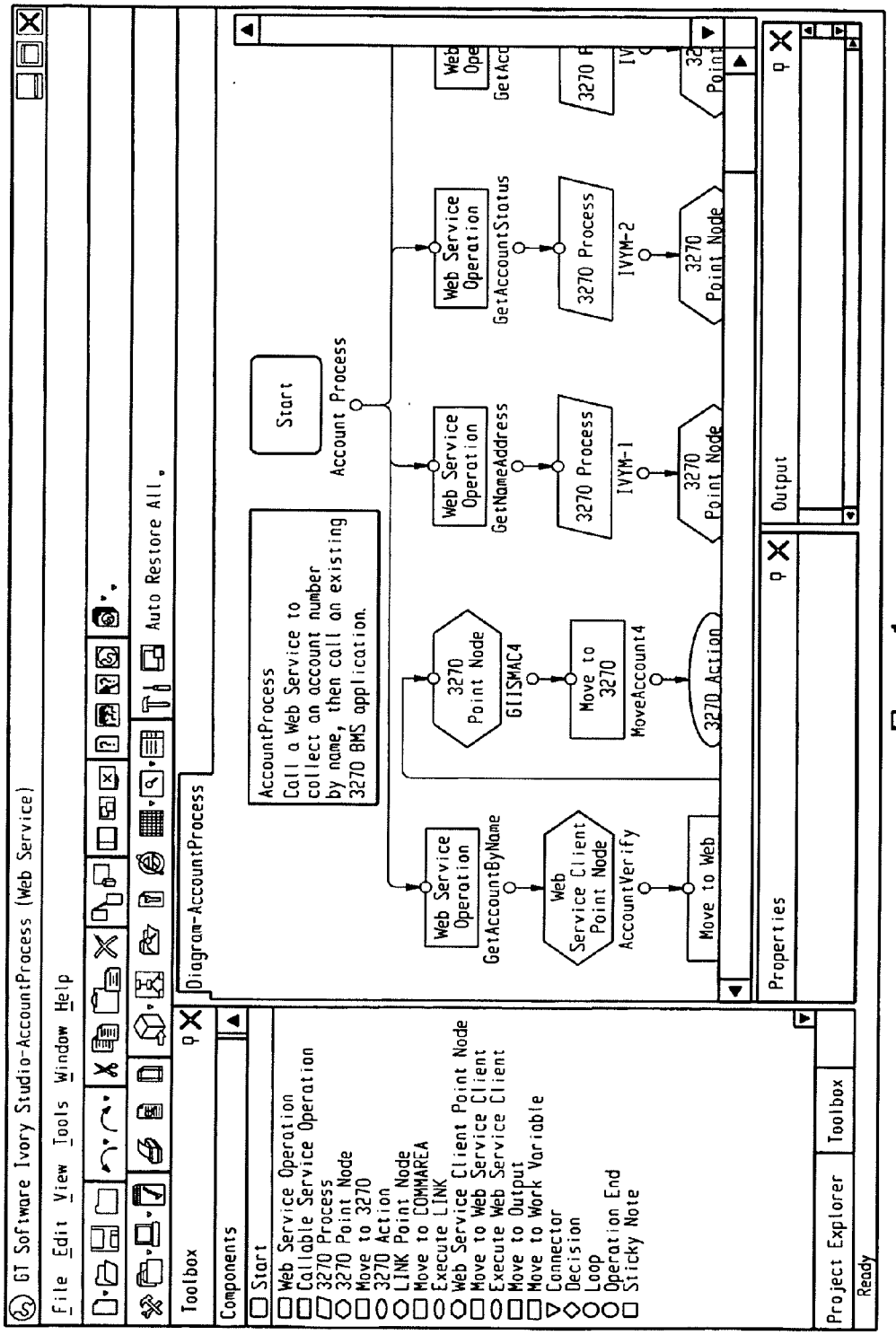

GT Software Ivory Studio-AccountProcess (Web Service)
File Edit View Tools Window Help
Auto Restore All
Toolbox
Components
Start
Web Service Operation
Callable Service Operation
3270 Process
3270 Point Node
Move to 3270
3270 Action
LINK Point Node
Move to COMMAREA
Execute LINK
Web Service Client Point Node
Move to Web Service Client
Execute Web Service Client
Move to Output
Move to Work Variable
Connector
Decision
Loop
Operation End
Sticky Note
Project Explorer
Toolbox
Ready
Diagram-AccountProcess
AccountProcess
Call a Web Service to collect an account number by name, then call an existing 3270 BMS application.
Start
Account Process
Web Service Operation
GetAccountByName
Web Service Client Point Node
AccountVerify
Move to Web
3270 Point Node
GIISMAC4
Move to 3270
MoveAccount4
3270 Action
Web Service Operation
GetNameAddress
3270 Process
IVYM-1
3270 Point Node
Web Service Operation
GetAccountStatus
3270 Process
IVYM-2
3270 Point Node
Properties
Output

Fig. 1

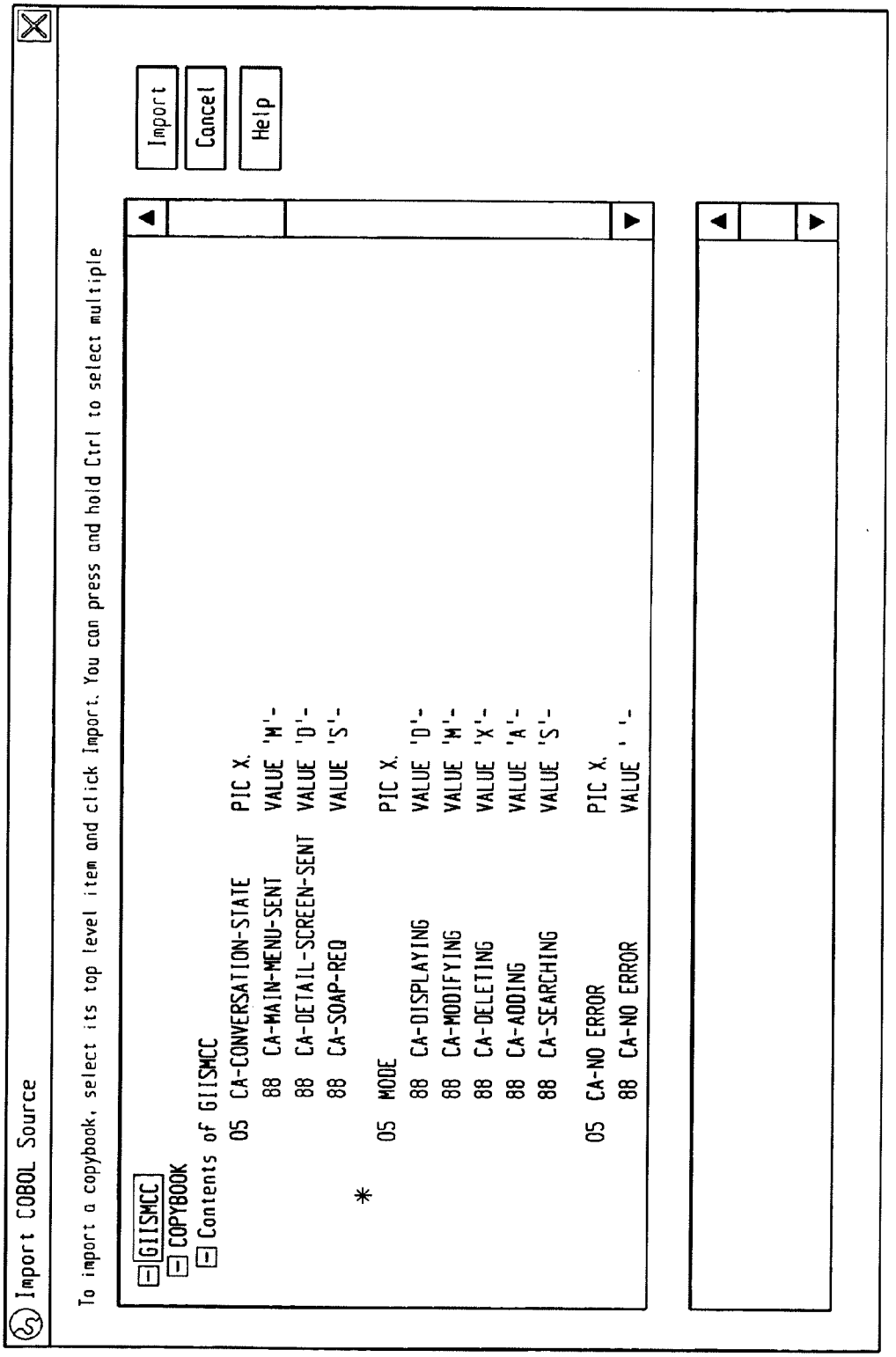

Import COBOL Source
To import a copybook, select its top level item and click Import. You can press and hold Ctrl to select multiple
Import
Cancel
Help
GIISMCC
COPYBOOK
Contents of GIISMCC
05 CA-CONVERSATION-STATE PIC X.
88 CA-MAIN-MENU-SENT VALUE 'M'-
88 CA-DETAIL-SCREEN-SENT VALUE 'D'-
88 CA-SOAP-REQ VALUE 'S'-
*
05 MODE PIC X.
88 CA-DISPLAYING VALUE 'D'-
88 CA-MODIFYING VALUE 'M'-
88 CA-DELETING VALUE 'X'-
88 CA-ADDING VALUE 'A'-
88 CA-SEARCHING VALUE 'S'-
05 CA-NO ERROR PIC X.
88 CA-NO ERROR VALUE ' '-

Fig. 2

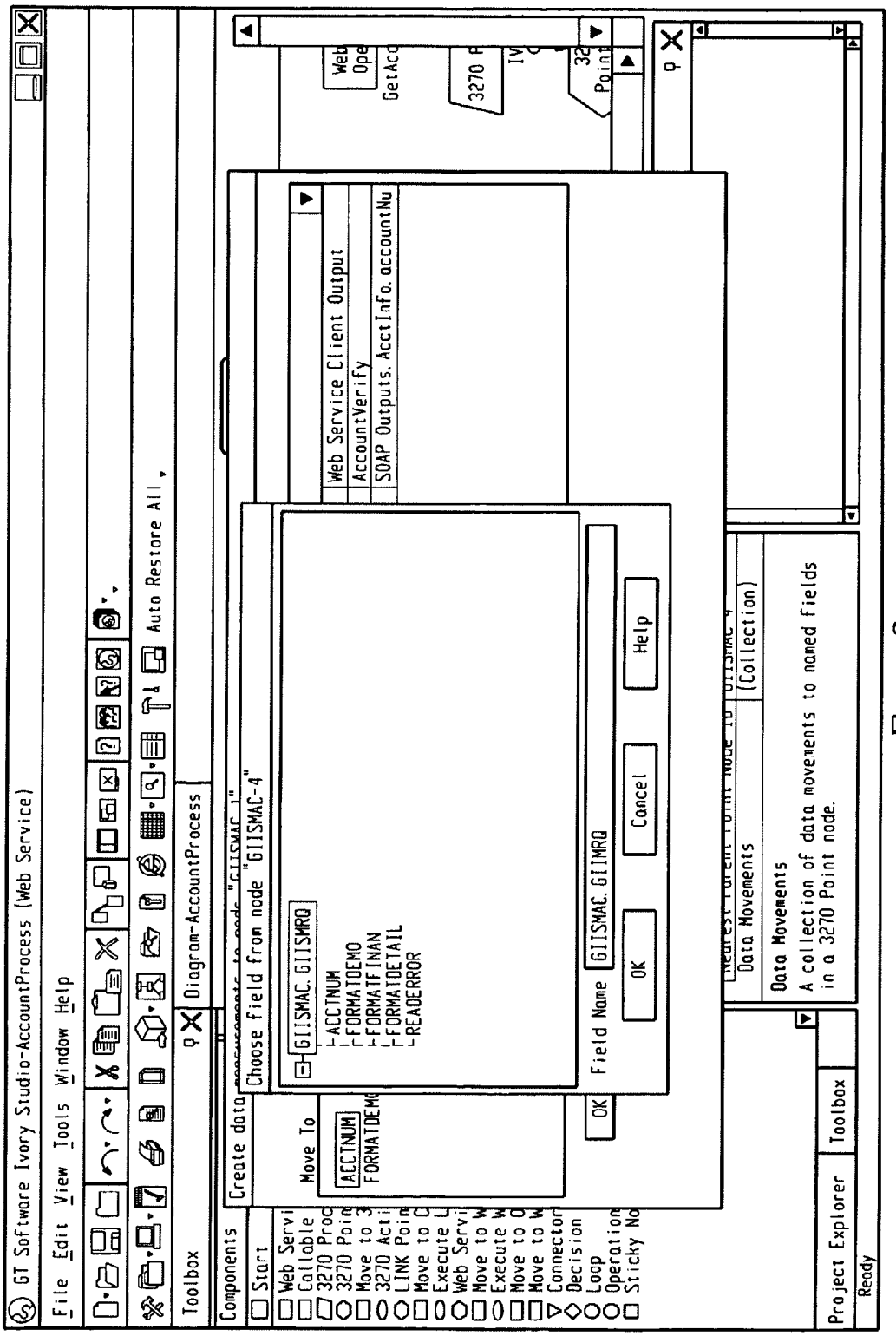
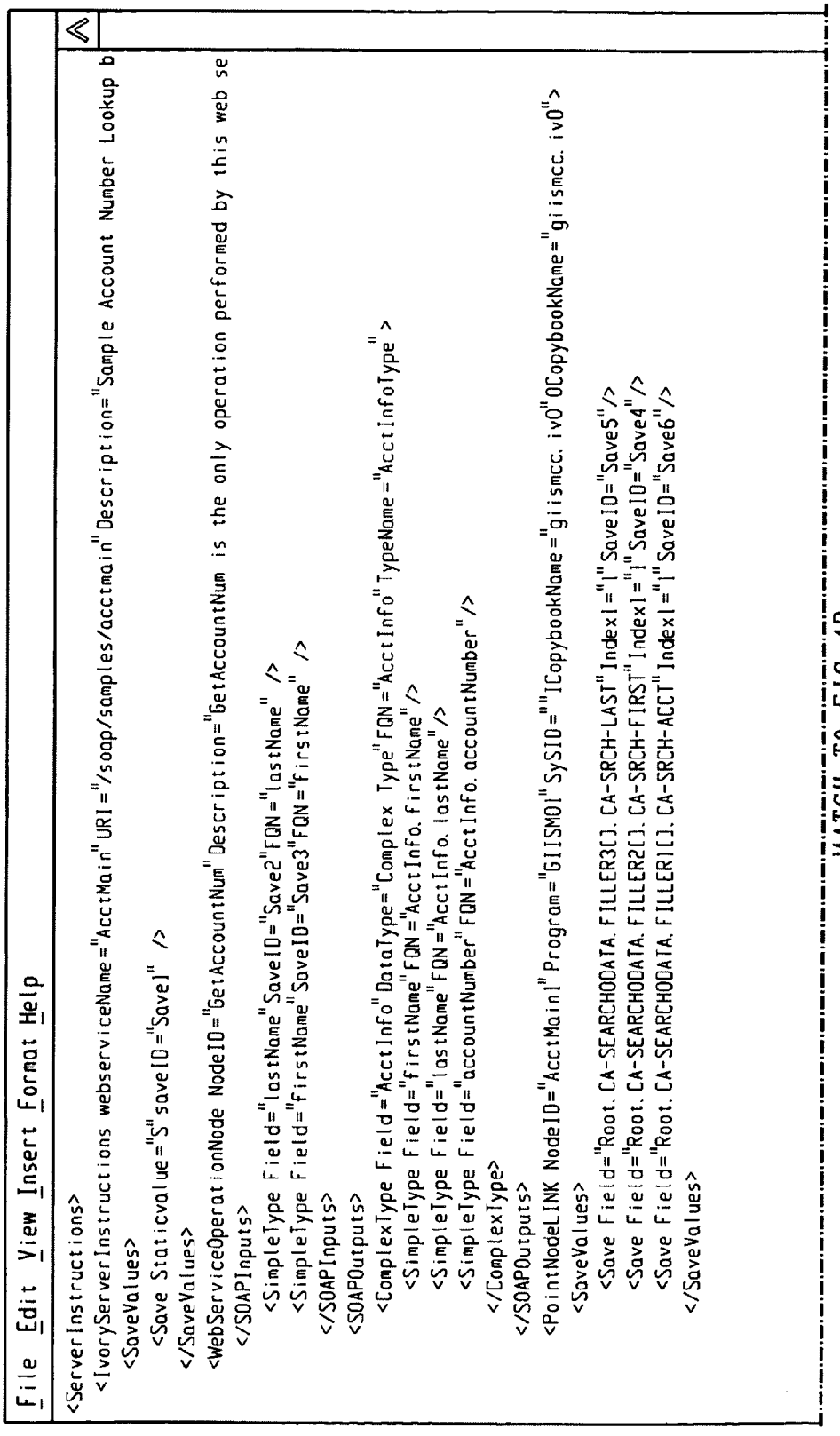

GT Software Ivory Studio-AccountProcess (Web Service)
File Edit View Tools Window Help
Auto Restore All
Toolbox
Diagram-AccountProcess
Components
Start
Web Servi
Callable
3270 Proc
3270 Poin
Move to 3
3270 Acti
LINK Poin
Move to C
Execute L
Web Servi
Move to W
Execute W
Move to C
Move to W
Connector
Decision
Loop
Operation
Sticky No
Create data
Move To
ACCTNUM
FORMATDEMO
Choose field from node "GIISMAC-4"
GIISMAC.GIISMRQ
ACCTNUM
FORMATDEMO
FORMATFINAN
FORMATDETAIL
READERROR
OK
Field Name
GIISMAC.GIIMRQ
OK
Cancel
Help
Web Service Client Output
AccountVerify
SOAP Outputs.AcctInfo.accountNu
Web
Ope
GetAcc
3270 F
IV
32
Point
Data Movements
(Collection)
Data Movements
A collection of data movements to named Fields in a 3270 Point node.
Project Explorer
Toolbox
Ready

Fig. 3

Fig. 4A



File Edit View Insert Format Help

```
<ServerInstructions>
  <IvoryServerInstructions webserviceName="AcctMain" URI="/soap/samples/acctmain" Description="Sample Account Number Lookup b
    <SaveValues>
      <Save StaticValue="S" saveID="Save1" />
    </SaveValues>
    <WebServiceOperationNode NodeID="GetAccountNum" Description="GetAccountNum is the only operation performed by this web se
      </SOAPInputs>
        <SimpleType Field="lastName" SaveID="Save2" FQN="lastName" />
        <SimpleType Field="FirstName" SaveID="Save3" FQN="FirstName" />
      </SOAPInputs>
      <SOAPOutputs>
        <ComplexType Field="AcctInfo" DataType="Complex Type" FQN="AcctInfo" TypeName="AcctInfoType" >
          <SimpleType Field="firstName" FQN="AcctInfo.firstName" />
          <SimpleType Field="lastName" FQN="AcctInfo.lastName" />
          <SimpleType Field="accountNumber" FQN="AcctInfo.accountNumber" />
        </ComplexType>
      </SOAPOutputs>
      <PointNodeLINK NodeID="AcctMain1" Program="GIISMOI" SySID="" "ICopybookName="giismcc. ivO" OCopybookName="giismcc. ivO">
        <SaveValues>
          <Save Field="Root. CA-SEARCHODATA. FILLER3[]. CA-SRCH-LAST" IndexI="|" SaveID="Save5" />
          <Save Field="Root. CA-SEARCHODATA. FILLER2[]. CA-SRCH-FIRST" IndexI="1" SaveID="Save4" />
          <Save Field="Root. CA-SEARCHODATA. FILLER1[]. CA-SRCH-ACCT" IndexI="|" SaveID="Save6" />
        </SaveValues>
```

MATCH TO FIG. 4B

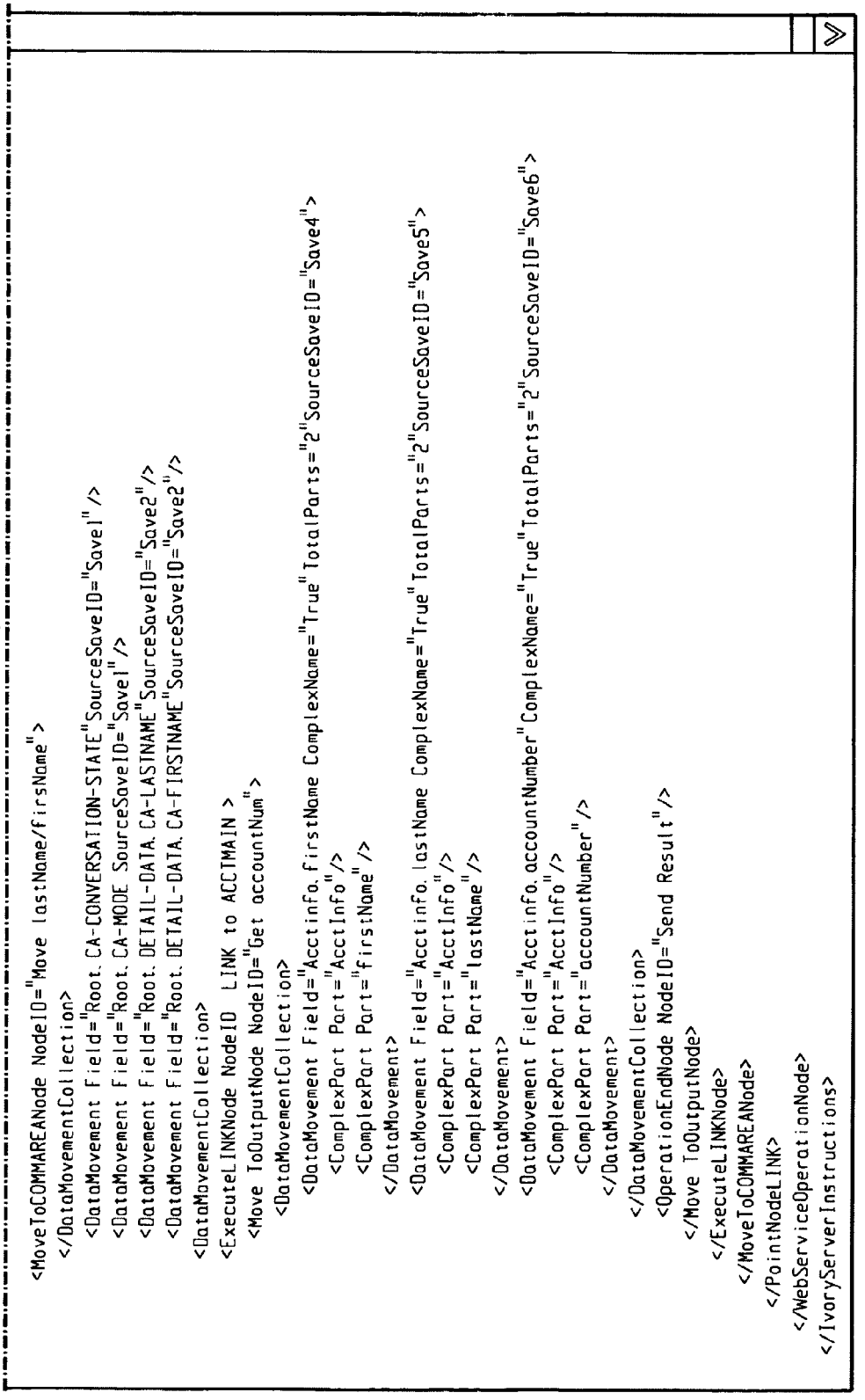

MATCH TO FIG. 4A
<MoveToCOMMAREANode NodeID="Move lastName/firsName">
</DataMovementCollection>
<DataMovement Field="Root. CA-CONVERSATION-STATE" SourceSaveID="Save1" />
<DataMovement Field="Root. CA-MODE SourceSaveID="Save1" />
<DataMovement Field="Root. DETAIL-DATA. CA-LASTNAME" SourceSaveID="Save2" />
<DataMovement Field="Root. DETAIL-DATA. CA-FIRSTNAME" SourceSaveID="Save2" />
<DataMovementCollection>
<ExecuteLINKNode NodeID LINK to ACCTMAIN >
<Move ToOutputNode NodeID="Get accountNum">
<DataMovementCollection>
<DataMovement Field="Acctinfo. firstName ComplexName="True" TotalParts="2" SourceSaveID="Save4">
<ComplexPart Part="AcctInfo"/>
<ComplexPart Part="firstName" />
</DataMovement>
<DataMovement Field="Acctinfo. lastName ComplexName="True" TotalParts="2" SourceSaveID="Save5" >
<ComplexPart Part="AcctInfo"/>
<ComplexPart Part="lastName"/>
</DataMovement>
<DataMovement Field="Acctinfo. accountNumber" ComplexName="True" TotalParts="2" SourceSaveID="Save6" >
<ComplexPart Part="AcctInfo"/>
<ComplexPart Part="accountNumber" />
</DataMovement>
</DataMovementCollection>
<OperationEndNode NodeID="Send Result"/>
</Move ToOutputNode>
</ExecuteLINKNode>
</MoveToCOMMAREANode>
</PointNodeLINK>
</WebServiceOperationNode>
</IvoryServerInstructions>

Fig. 4B

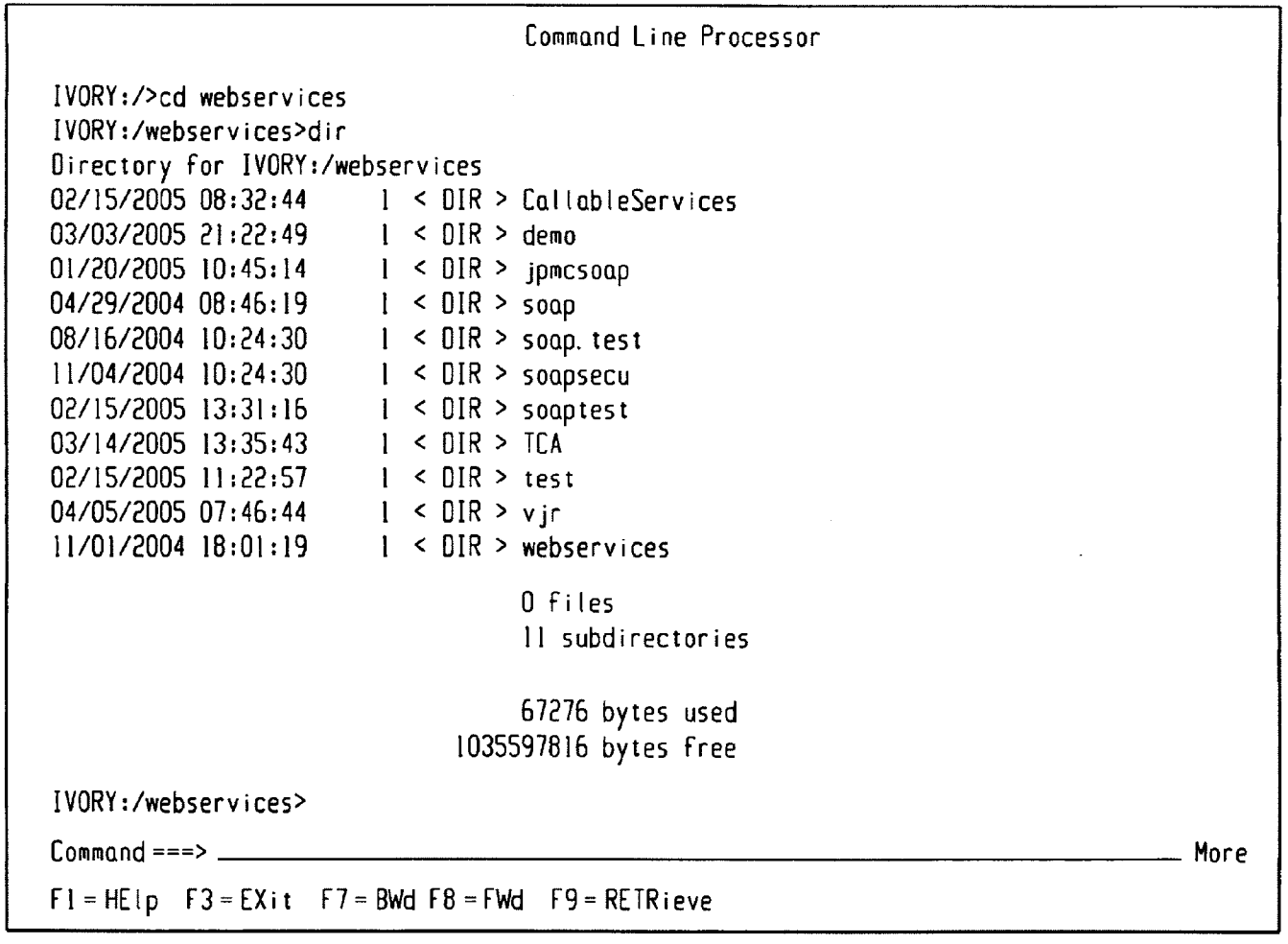

Command Line Processor
IVORY:/>cd webservices
IVORY:/webservices>dir
Directory for IVORY:/webservices
02/15/2005 08:32:44 1 < DIR > CallableServices
03/03/2005 21:22:49 1 < DIR > demo
01/20/2005 10:45:14 1 < DIR > jpmcsoap
04/29/2004 08:46:19 1 < DIR > soap
08/16/2004 10:24:30 1 < DIR > soap.test
11/04/2004 10:24:30 1 < DIR > soapsecu
02/15/2005 13:31:16 1 < DIR > soaptest
03/14/2005 13:35:43 1 < DIR > TCA
02/15/2005 11:22:57 1 < DIR > test
04/05/2005 07:46:44 1 < DIR > vjr
11/01/2004 18:01:19 1 < DIR > webservices
0 files
11 subdirectories
67276 bytes used
1035597816 bytes free
IVORY:/webservices>
Command ===>
More
F1 = HElp F3 = EXit F7 = BWd F8 = FWd F9 = RETRieve

Fig. 5

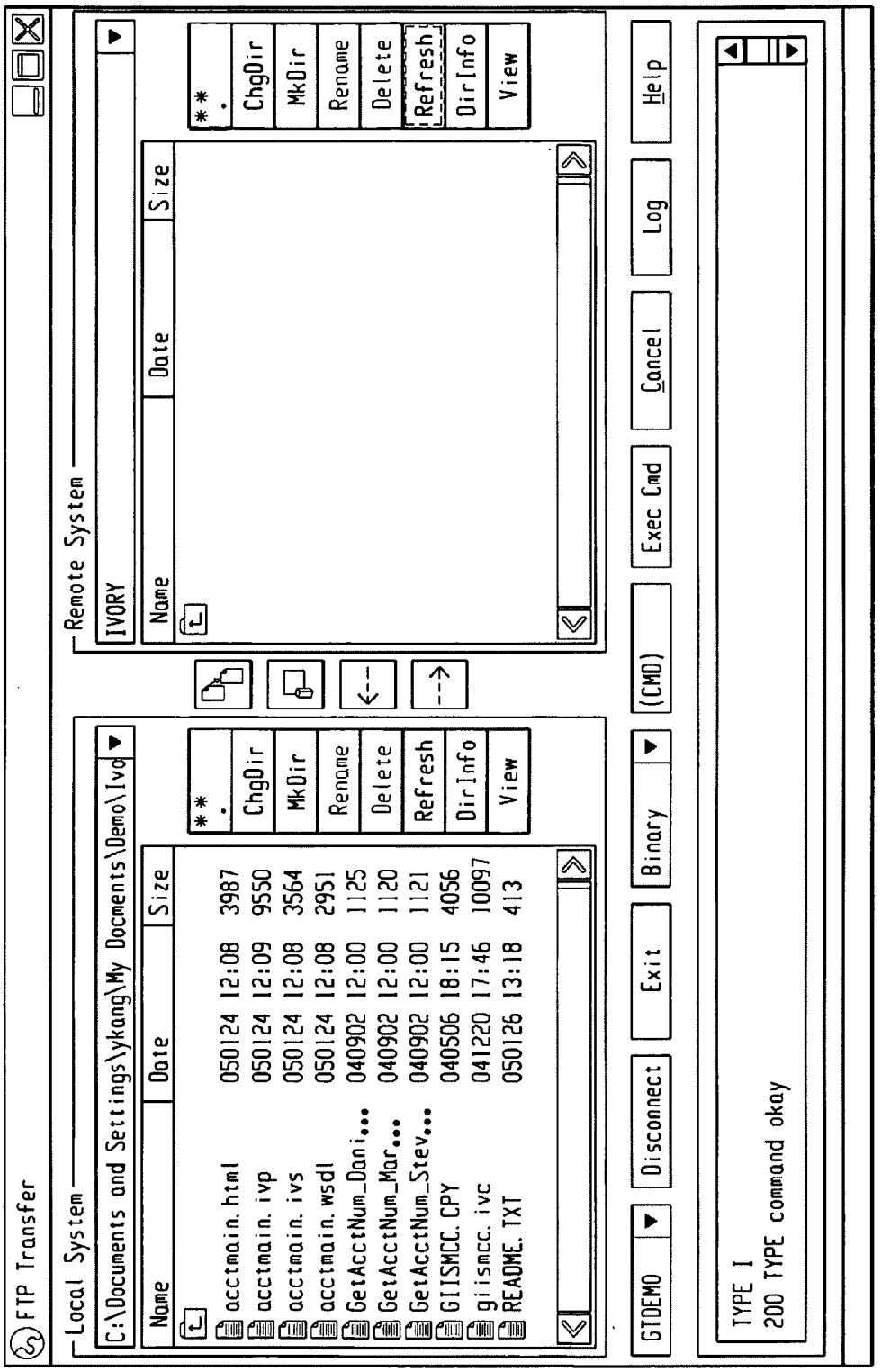

FTP Transfer
Local System
C:\Documents and Settings\ykang\My Documents\Demo\Ivp
Name
Date
Size
acctmain.html 050124 12:08 3987
acctmain.ivp 050124 12:09 9550
acctmain.ivs 050124 12:08 3564
acctmain.wsdl 050124 12:08 2951
GetAcctNum_Dani... 040902 12:00 1125
GetAcctNum_Mar... 040902 12:00 1120
GetAcctNum_Stev... 040902 12:00 1121
GIISMCC.CPY 040506 18:15 4056
giismcc.ivc 041220 17:46 10097
README.TXT 050126 13:18 413
*.*
ChgDir
MkDir
Rename
Delete
Refresh
DirInfo
View
Remote System
IVORY
Name
Date
Size
GTDEMO
Disconnect
Exit
Binary
(CMD)
Exec Cmd
Cancel
Log
Help
TYPE I
200 TYPE command okay

Fig. 6

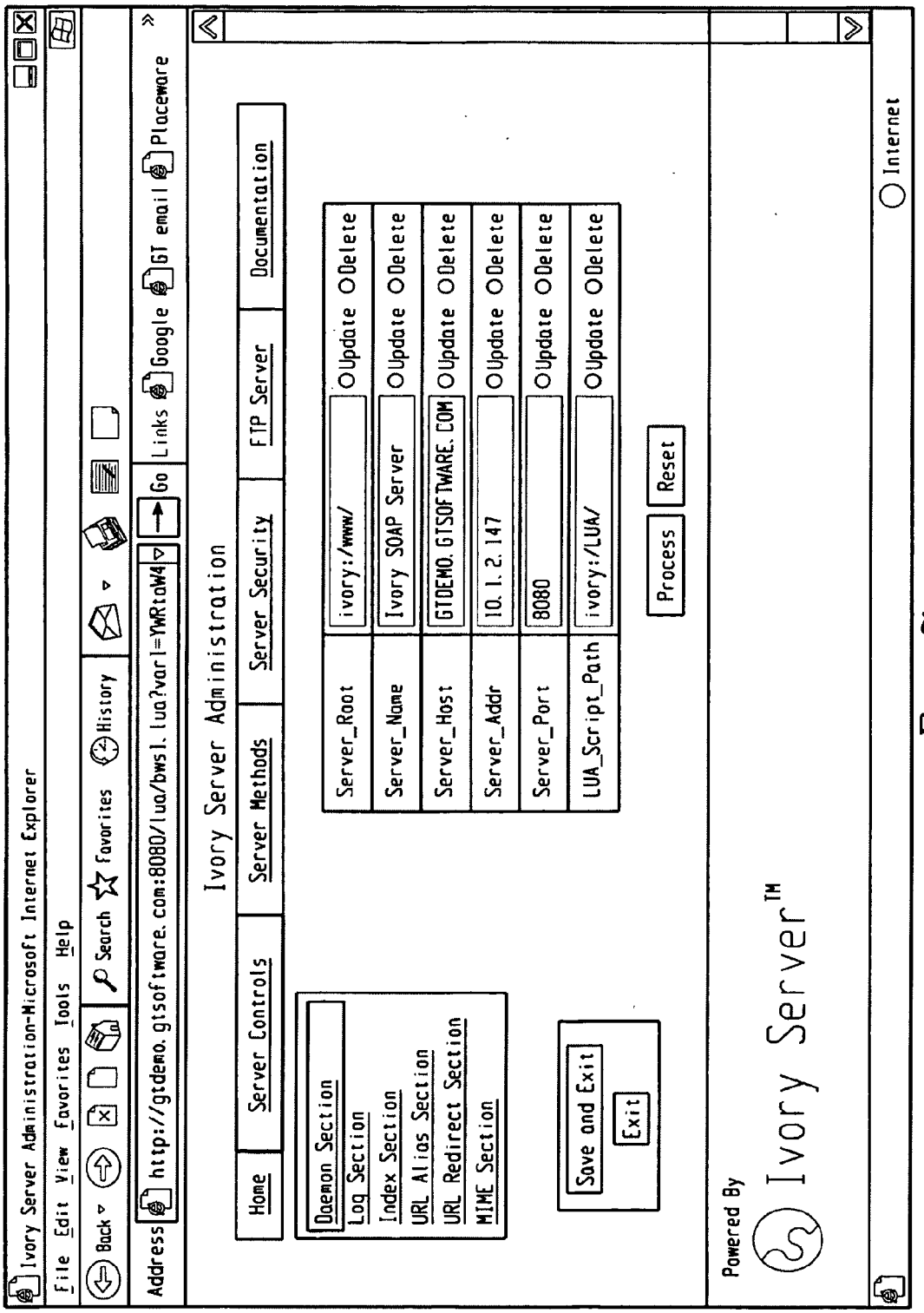

Ivory Server Administration-Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://gtdemo.gtsoftware.com:8080/lua/bwsl.lua?var1=YWRtaW4
Go Links Google GT email Placeware
Ivory Server Administration
Home
Server Controls
Server Methods
Server Security
FTP Server
Documentation
Daemon Section
Log Section
Index Section
URL Alias Section
URL Redirect Section
MIME Section
Server_Root
ivory:/www/
Server_Name
Ivory SOAP Server
Server_Host
GTDEMO.GTSOFTWARE.COM
Server_Addr
10.1.2.147
Server_Port
8080
LUA_Script_Path
ivory:/LUA/
Update Delete
Process
Reset
Save and Exit
Exit
Powered By
Ivory Server™
Internet

Fig. 7

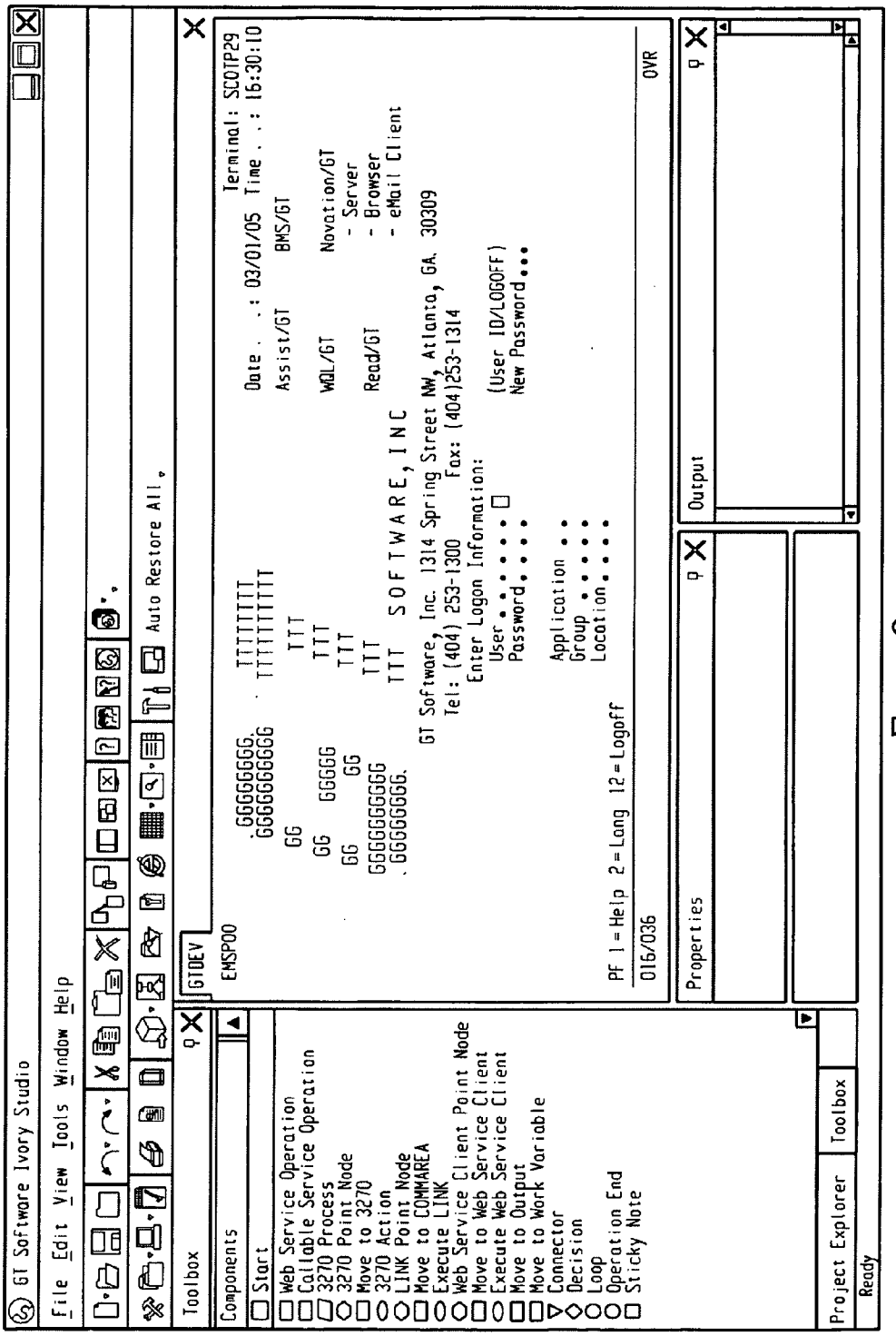

GT Software Ivory Studio
File Edit View Tools Window Help
Toolbox
Components
Start
Web Service Operation
Callable Service Operation
3270 Process
3270 Point Node
Move to 3270
3270 Action
LINK Point Node
Move to COMMAREA
Execute LINK
Web Service Client Point Node
Move to Web Service Client
Execute Web Service Client
Move to Output
Move to Work Variable
Connector
Decision
Loop
Operation End
Sticky Note
Project Explorer
Toolbox
Ready
GTDEV
EMSP00
Terminal: SCOTP29
Date . .: 03/01/05 Time . .: 16:30:10
Assist/GT BMS/GT
WQL/GT Novation/GT
- Server
Read/GT - Browser
- eMail Client
GT Software, Inc. 1314 Spring Street NW, Atlanta, GA. 30309
Tel: (404) 253-1300 Fax: (404)253-1314
Enter Logon Information:
(User ID/LOGOFF)
New Password...
PF 1=Help 2=Lang 12=Logoff
016/036
OVR
Properties
Output
Auto Restore All

Fig. 8

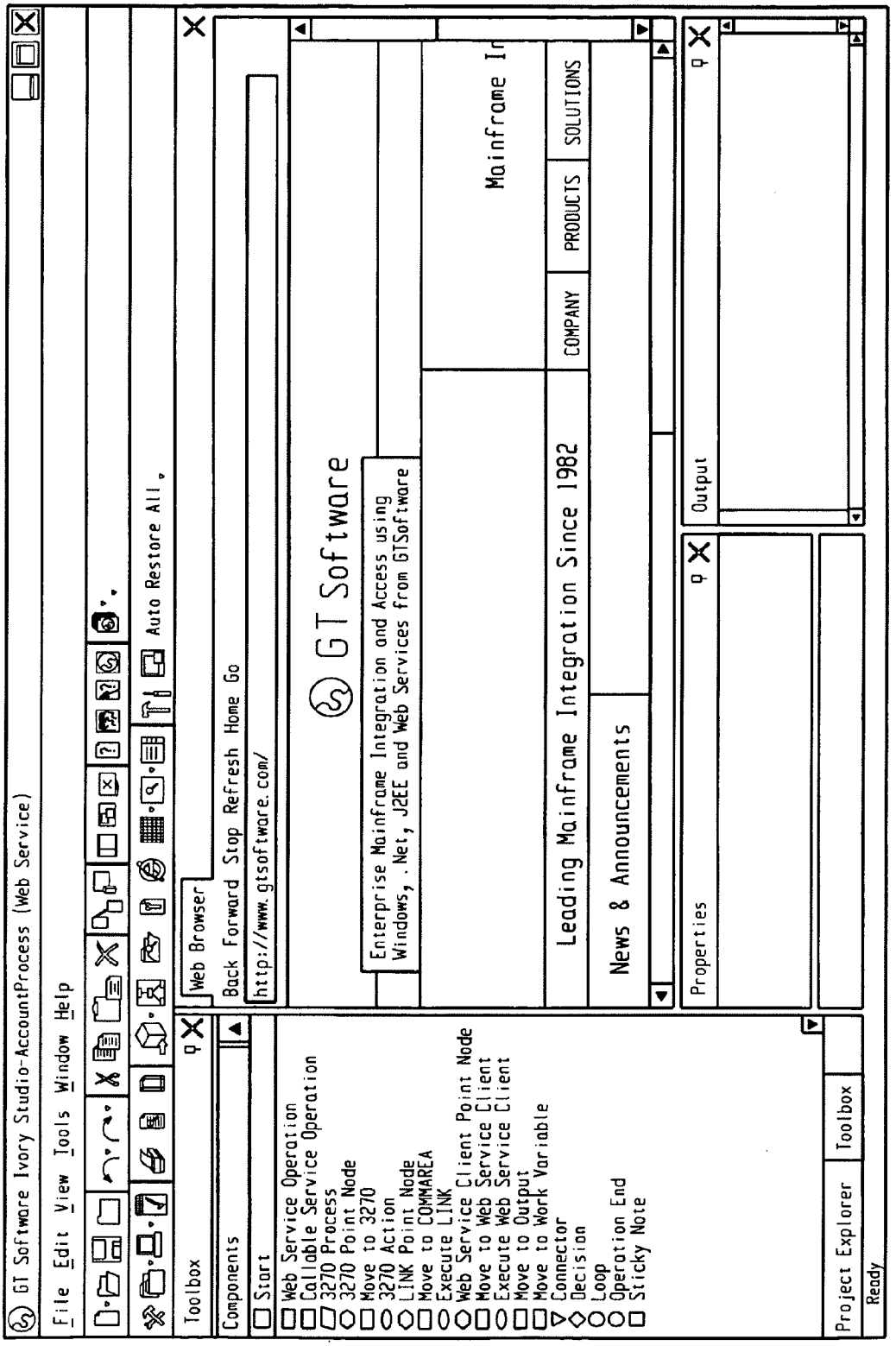

GT Software Ivory Studio-AccountProcess (Web Service)
File Edit View Tools Window Help
Auto Restore All
Toolbox
Components
Start
Web Service Operation
Callable Service Operation
3270 Process
3270 Point Node
Move to 3270
3270 Action
LINK Point Node
Move to COMMAREA
Execute LINK
Web Service Client Point Node
Move to Web Service Client
Execute Web Service Client
Move to Output
Move to Work Variable
Connector
Decision
Loop
Operation End
Sticky Note
Project Explorer
Toolbox
Ready
Web Browser
Back Forward Stop Refresh Home Go
http://www.gtsoftware.com/
GT Software
Enterprise Mainframe Integration and Access using Windows, .Net, J2EE and Web Services from GTSoftware
Mainframe In
Leading Mainframe Integration Since 1982
COMPANY
PRODUCTS
SOLUTIONS
News & Announcements
Properties
Output

Fig. 9

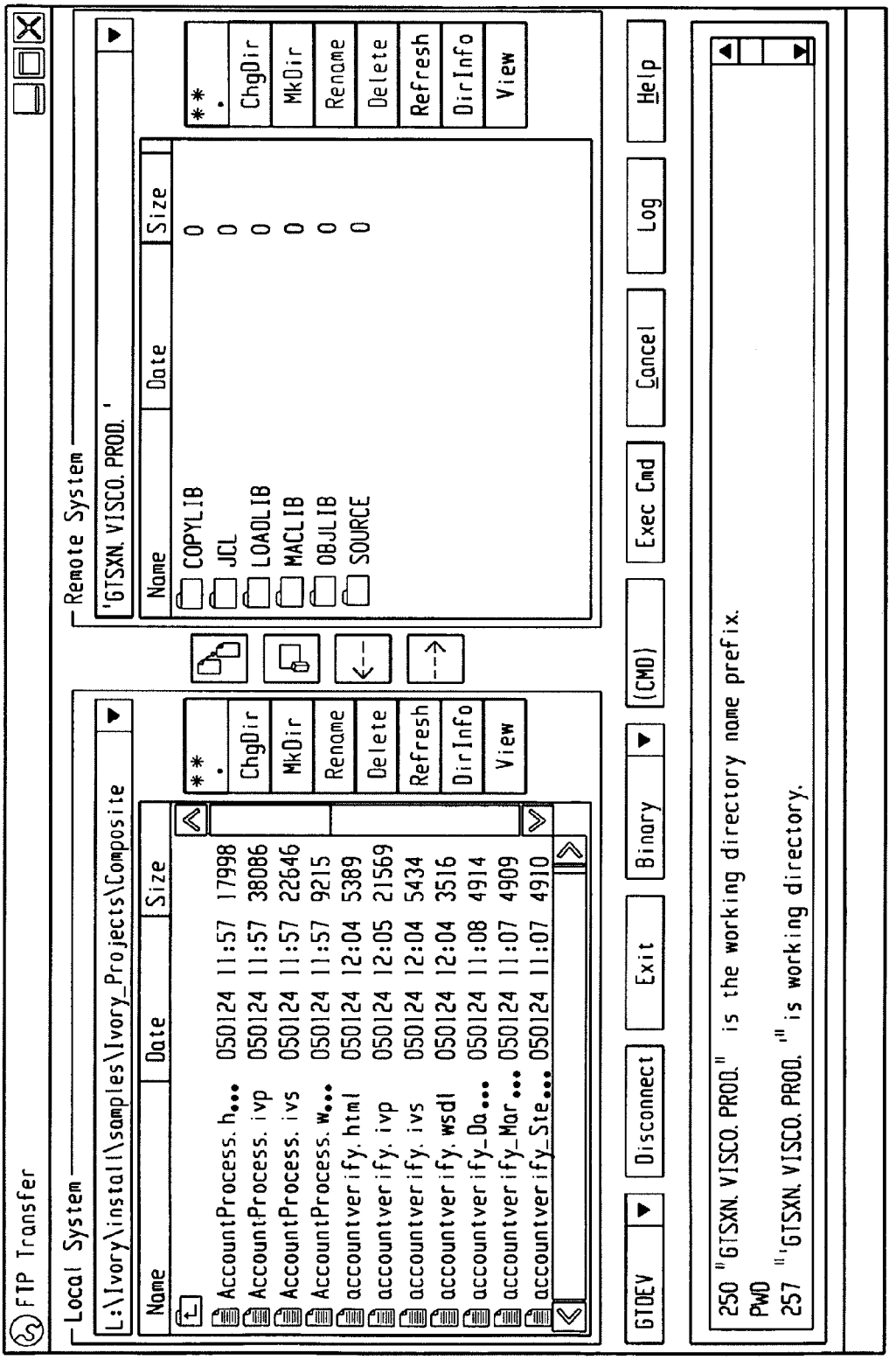

FTP Transfer
Local System
L:\Ivory\install\samples\Ivory_Projects\Composite
Name
Date
Size
AccountProcess.h... 050124 11:57 17998
AccountProcess.ivp 050124 11:57 38086
AccountProcess.ivs 050124 11:57 22646
AccountProcess.w... 050124 11:57 9215
accountverify.html 050124 12:04 5389
accountverify.ivp 050124 12:05 21569
accountverify.ivs 050124 12:04 5434
accountverify.wsdl 050124 12:04 3516
accountverify_Da... 050124 11:08 4914
accountverify_Mar... 050124 11:07 4909
accountverify_Ste... 050124 11:07 4910
*.*
ChgDir
MkDir
Rename
Delete
Refresh
DirInfo
View
Remote System
'GTSXN.VISCO.PROD.'
Name
Date
Size
COPYLIB 0
JCL 0
LOADLIB 0
MACLIB 0
OBJLIB 0
SOURCE 0
*.*
ChgDir
MkDir
Rename
Delete
Refresh
DirInfo
View
GTDEV
Disconnect
Exit
Binary
(CMD)
Exec Cmd
Cancel
Log
Help
250 "GTSXN.VISCO.PROD." is the working directory name prefix.
PWD
257 "'GTSXN.VISCO.PROD.'" is working directory.

Fig. 10

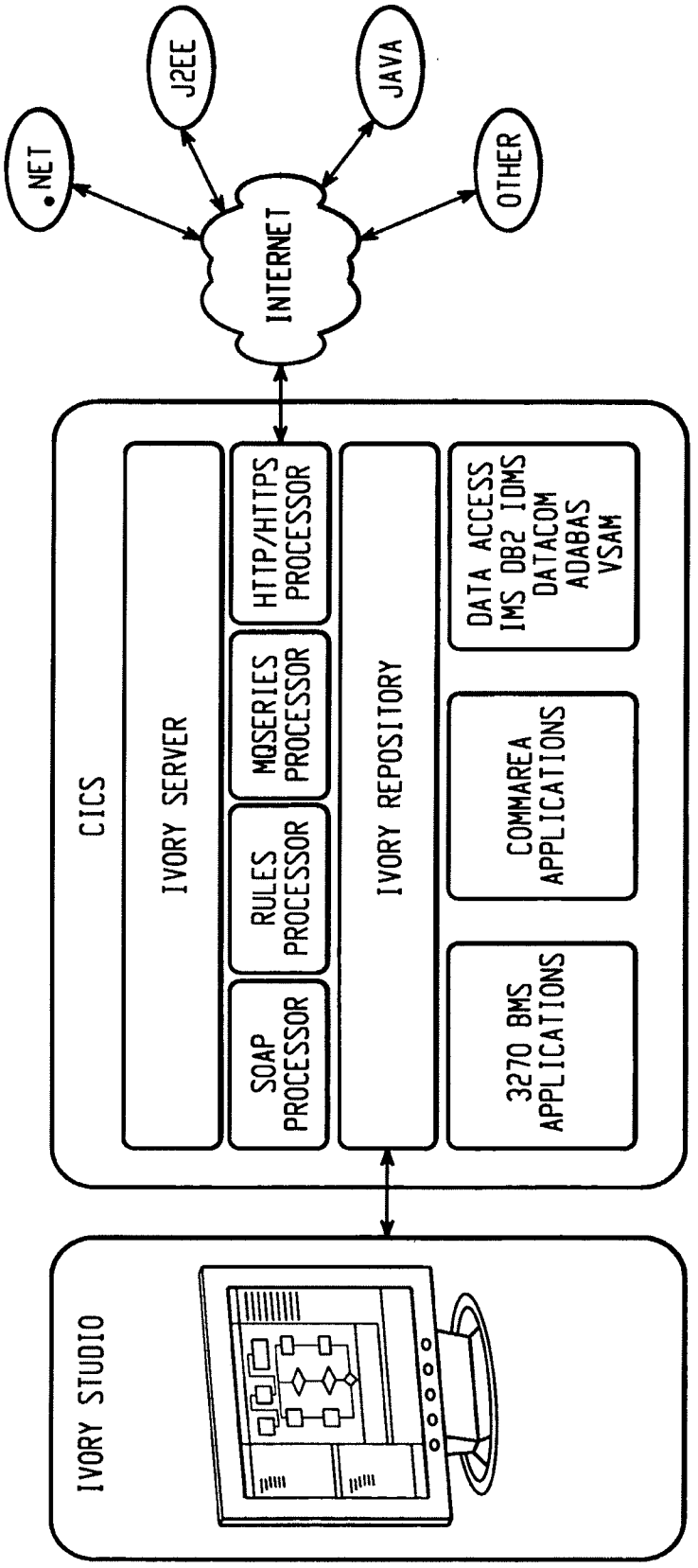

.NET
J2EE
JAVA
OTHER
INTERNET
CICS
IVORY SERVER
SOAP PROCESSOR
RULES PROCESSOR
MQSERIES PROCESSOR
HTTP/HTTPS PROCESSOR
IVORY REPOSITORY
3270 BMS APPLICATIONS
COMMAREA APPLICATIONS
DATA ACCESS IMS DB2 IDMS DATACOM ADABAS VSAM
IVORY STUDIO

Fig. 11

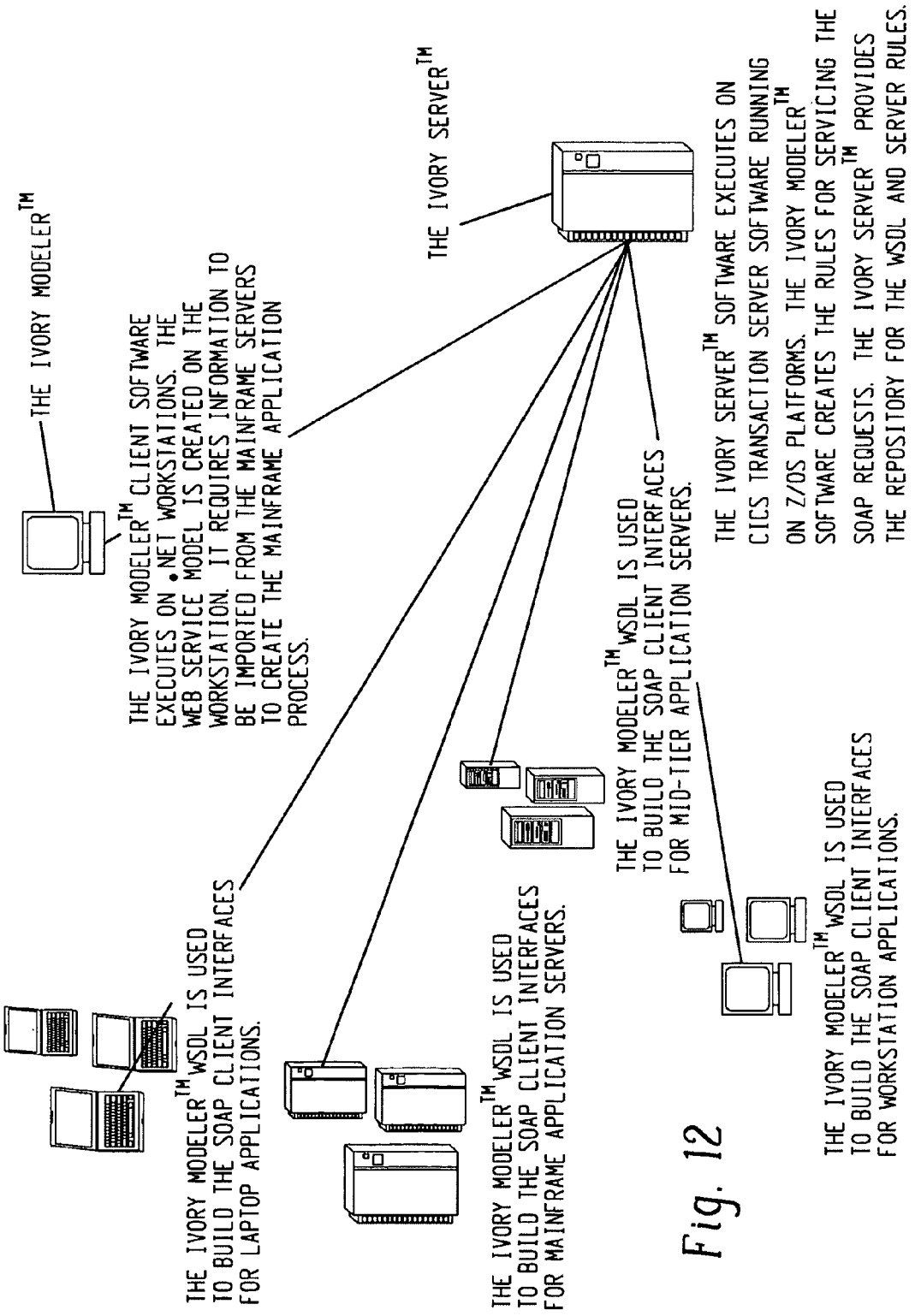
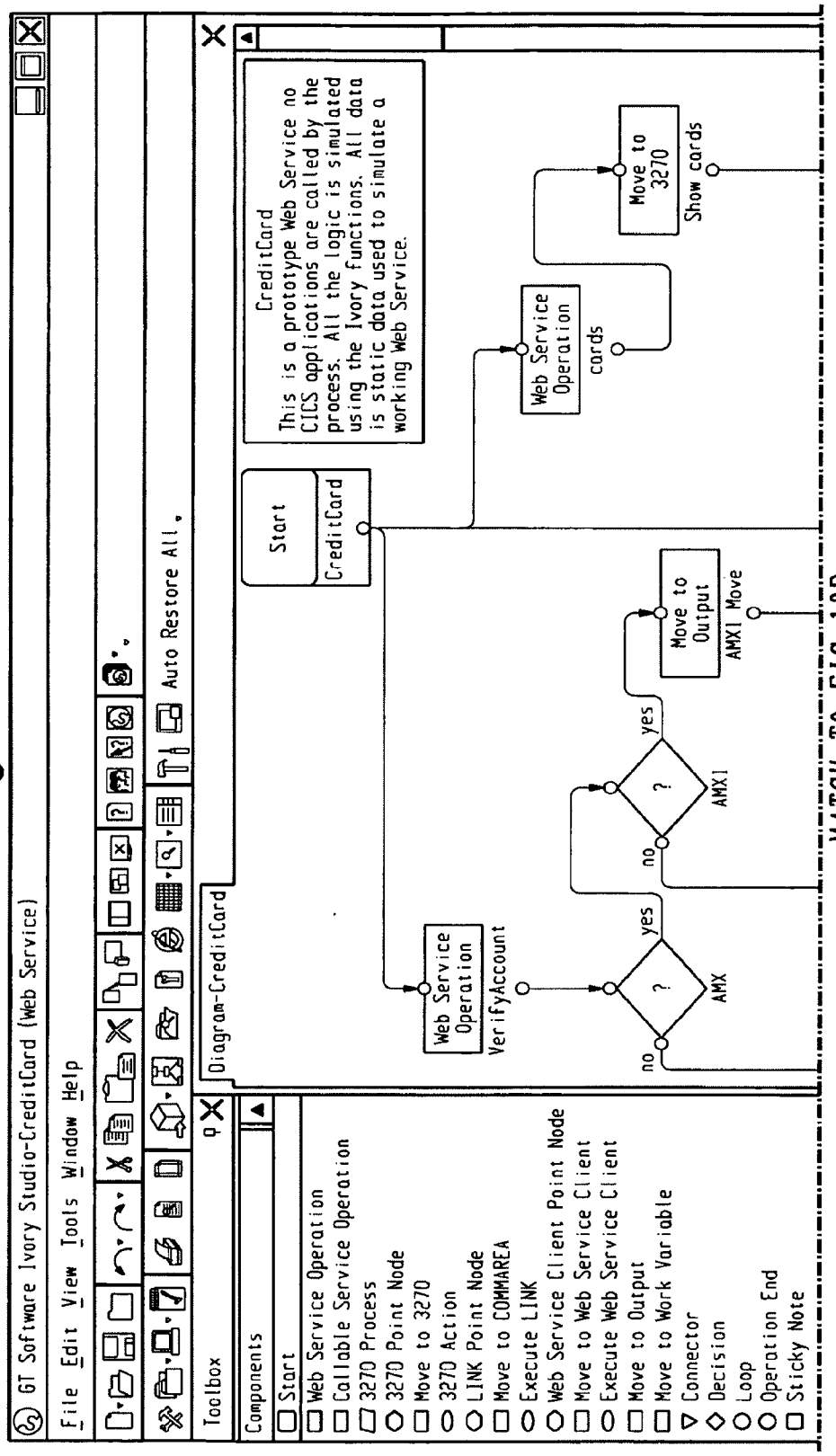

THE IVORY MODELER™
THE IVORY MODELER™ CLIENT SOFTWARE EXECUTES ON .NET WORKSTATIONS. THE WEB SERVICE MODEL IS CREATED ON THE WORKSTATION. IT REQUIRES INFORMATION TO BE IMPORTED FROM THE MAINFRAME SERVERS TO CREATE THE MAINFRAME APPLICATION PROCESS.
THE IVORY SERVER™
THE IVORY MODELER™ WSDL IS USED TO BUILD THE SOAP CLIENT INTERFACES FOR LAPTOP APPLICATIONS.
THE IVORY MODELER™ WSDL IS USED TO BUILD THE SOAP CLIENT INTERFACES FOR MAINFRAME APPLICATION SERVERS.
THE IVORY MODELER™ WSDL IS USED TO BUILD THE SOAP CLIENT INTERFACES FOR MID-TIER APPLICATION SERVERS.
THE IVORY SERVER™ SOFTWARE EXECUTES ON CICS TRANSACTION SERVER SOFTWARE RUNNING ON Z/OS PLATFORMS. THE IVORY MODELER™ SOFTWARE CREATES THE RULES FOR SERVICING THE SOAP REQUESTS. THE IVORY SERVER™ PROVIDES THE REPOSITORY FOR THE WSDL AND SERVER RULES.
THE IVORY MODELER™ WSDL IS USED TO BUILD THE SOAP CLIENT INTERFACES FOR WORKSTATION APPLICATIONS.

Fig. 12

Fig. 13A


GT Software Ivory Studio-CreditCard (Web Service)
File Edit View Tools Window Help
Auto Restore All
Toolbox
Components
Start
Web Service Operation
Callable Service Operation
3270 Process
3270 Point Node
Move to 3270
3270 Action
LINK Point Node
Move to COMMAREA
Execute LINK
Web Service Client Point Node
Move to Web Service Client
Execute Web Service Client
Move to Output
Move to Work Variable
Connector
Decision
Loop
Operation End
Sticky Note
Diagram-CreditCard
Start
CreditCard
CreditCard
This is a prototype Web Service no CICS applications are called by the process. All the logic is simulated using the Ivory functions. All data is static data used to simulate a working Web Service.
Web Service Operation
VerifyAccount
?
AMX
yes
no
?
AMX1
yes
no
Move to Output
AMX1 Move
Web Service Operation
cards
Move to 3270
Show cards
MATCH TO FIC.13B

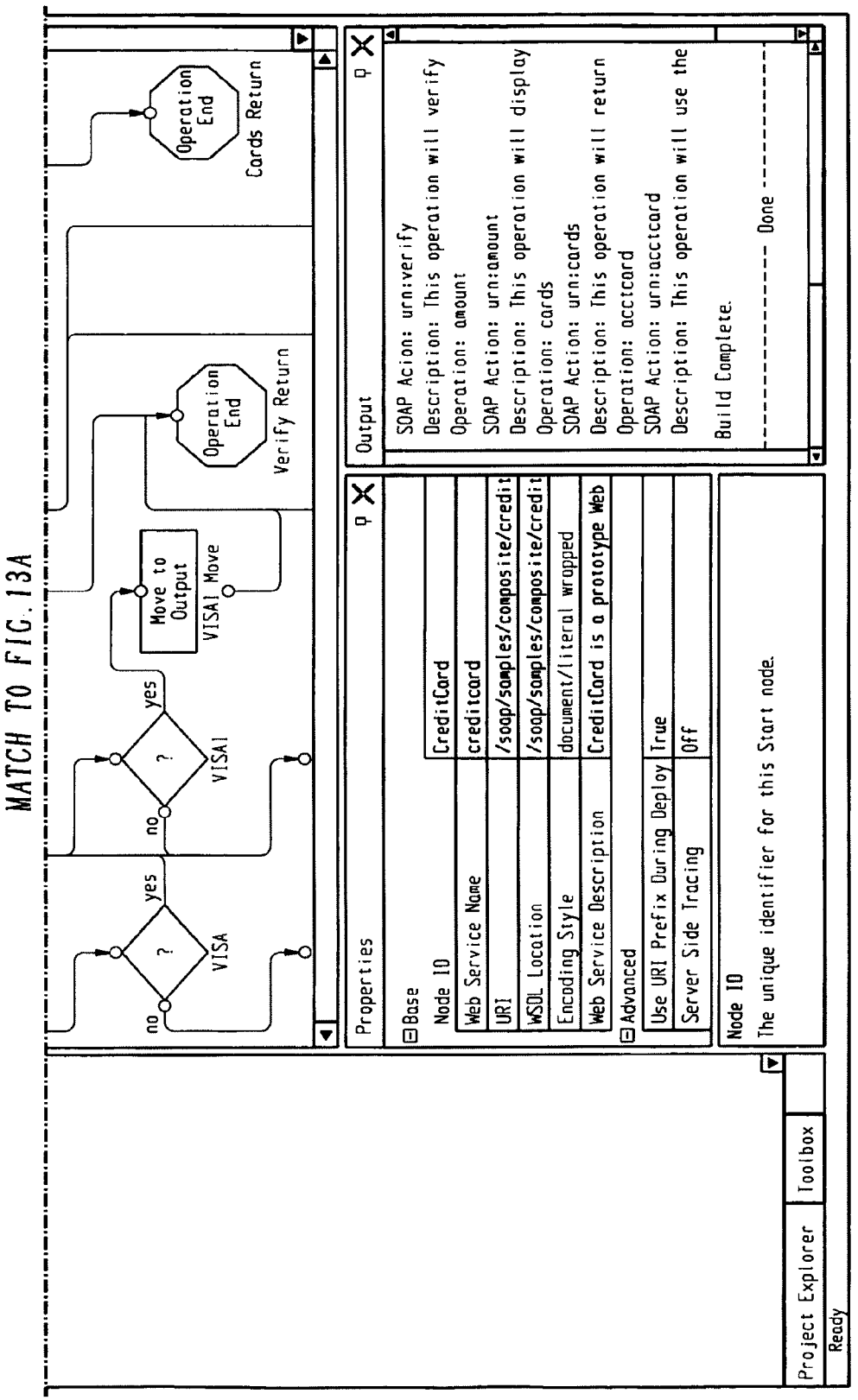

MATCH TO FIG. 13A
Operation End
Cards Return
Operation End
Verify Return
Move to Output
VISA1 Move
yes
no
?
VISA1
yes
no
?
VISA
Properties
Base
Node ID
CreditCard
Web Service Name
creditcard
URI
/soap/samples/composite/credit
WSDL Location
/soap/samples/composite/credit
Encoding Style
document/literal wrapped
Web Service Description
CreditCard is a prototype Web
Advanced
Use URI Prefix During Deploy
True
Server Side Tracing
Off
Output
SOAP Acion: urn:verify
Description: This operation will verify
Operation: amount
SOAP Action: urn:amount
Description: This operation will display
Operation: cards
SOAP Action: urn:cards
Description: This operation will return
Operation: acctcard
SOAP Action: urn:acctcard
Description: This operation will use the
Build Complete.
---------- Done ----------
Node ID
The unique identifier for this Start node.
Project Explorer
Toolbox
Ready

Fig. 13B

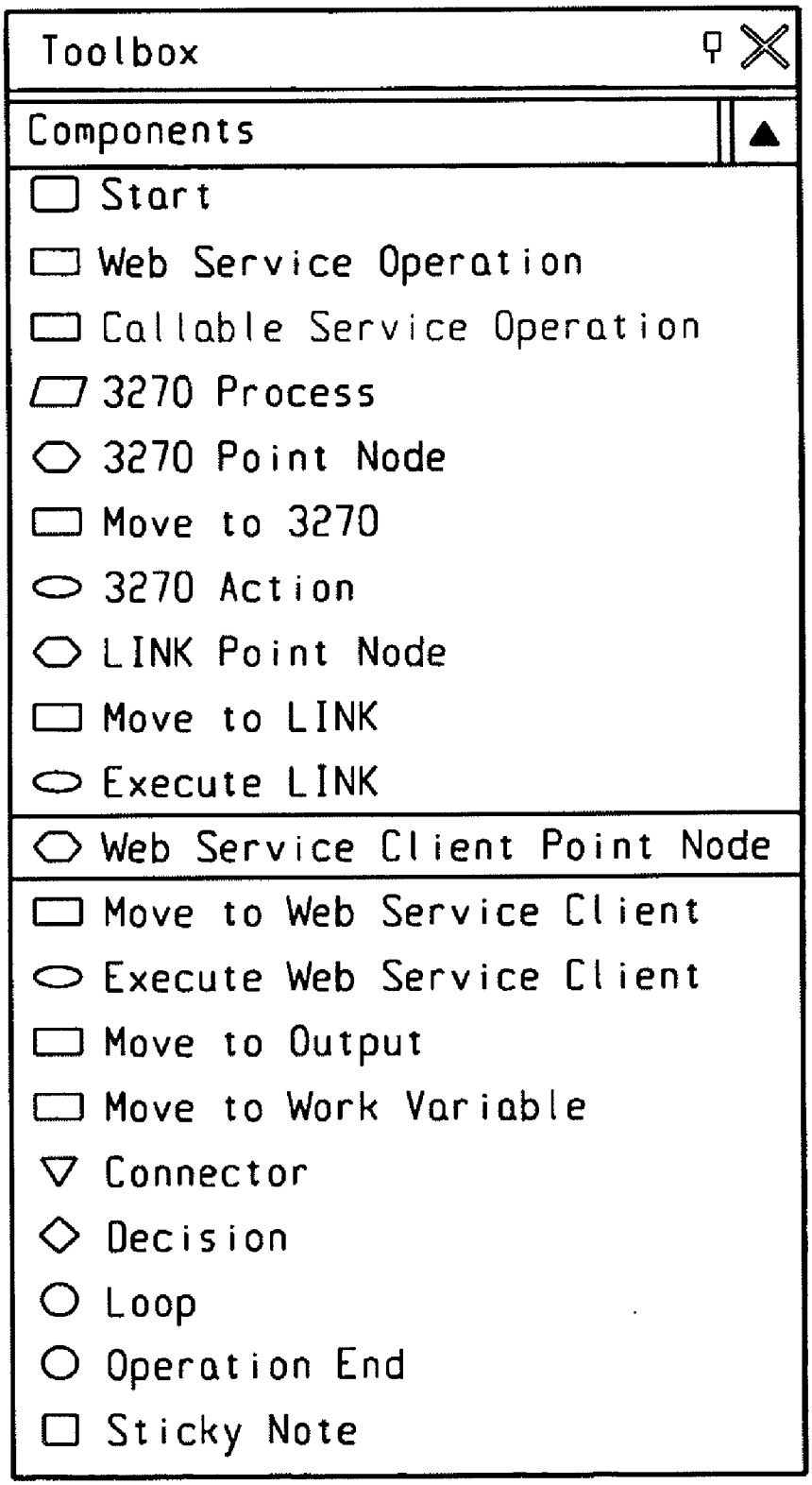

Toolbox
Components
Start
Web Service Operation
Callable Service Operation
3270 Process
3270 Point Node
Move to 3270
3270 Action
LINK Point Node
Move to LINK
Execute LINK
Web Service Client Point Node
Move to Web Service Client
Execute Web Service Client
Move to Output
Move to Work Variable
Connector
Decision
Loop
Operation End
Sticky Note

Fig. 14

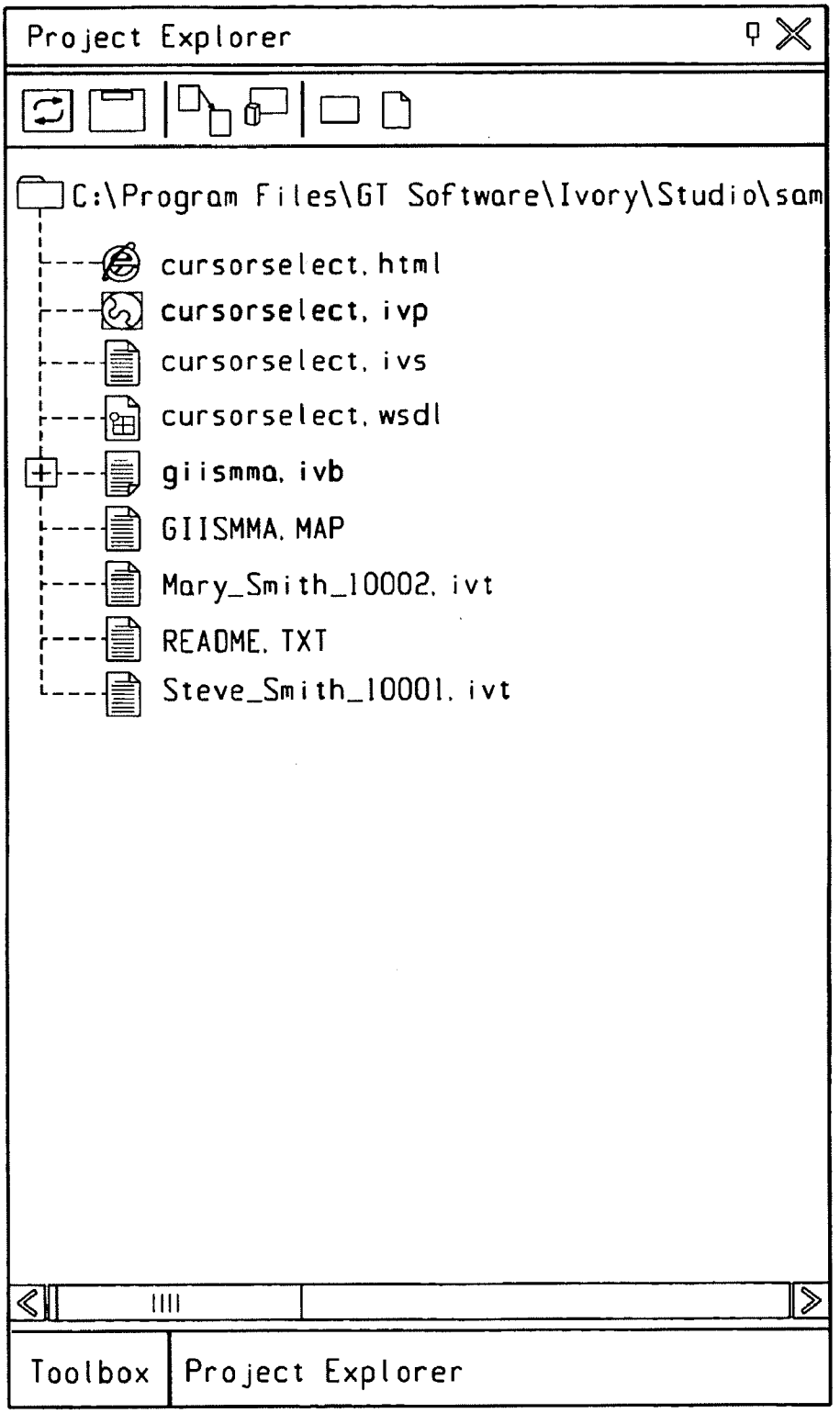

Project Explorer
C:\Program Files\GT Software\Ivory\Studio\sam
cursorselect.html
cursorselect.ivp
cursorselect.ivs
cursorselect.wsdl
giismma.ivb
GIISMMA.MAP
Mary_Smith_10002.ivt
README.TXT
Steve_Smith_10001.ivt
Toolbox
Project Explorer

Fig. 15

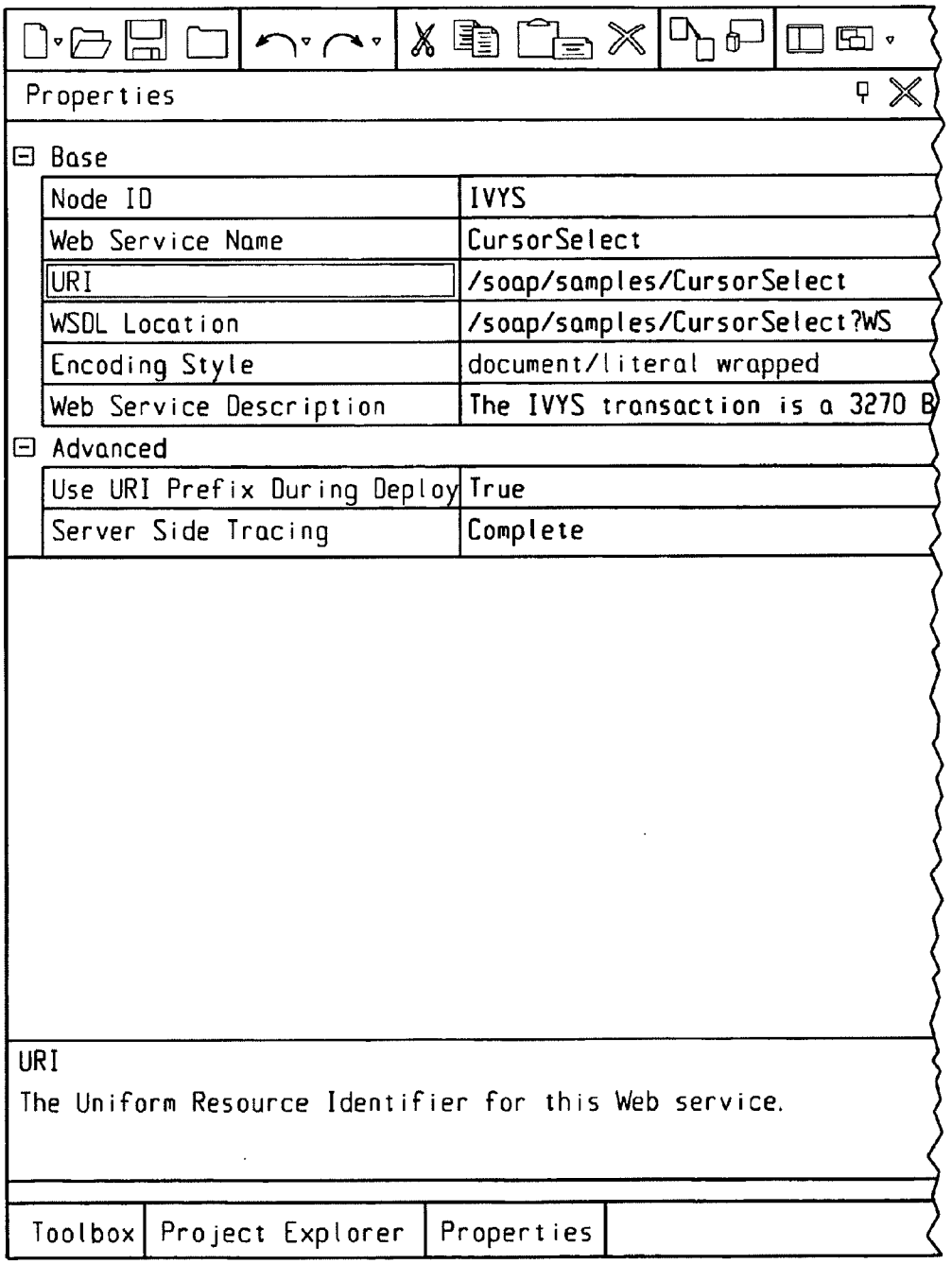

Properties
Base
Node ID
IVYS
Web Service Name
CursorSelect
URI
/soap/samples/CursorSelect
WSDL Location
/soap/samples/CursorSelect?WS
Encoding Style
document/literal wrapped
Web Service Description
The IVYS transaction is a 3270 B
Advanced
Use URI Prefix During Deploy
True
Server Side Tracing
Complete
URI
The Uniform Resource Identifier for this Web service.
Toolbox
Project Explorer
Properties

Fig. 16

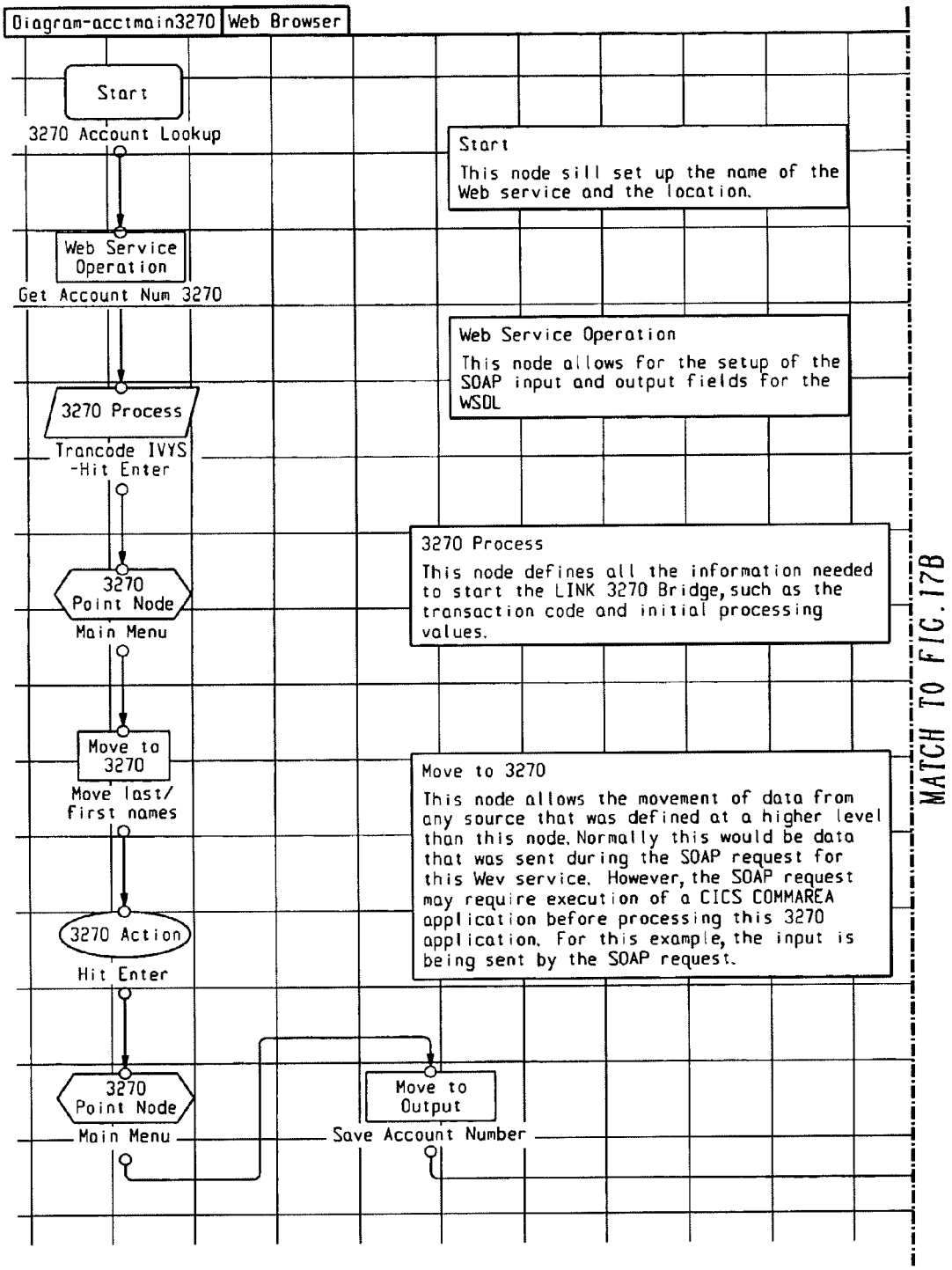


Diagram-acctmain3270
Web Browser
Start
3270 Account Lookup
Web Service Operation
Get Account Num 3270
3270 Process
Trancode IVYS -Hit Enter
3270 Point Node
Main Menu
Move to 3270
Move last/ first names
3270 Action
Hit Enter
3270 Point Node
Main Menu
Move to Output
Save Account Number
Start
This node sill set up the name of the Web service and the location.
Web Service Operation
This node allows for the setup of the SOAP input and output fields for the WSDL
3270 Process
This node defines all the information needed to start the LINK 3270 Bridge, such as the transaction code and initial processing values.
Move to 3270
This node allows the movement of data from any source that was defined at a higher level than this node. Normally this would be data that was sent during the SOAP request for this Wev service. However, the SOAP request may require execution of a CICS COMMAREA application before processing this 3270 application. For this example, the input is being sent by the SOAP request.
MATCH TO FIG. 17B

Fig. 17A

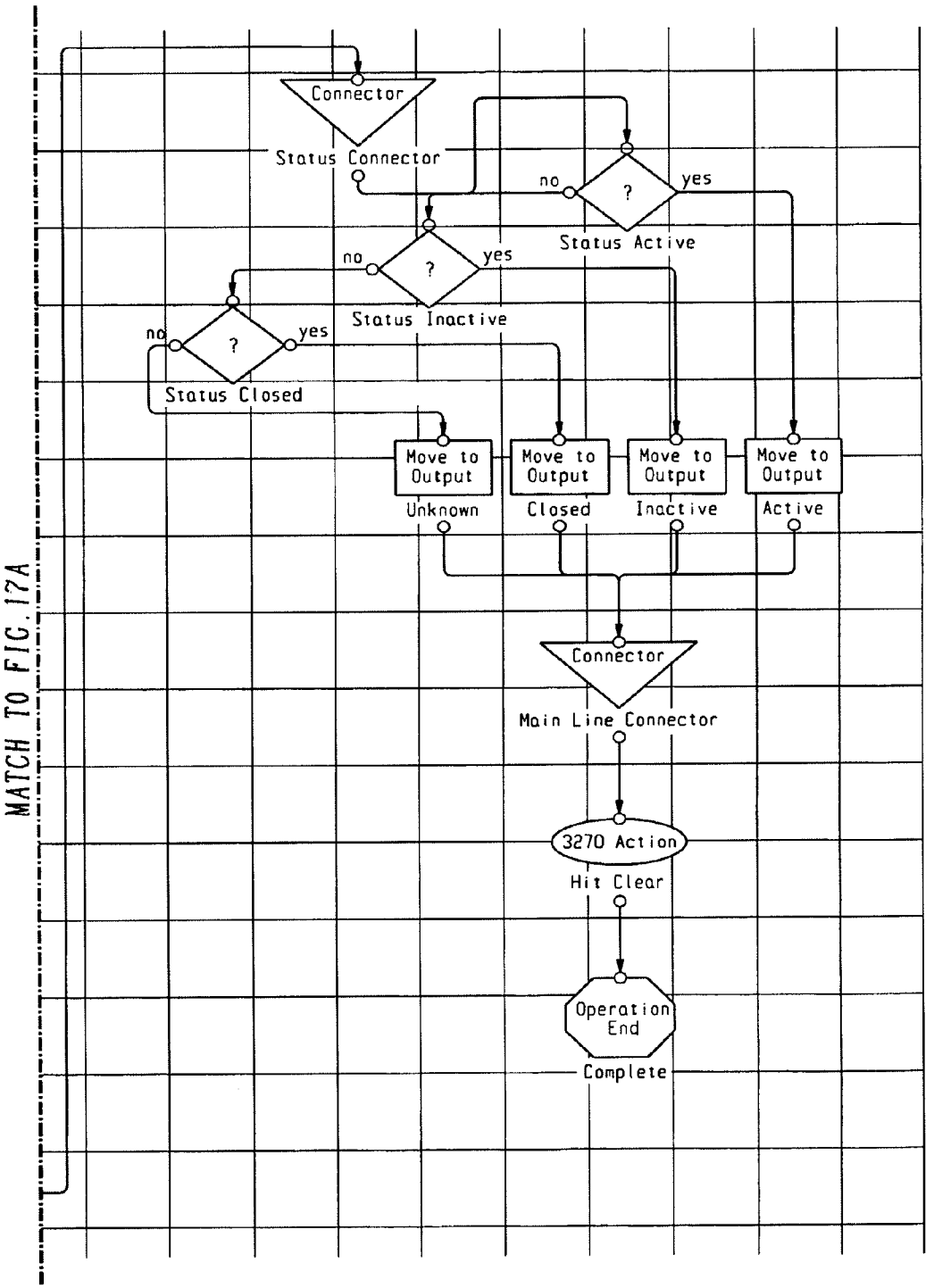

MATCH TO FIG. 17A
Connector
Status Connector
no
?
yes
Status Active
no
?
yes
Status Inactive
no
?
yes
Status Closed
Move to Output
Unknown
Move to Output
Closed
Move to Output
Inactive
Move to Output
Active
Connector
Main Line Connector
3270 Action
Hit Clear
Operation End
Complete

Fig. 17B

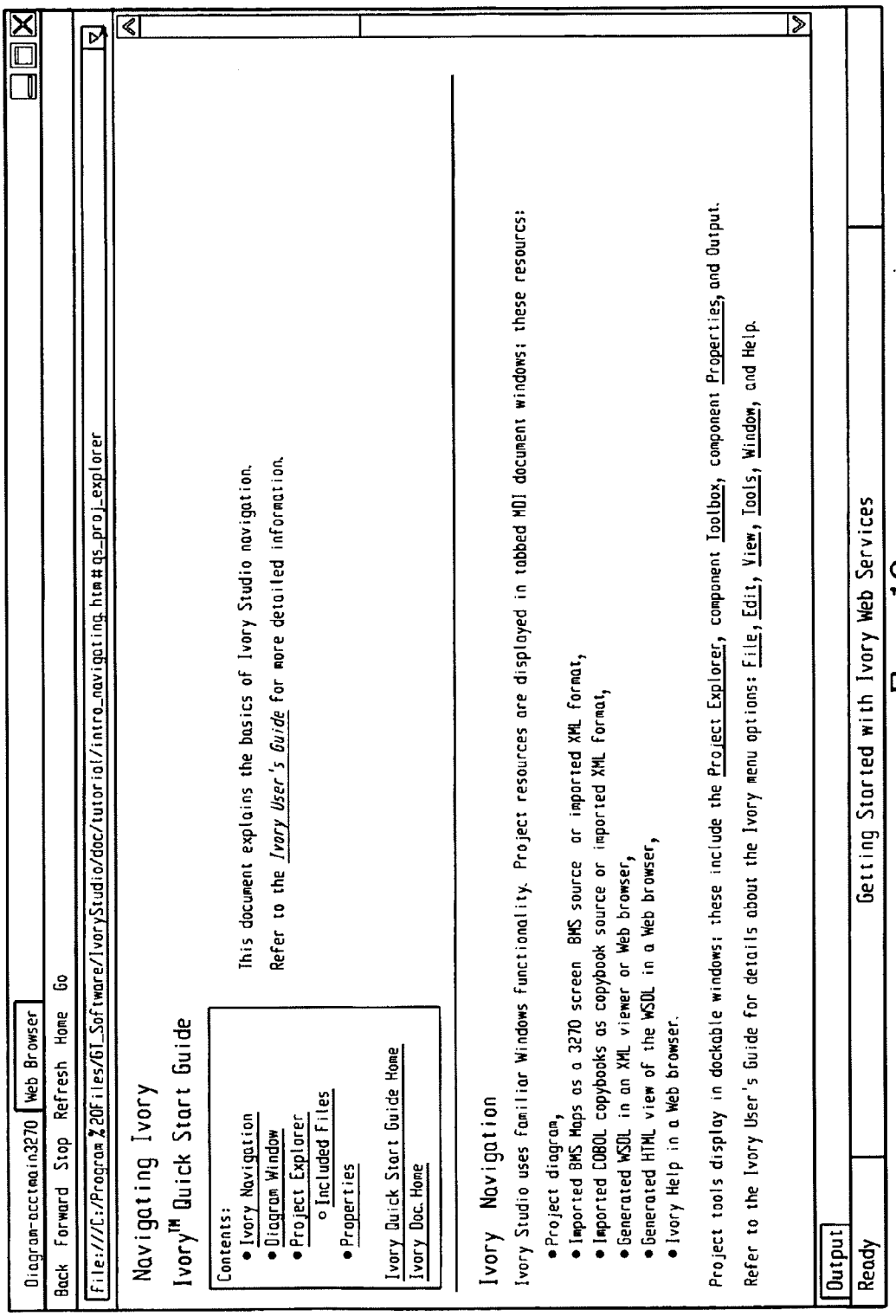


Diagram-acctmain3270
Web Browser
Back Forward Stop Refresh Home Go
File:///C:/Program%20Files/GI_Software/IvoryStudio/doc/tutorial/intro_navigating.htm#qs_proj_explorer
Navigating Ivory
Ivory™ Quick Start Guide
Contents:
Ivory Navigation
Diagram Window
Project Explorer
Included Files
Properties
Ivory Quick Start Guide Home
Ivory Doc. Home
This document explains the basics of Ivory Studio navigation.
Refer to the Ivory User's Guide for more detailed information.
Ivory Navigation
Ivory Studio uses familiar Windows functionality. Project resources are displayed in tabbed MDI document windows: these resources:
Project diagram,
Imported BMS Maps as a 3270 screen BMS source or imported XML format,
Imported COBOL copybooks as copybook source or imported XML format,
Generated WSDL in an XML viewer or Web browser,
Generated HTML view of the WSDL in a Web browser,
Ivory Help in a Web browser.
Project tools display in dockable windows: these include the Project Explorer, component Toolbox, component Properties, and Output.
Refer to the Ivory User's Guide for details about the Ivory menu options: File, Edit, View, Tools, Window, and Help.
Output
Ready
Getting Started with Ivory Web Services

Fig. 18

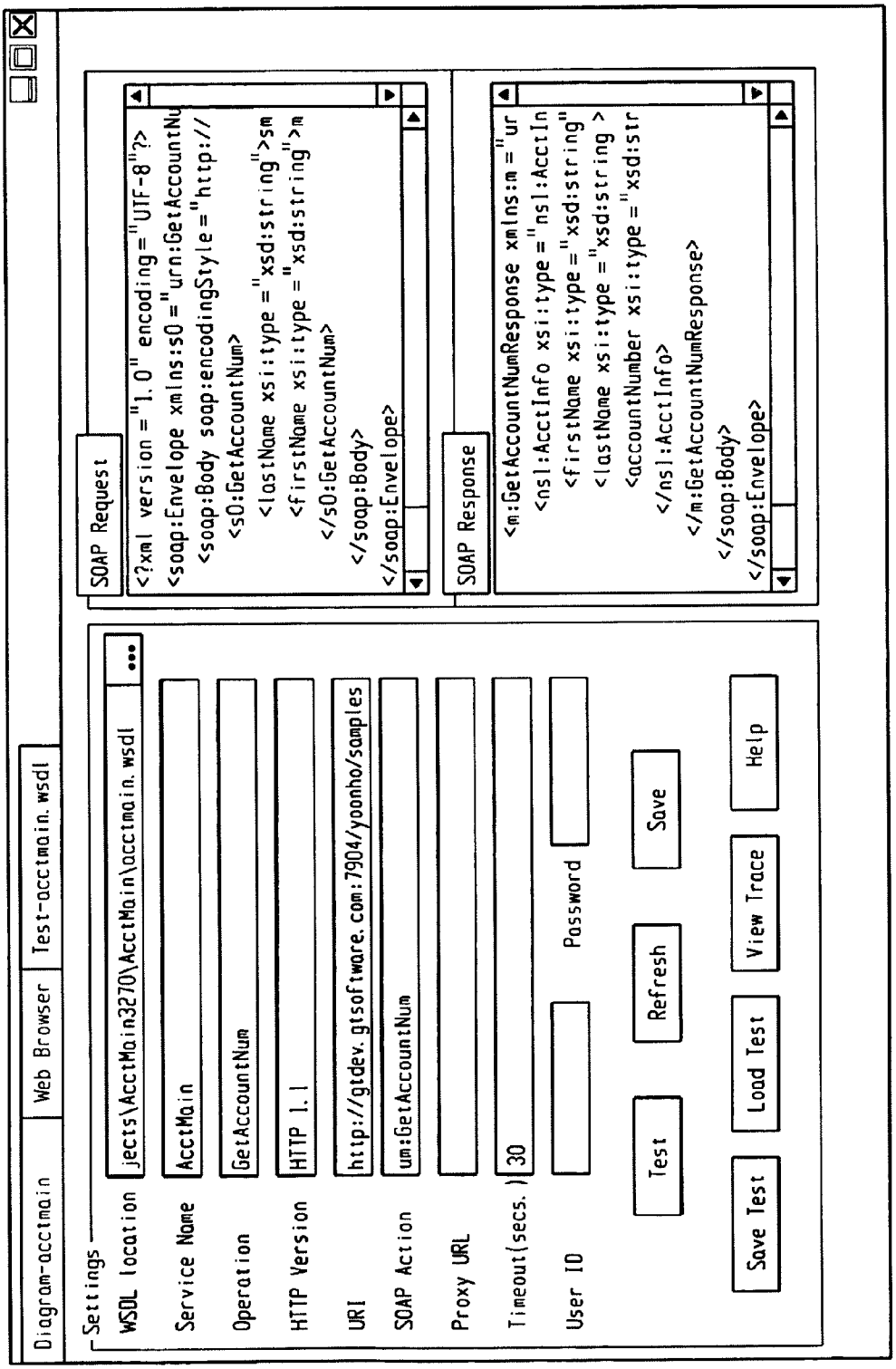


Diagram-acctmain
Web Browser
Test-acctmain.wsdl
Settings
WSDL location
jects\AcctMain3270\AcctMain\acctmain.wsdl
Service Name
AcctMain
Operation
GetAccountNum
HTTP Version
HTTP 1.1
URI
http://gtdev.gtsoftware.com:7904/yoonho/samples
SOAP Action
urn:GetAccountNum
Proxy URL
Timeout(secs.)
30
User ID
Password
Test
Refresh
Save
Save Test
Load Test
View Trace
Help
SOAP Request
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:s0="urn:GetAccountNu
<soap:Body soap:encodingStyle="http://
<s0:GetAccountNum>
<lastName xsi:type="xsd:string">sm
<firstName xsi:type="xsd:string">m
</s0:GetAccountNum>
</soap:Body>
</soap:Envelope>
SOAP Response
<m:GetAccountNumResponse xmlns:m="ur
<nsl:AcctInfo xsi:type="nsl:AcctIn
<firstName xsi:type="xsd:string"
<lastName xsi:type="xsd:string">
<accountNumber xsi:type="xsd:str
</nsl:AcctInfo>
</m:GetAccountNumResponse>
</soap:Body>
</soap:Envelope>

Fig. 19

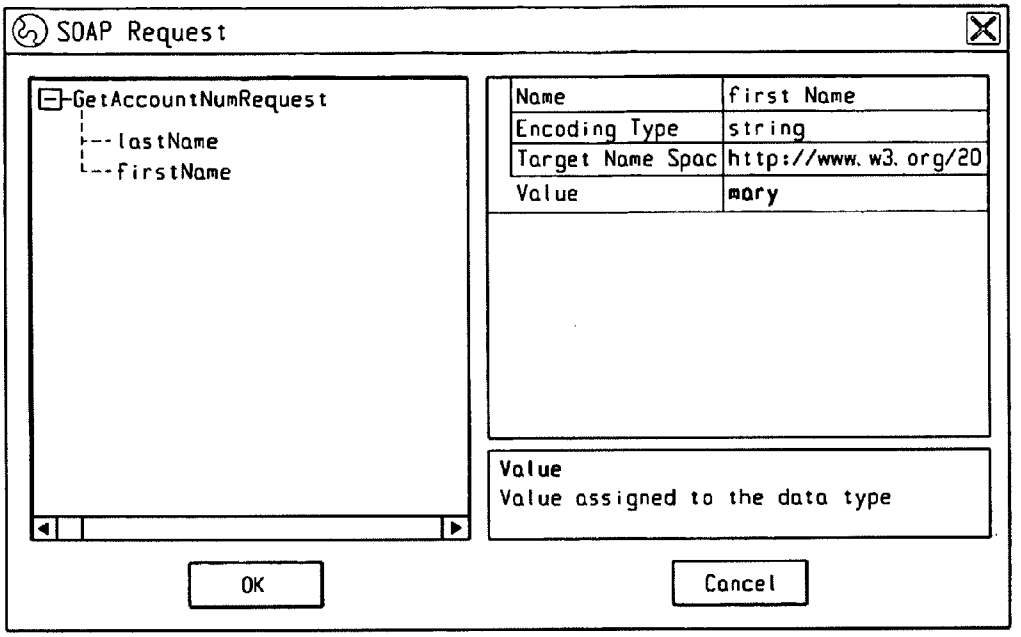

SOAP Request
GetAccountNumRequest
lastName
firstName
Name
first Name
Encoding Type
string
Target Name Spac
http://www.w3.org/20
Value
mary
Value
Value assigned to the data type
OK
Cancel

Fig. 20

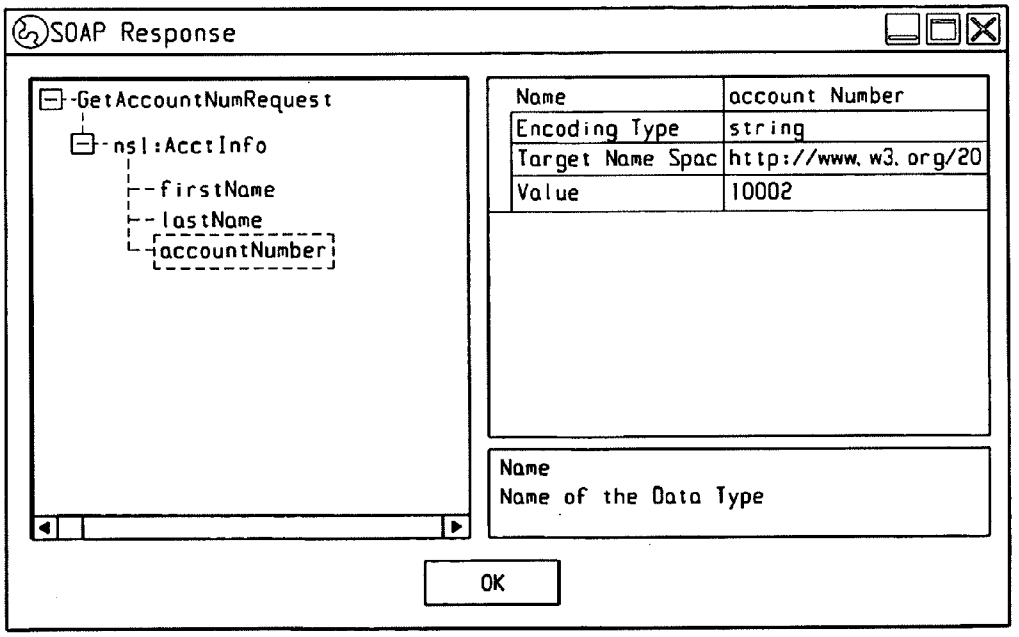

SOAP Response
GetAccountNumRequest
ns1:AcctInfo
firstName
lastName
accountNumber
Name
account Number
Encoding Type
string
Target Name Spac
http://www.w3.org/20
Value
10002
Name
Name of the Data Type
OK

Fig. 21

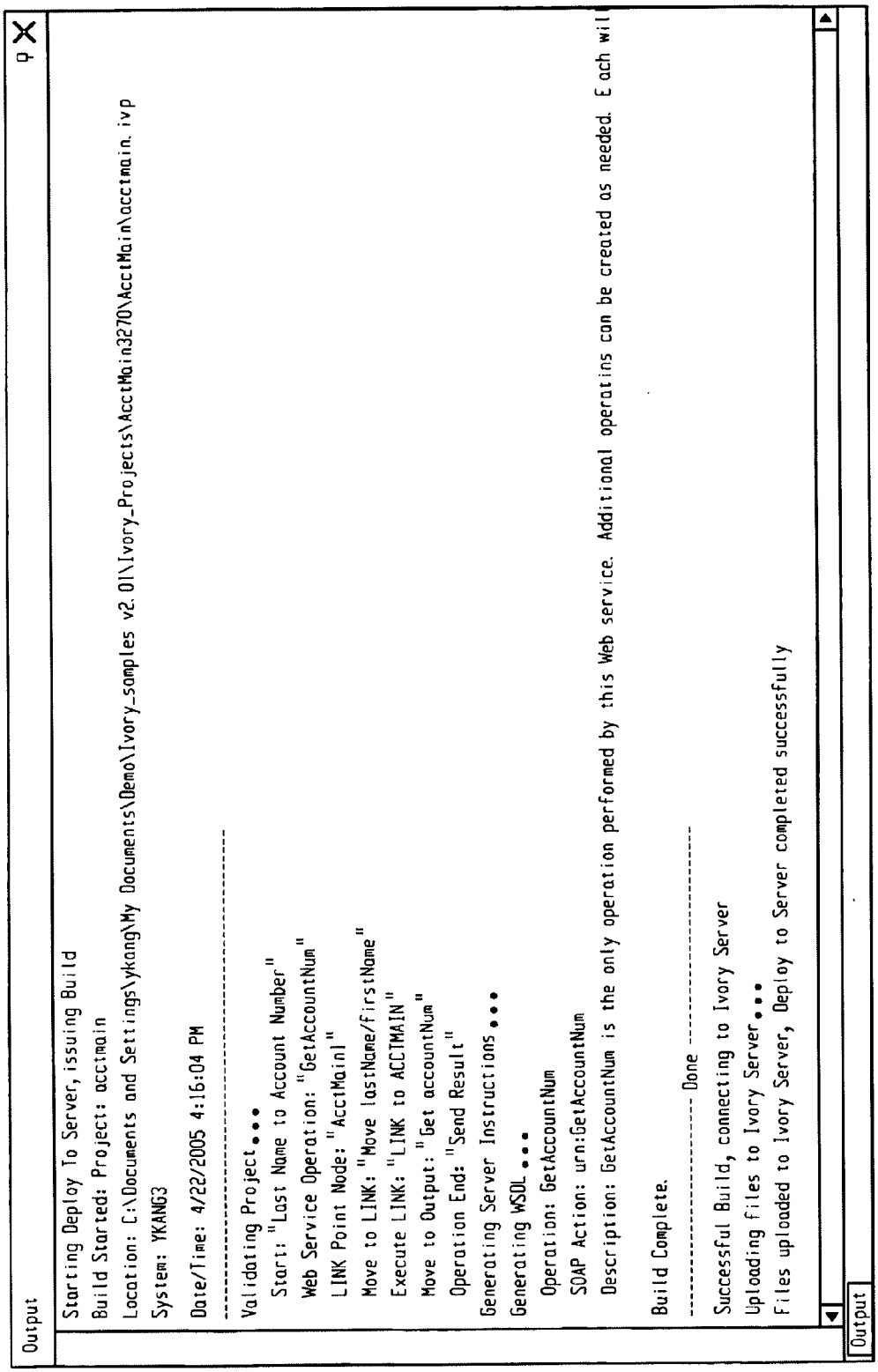

Output
Starting Deploy To Server, issuing Build
Build Started: Project: acctmain
Location: C:\Documents and Settings\ykang\My Documents\Demo\Ivory_samples v2.01\Ivory_Projects\AcctMain3270\AcctMain\acctmain.ivp
System: YKANG3
Date/Time: 4/22/2005 4:16:04 PM
Validating Project...
Start: "Last Name to Account Number"
Web Service Operation: "GetAccountNum"
LINK Point Node: "AcctMain1"
Move to LINK: "Move lastName/firstName"
Execute LINK: "LINK to ACCTMAIN"
Move to Output: "Get accountNum"
Operation End: "Send Result"
Generating Server Instructions...
Generating WSDL ...
Operation: GetAccountNum
SOAP Action: urn:GetAccountNum
Description: GetAccountNum is the only operation performed by this Web service. Additional operations can be created as needed. Each wil
Build Complete.
Done
Successful Build, connecting to Ivory Server
Uploading files to Ivory Server...
Files uploaded to Ivory Server, Deploy to Server completed successfully
Output

Fig. 22

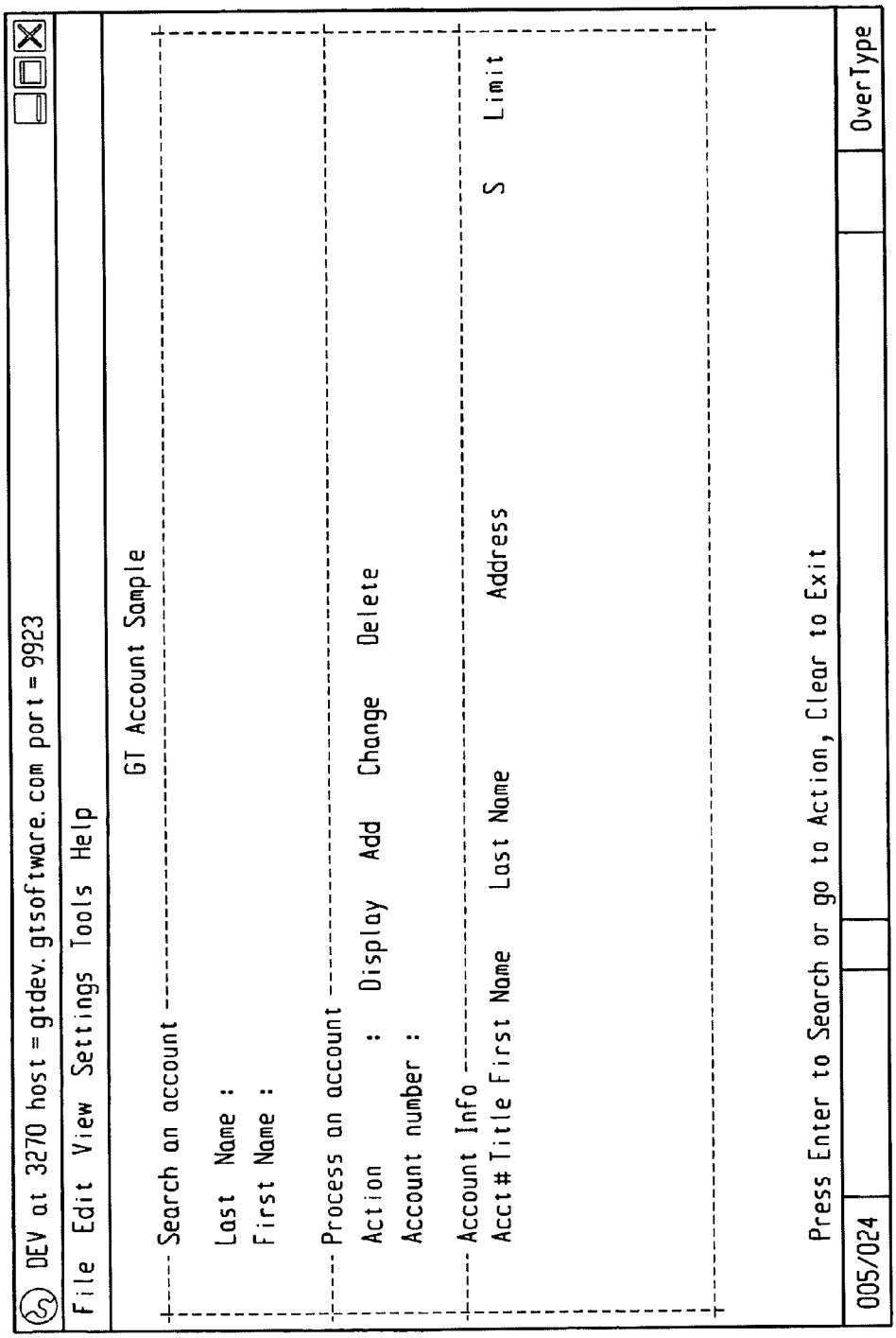

DEV at 3270 host = gtdev.gtsoftware.com port = 9923
File Edit View Settings Tools Help
GT Account Sample
Search an account
Last Name :
First Name :
Process an account
Action : Display Add Change Delete
Account number :
Account Info
Acct# Title First Name Last Name Address S Limit
Press Enter to Search or go to Action, Clear to Exit
005/024
OverType

Fig. 23

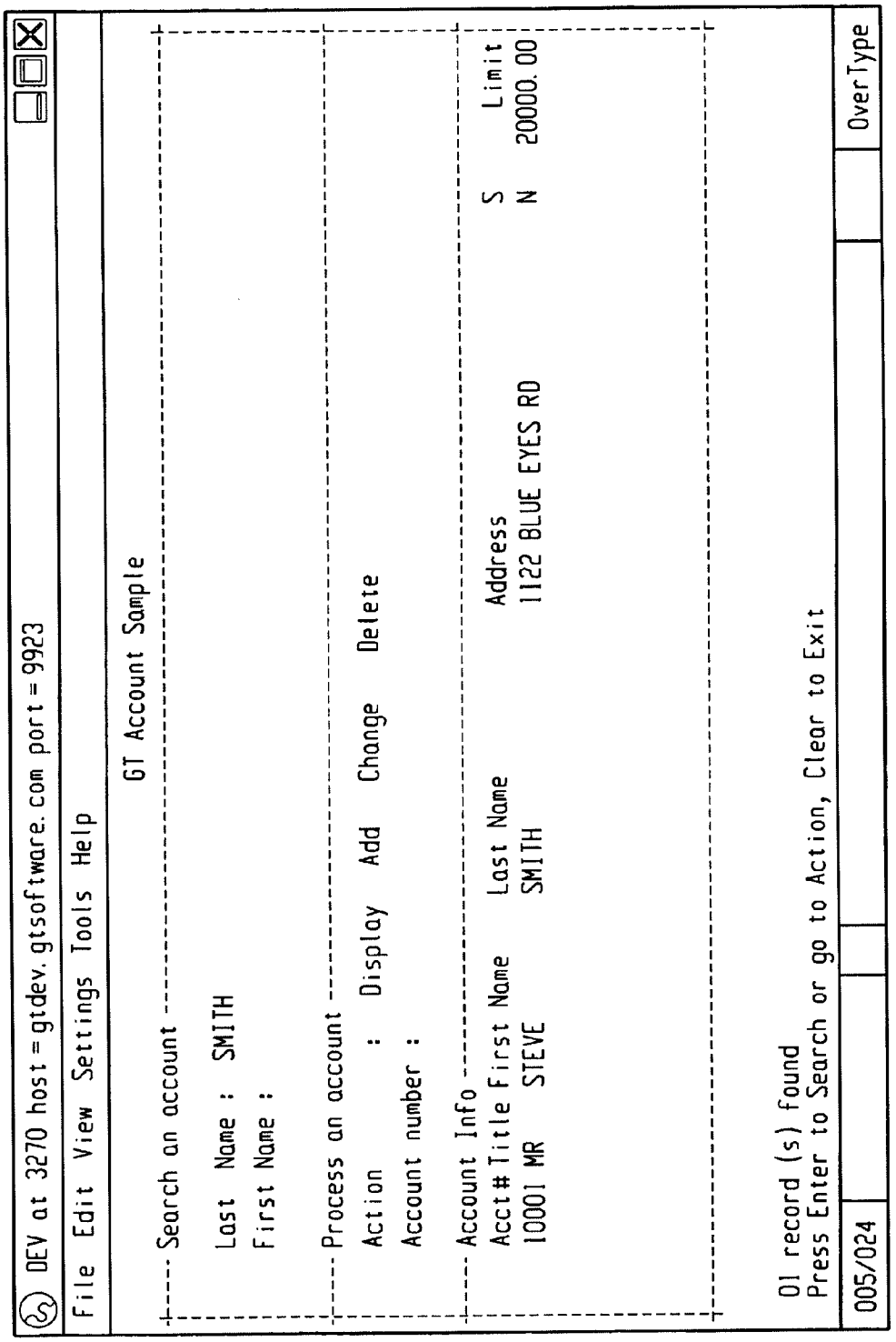

DEV at 3270 host = gtdev.gtsoftware.com port = 9923
File Edit View Settings Tools Help
GT Account Sample
Search an account
Last Name : SMITH
First Name :
Process an account
Action : Display Add Change Delete
Account number :
Account Info
Acct# Title First Name Last Name Address S Limit
10001 MR STEVE SMITH 1122 BLUE EYES RD N 20000.00
01 record (s) found
Press Enter to Search or go to Action, Clear to Exit
005/024
OverType

Fig. 24

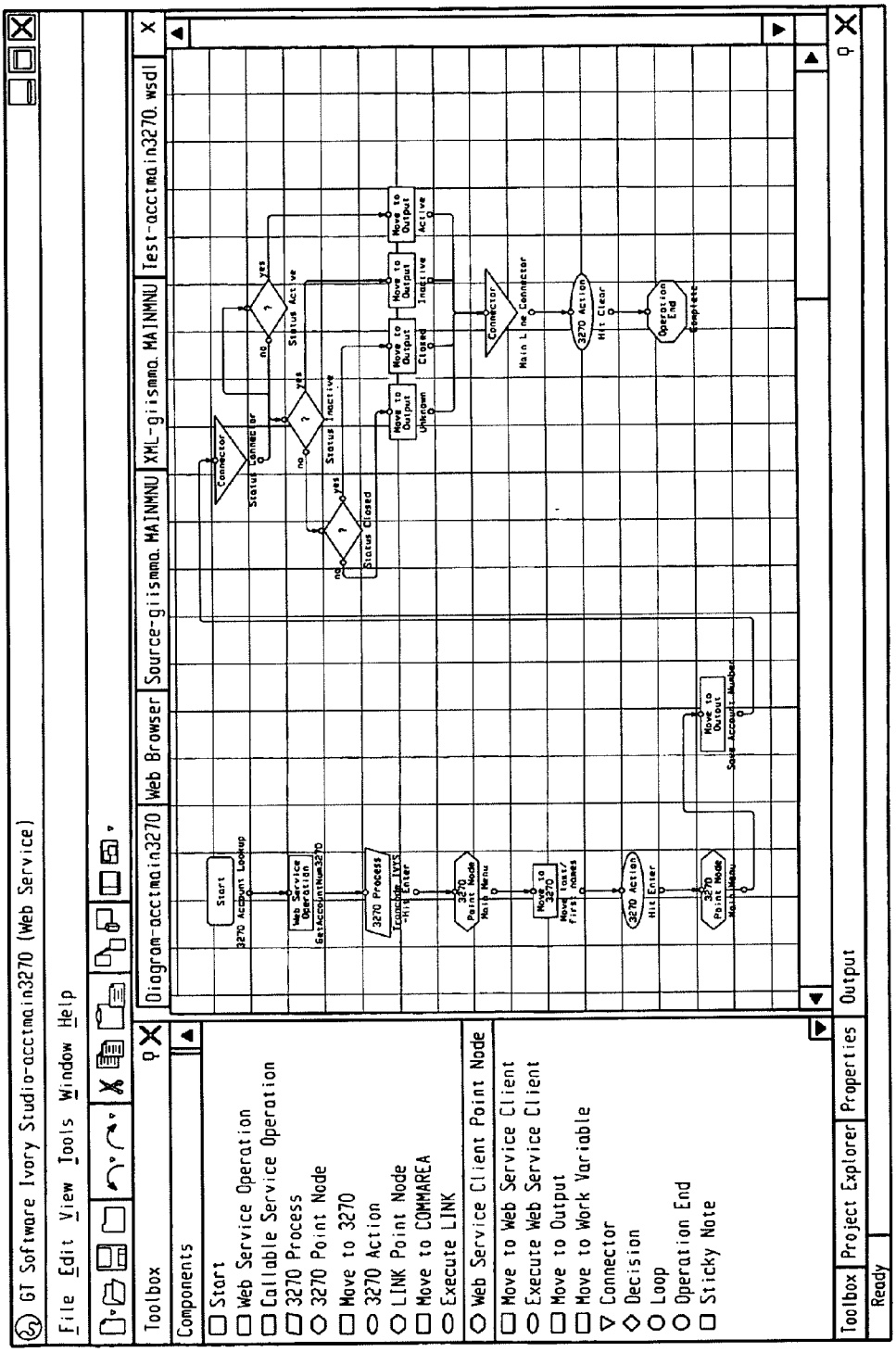

GT Software Ivory Studio-acctmain3270 (Web Service)
File Edit View Tools Window Help
Toolbox
Components
Start
Web Service Operation
Callable Service Operation
3270 Process
3270 Point Node
Move to 3270
3270 Action
LINK Point Node
Move to COMMAREA
Execute LINK
Web Service Client Point Node
Move to Web Service Client
Execute Web Service Client
Move to Output
Move to Work Variable
Connector
Decision
Loop
Operation End
Sticky Note
Diagram-acctmain3270
Web Browser
Source-giismma.MAINMNU
XML-giismma.MAINMNU
Test-acctmain3270.wsdl
Toolbox
Project Explorer
Properties
Output
Ready

Fig. 25

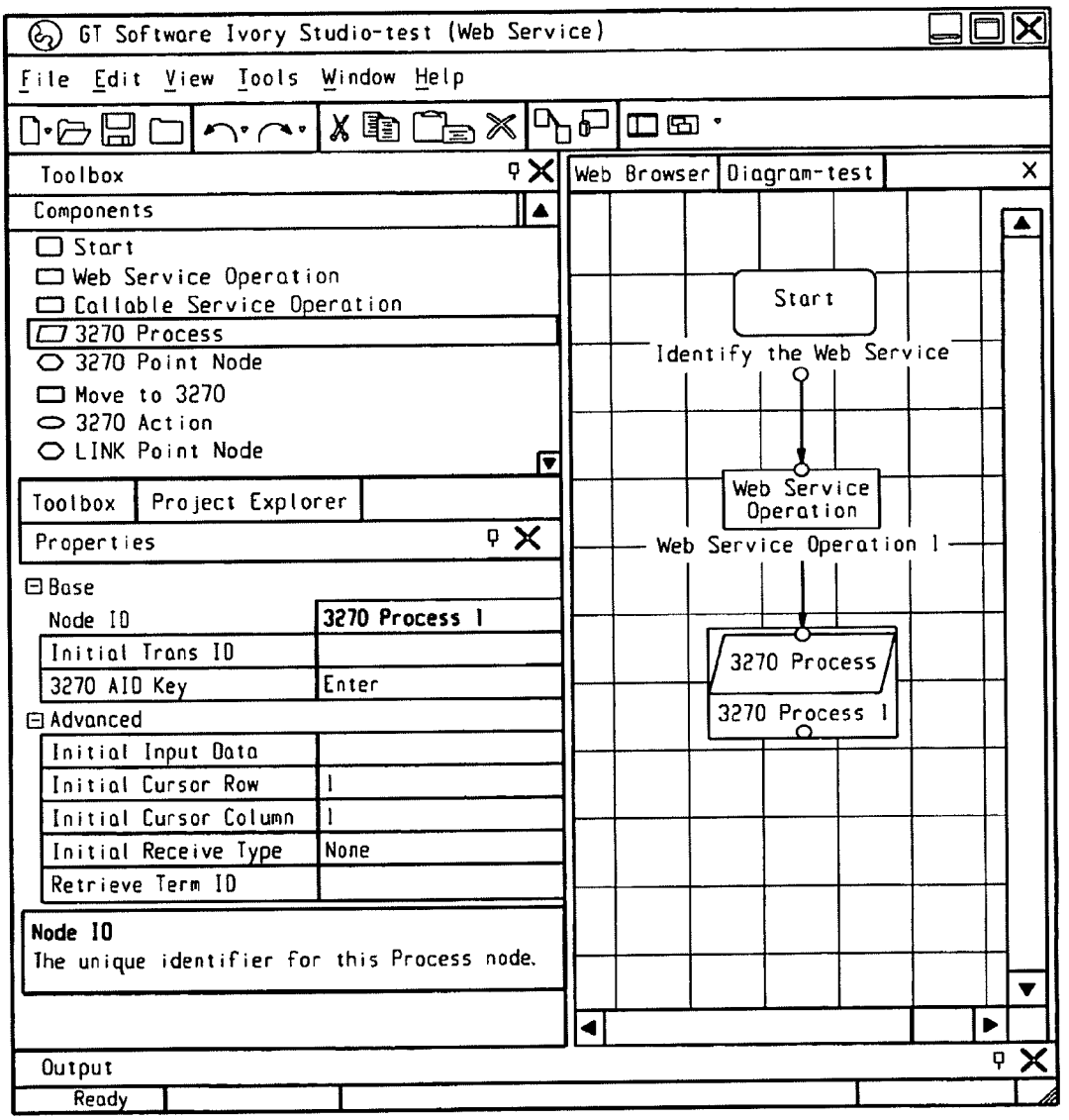

GT Software Ivory Studio-test (Web Service)
File Edit View Tools Window Help
Toolbox
Components
Start
Web Service Operation
Callable Service Operation
3270 Process
3270 Point Node
Move to 3270
3270 Action
LINK Point Node
Toolbox
Project Explorer
Properties
Base
Node ID
3270 Process 1
Initial Trans ID
3270 AID Key
Enter
Advanced
Initial Input Data
Initial Cursor Row
1
Initial Cursor Column
1
Initial Receive Type
None
Retrieve Term ID
Node ID
The unique identifier for this Process node.
Web Browser
Diagram-test
Start
Identify the Web Service
Web Service Operation
Web Service Operation 1
3270 Process
3270 Process 1
Output
Ready

Fig. 26

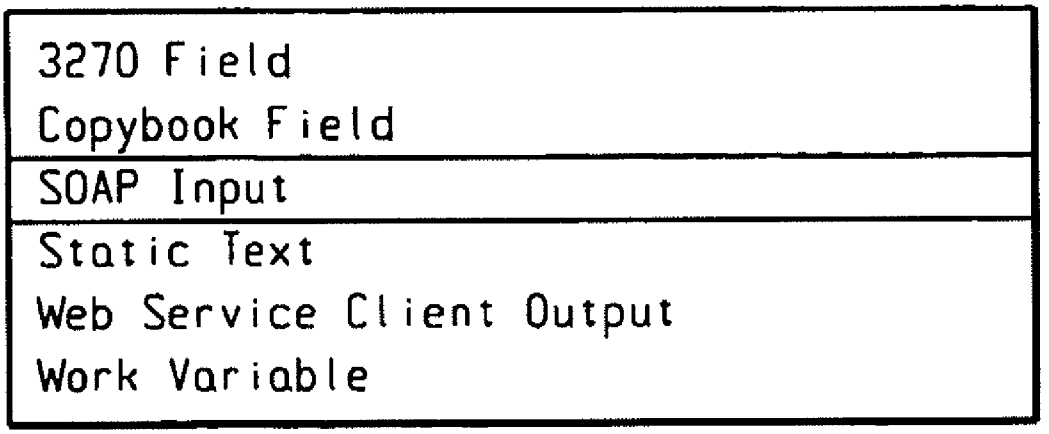

3270 Field
Copybook Field
SOAP Input
Static Text
Web Service Client Output
Work Variable

Fig. 27

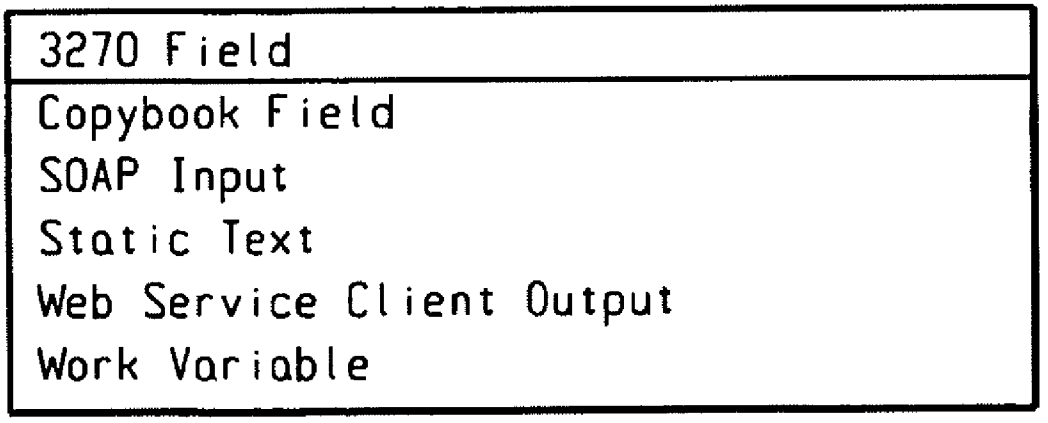

3270 Field
Copybook Field
SOAP Input
Static Text
Web Service Client Output
Work Variable

Fig. 28

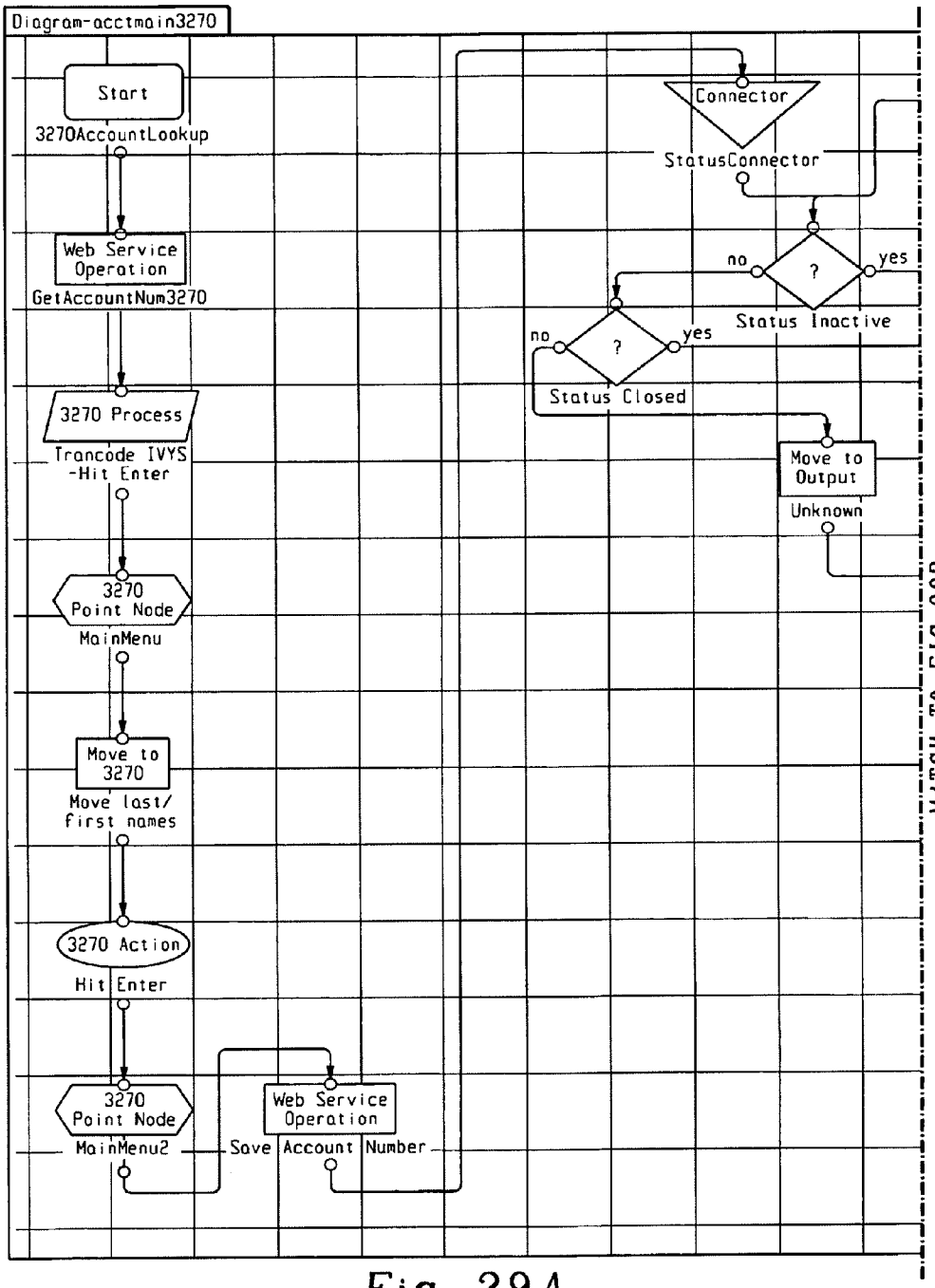


Diagram-acctmain3270
Start
3270AccountLookup
Web Service Operation
GetAccountNum3270
3270 Process
Trancode IVYS -Hit Enter
3270 Point Node
MainMenu
Move to 3270
Move last/ first names
3270 Action
Hit Enter
3270 Point Node
MainMenu2
Web Service Operation
Save Account Number
Connector
StatusConnector
no
?
yes
Status Inactive
no
?
yes
Status Closed
Move to Output
Unknown
MATCH TO FIG.29B

Fig. 29A

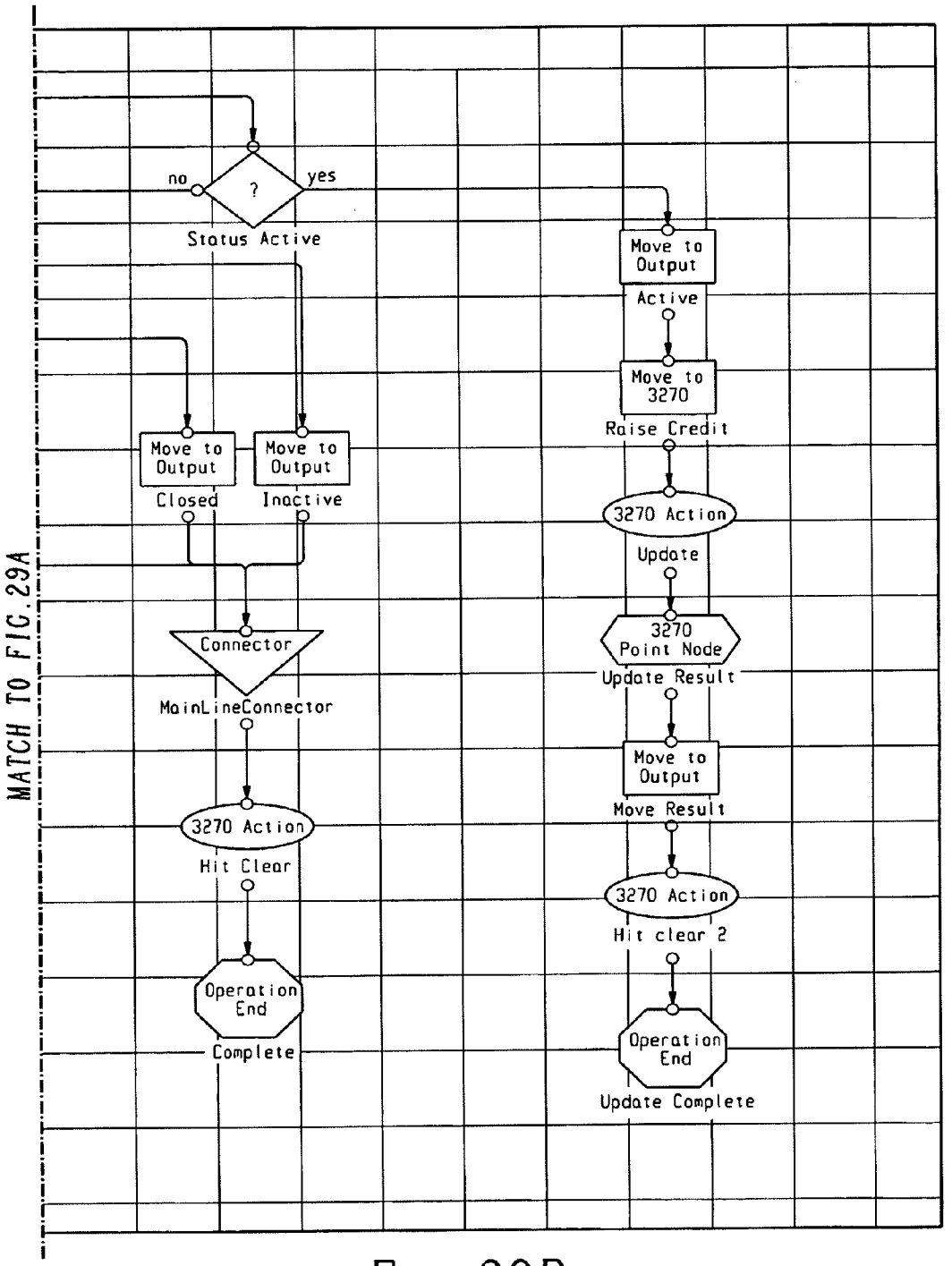

MATCH TO FIG. 29A
no
?
yes
Status Active
Move to Output
Closed
Move to Output
Inactive
Connector
MainLineConnector
3270 Action
Hit Clear
Operation End
Complete
Move to Output
Active
Move to 3270
Raise Credit
3270 Action
Update
3270 Point Node
Update Result
Move to Output
Move Result
3270 Action
Hit clear 2
Operation End
Update Complete

Fig. 29B

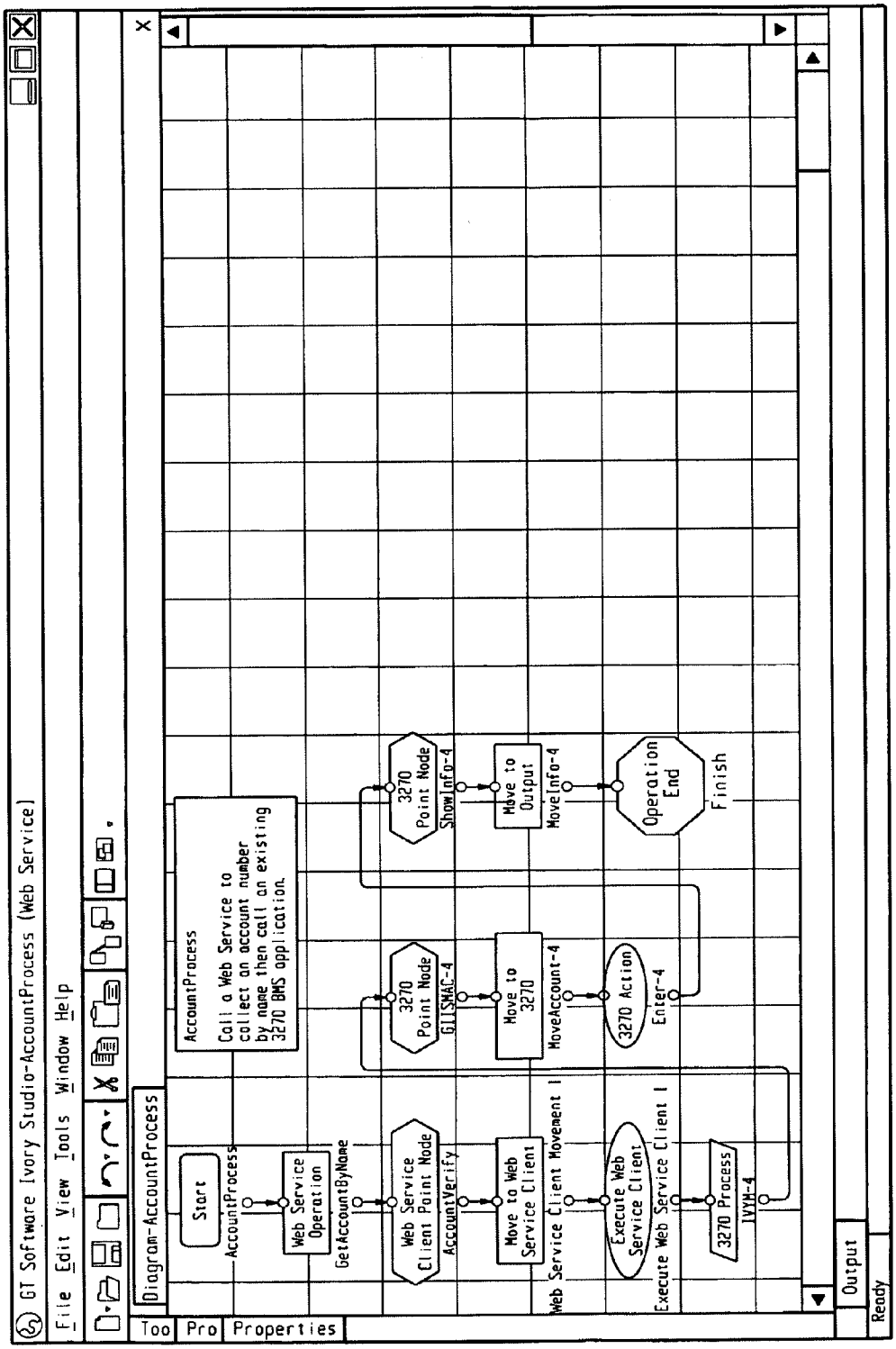

GT Software Ivory Studio-AccountProcess (Web Service)
File Edit View Tools Window Help
Too
Pro
Properties
Diagram-AccountProcess
Start
AccountProcess
Web Service Operation
GetAccountByName
Web Service Client Point Node
AccountVerify
Move to Web Service Client
Web Service Client Movement 1
Execute Web Service Client
Execute Web Service Client 1
3270 Process
IVYH-4
AccountProcess
Call a Web Service to collect an account number by name then call an existing 3270 BMS application.
3270 Point Node
GIISMAC-4
Move to 3270
MoveAccount-4
3270 Action
Enter-4
3270 Point Node
ShowInfo-4
Move to Output
MoveInfo-4
Operation End
Finish
Output
Ready

Fig. 30

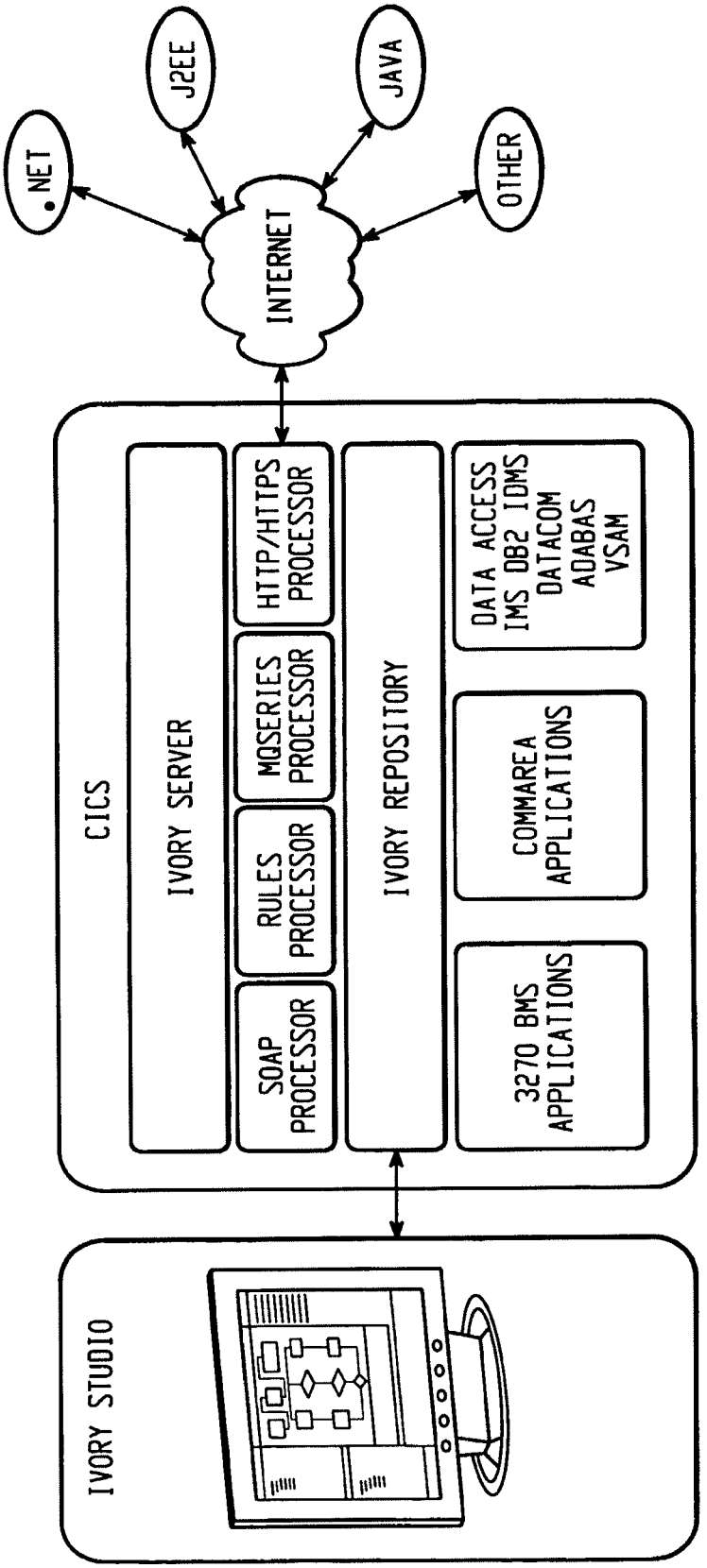

.NET
J2EE
JAVA
OTHER
INTERNET
CICS
IVORY SERVER
SOAP PROCESSOR
RULES PROCESSOR
MQSERIES PROCESSOR
HTTP/HTTPS PROCESSOR
IVORY REPOSITORY
3270 BMS APPLICATIONS
COMMAREA APPLICATIONS
DATA ACCESS IMS DB2 IDMS DATACOM ADABAS VSAM
IVORY STUDIO

Fig. 31

# SYSTEMS AND METHODS FOR WEB SERVICE FUNCTION, DEFINITION, IMPLEMENTATION, AND/OR EXECUTION

## CROSS REFERENCE TO RELATED APPLICATIONS

**[0001]** This patent application is a continuation of copending U.S. patent application Ser. No. 11/129,597, filed May 13, 2005, entitled SYSTEMS AND METHODS FOR WEB SERVICE FUNCTION, DEFINITION, IMPLEMENTATION, AND/OR EXECUTION, which claims the benefit of U.S. Provisional Patent Application No. 60/571,652, filed May 14, 2004, the disclosures of which are incorporated herein by reference in their entirety.

## BACKGROUND

**[0002]** The present application is directed to systems and methods for Web Service function, definition, implementation, and/or execution. The Internet is a global network of connected computer networks. Over the last several years, the Internet has grown in significant measure. A large number of computers on the Internet provide information in various forms. Anyone with a computer connected to the Internet can potentially tap into this vast pool of information.

**[0003]** The information available via the Internet encompasses information available via a variety of types of application layer information servers such as SMTP (Simple Mail Transfer Protocol), POP3 (Post Office Protocol), GOPHER (RFC 1436), WAIS, HTTP (Hypertext Transfer Protocol, RFC 2616) and FTP (File Transfer Protocol, RFC 1123).

**[0004]** One of the most wide spread methods of providing information over the Internet is via the World Wide Web (the Web). The Web consists of a subset of the computers connected to the Internet; the computers in this subset run HTTP servers ("Web servers"). Several extensions and modifications to HTTP have been proposed including, for example, an extension framework (RFC 2774) and authentication (RFC 2617). Information on the Internet can be accessed through the use of a Uniform Resource Identifier ("URI," RFC 2396). A URI uniquely specifies the location of a particular piece of information on the Internet. A URI will typically be composed of several components. The first component typically designates the protocol by which the address piece of information is accessed (e.g., HTTP, GOPHER, etc.). This first component is separated from the remainder of the URI by a colon (':'). The remainder of the URI will depend upon the protocol component. Typically, the remainder designates a computer on the Internet by name, or by IP number, as well as a more specific designation of the location of the resource on the designated computer. For instance, a typical URI for an HTTP resource might be:

**[0005]** http://www.server.com/dir1/dir2/resource.htm

**[0006]** Where HTTP is the protocol, www.server.com is the designated computer name and /dir1/dir2/resouce.htm designates the location of the resource on the designated computer. The term URI includes Uniform Resource Names ("URNs") including URNs as defined according to RFC 2141.

**[0007]** Web servers host information in the form of Web pages; collectively the server and the information hosted are referred to as a Web site. A significant number of Web pages are encoded using the Hypertext Markup Language ("HTML") although other encodings using Standard Generalized Markup Language ("SGML"), eXtensible Markup Language ("XML"), Dynamic HTML ("DHMTL") (the combination of HTML, style sheets and scripts that allows documents to be animated) or Extensible HyperText Markup Language ("XHTML") are possible. The published specifications for these languages are incorporated by reference herein; such specifications are available from the World Wide Web Consortium and its Web site (http://www.w3.org). Web pages in these formatting languages may include links to other Web pages on the same Web site or another. As will be known to those skilled in the art, Web pages may be generated dynamically by a server by integrating a variety of elements into a formatted page prior to transmission to a Web client. Web servers, and information servers of other types, await requests for the information from Internet clients.

**[0008]** Client software has evolved that allows users of computers connected to the Internet to access this information. Advanced clients such as Netscape's Navigator and Microsoft's Internet Explorer allow users to access software provided via a variety of information servers in a unified client environment. Typically, such client software is referred to as browser software.

**[0009]** Web services further facilitate access to information on the Internet by computer users. Web services address the need to integrate legacy mainframe applications by acting as platform-independent interfaces that allow communication with other applications using standards-based Internet technologies, such as HTTP and XML. With traditional integration techniques, there are multiple point-to-point communication and data conversions that may change as new applications are integrated or data formats change. Web services simplify integration by reducing the number of Application Program Interfaces ("API") to one, Simple Object Access Protocol ("SOAP") and the number of data formats to one, XML. SOAP overlays XML and transmits data in a way that can be understood and accepted by Web browsers and servers. The XML is also human readable. Web services allow programmers to make databases and/or other applications available across the Web for other programmers to access them and link the applications together to provide services.

**[0010]** Web services using the request and response methods are further described as being a Service Oriented Architecture ("SOA") approach to integration of electronic business applications or processes. A service-oriented architecture is essentially a collection of services. These services communicate with each other as described previously. The communication can involve either simple data passing or it could involve two or more services coordinating some activity. The methods of connecting services to each other involve the protocols and transport methods of SOAP.

**[0011]** Web Services Description Language ("WSDL") is a format for describing a Web services interface. It is a way to describe services and how they should be bound to specific network addresses. The WSDL includes three parts: definition, operations and service bindings.

**[0012]** WSDL definitions are generally expressed in XML and include both data type definitions and message definitions that use the data type definitions. These definitions are usually based on some agreed upon XML vocabulary. This agreement could be within an organization or between organizations. Vocabularies within an organization could be designed specifically for that organization. They may or may not be based on some industry-wide vocabulary. If data type

and message definitions need to be used between organizations, then most likely an industry-wide vocabulary will be used.

**[0013]** WSDL operations are grouped into port types. Port types define a set of operations supported by the Web service.

**[0014]** WSDL service bindings connect port types to a port. A port is defined by associating a network address with a port type. A collection of ports defines a service. This binding is commonly created using SOAP protocols and transport methods.

**[0015]** IBM created a SOAP interface for CICS (Customer Information Control System) which only supported a one-to-one relationship between the SOAP request and to the application code. This process does not provide automatic parsing and processing between the SOAP XML and the application communication areas. It also fails to provide any method for processing 3270 BMS applications. The IBM process provides neither flow processing nor graphical interface tooling with the SOAP process.

**[0016]** A main problem of most computer-based systems is their lack of ability to create composite processing in a Web service environment. A primary aspect of the Web service software described herein is to provide composite-based application processing using existing application software. Present systems lack support for a variety of functionality in the Web service area. Present systems do not support application flow and processing multiple applications via a single SOAP request would provide a much-needed process to the market place. Further, programmers have a significant learning curve to build the required XML and WSDL files required for SOAP processing.

## SUMMARY

**[0017]** The present application is directed to systems and methods for Web services function, definition, implementation, and/or execution. In one aspect, Web services are defined with respect to one or more functions available from applications executing on one or more remote systems. In a further aspect, such definitions are used to generate a programmatic implementation that is communicated to a Web services server executing on, or in communication with, the remote system(s). In yet another aspect, Web services clients can then use the defined service by posting appropriate requests to the Web services server and receiving back from that server a response encoding the results of performing the requested Web service.

**[0018]** One application for Web service definition and/or development may be referred to herein as the studio or modeling software. The modeling software can also be described as the application flow designer for the patent application. The modeler can preferably be implemented in software executable on a typical computer having a system data store ("SDS") and a system processor; however, the modeler functionality, or portions thereof, may be implemented in whole, or in part, via hardware. In addition, or instead, the modeler functionality, or portions thereof, can be embodied in instructions executable by a computer, where such instructions are stored in and/or on one or more computer readable media.

**[0019]** An application for generating a programmatic implementation that is communicated to a Web services server may be referred to herein as a server or rule-based flow engine. The manager can preferably be implemented in software executable on a typical computer having an SDS and a system processor, however, the server functionality, or portions thereof, may be implemented in whole, or in part, via hardware. In addition, or instead the server functionality or portions thereof, can be embodied in instructions executable by a computer, where such instructions are stored in and/or on one or more computer readable media. In some implementations the server functionality can be standalone, performed within the studio or performed within the server.

**[0020]** A Web services server preferably incorporates an SDS, a system processor, and one or more interfaces to one or more communications channels that may include one or more interfaces to user workstations over which electronic communications are transmitted and received. In addition, or instead, the server functionality, or portions thereof, can be embodied in instructions executable by a computer, where such instructions are stored in and/or on one or more computer readable media.

**[0021]** A Web services client ("Requestor") executing on a computer communicates a Web service request to the Web services server ("Provider"). The Web services server executes the programmatic implementation of the defined Web service to generate a response. The response is then communicated to the Requestor. The Provider may execute one or more applications on the system execute the server and/or one or more remote systems in order to generate the response.

**[0022]** In each of the studio and server processes, the system processor is in communication with the respective SDS via any suitable communication channel(s); system processor may further be in communication with the one or more communication interfaces via the same, or differing, communication channel(s). Each system processor may include one or more processing elements that provide electronic communication reception, transmission, interrogation, analysis, processing and/or other functionality. In some implementations, the system processor can include local, central and/or peer processing elements depending upon equipment and the configuration thereof. It should be noted that the modeler, manager, server (and client) are summarized above as discrete components; however, these various components all together, or taken in any selected grouping, could be implemented within a single execution environment where a particular system processor and/or SDS could support one or more such components. Each SDS may include multiple physical and/or logical data stores for storing the various types of information. Data storage and retrieval functionality may be provided by either the system processor or data storage processors associated with, or included within, the SDS.

**[0023]** The studio provides an automated graphical process of collection of information to define and build the Web service process using an SOA. The studio process provides a graphical flow of the application processes required to build a request and response SOAP-based Web service. The graphical flow or model is then used to create the rules or execution path the server must follow to provide the requested SOAP response. Composite application processing can be provided via the modeling or rules-based process, the designer also provides for external logic processing to control the logic flow through the multiple applications.

**[0024]** The manager may optimize its data movement processing and lower storage requirements by only holding data or storage as long as needed. Work meta data fields can be used to dynamically modify control settings at runtime. This process allows the modeler and manager to communicate changes to the flow before runtime that take place at runtime.

The manager can allow for called projects, so that an application program can call other application flows at any time based on application or business requirements. This can provide a very powerful way to extend working application flows. An application flow can call other deployed application flows at any time during its processing.

**[0025]** In another aspect, a flow processing engine is disclosed in some implementations that processes movement of data to and from standard application meta data formats to and from the SOAP XML meta data formats. This process reduces the need of the application programmer having to know or understand SOAP and XML processing and having to code additional application program to process SOAP and XML. This can significantly reduce the time to market and the possibility for errors to be introduced into the process. In a further aspect, some implementations of the modeler disclosed herein can increase programmer productivity by allowing a drag and drop interface for building application flows and data movements between one or many applications. Development time can be greatly reduced because the need to build routines to parse and move the XML data to/from existing meta data formats is eliminated.

**[0026]** A further feature of some implementations includes processing pure SOAP, XML and WSDL creation using a composite application process that allows mapping and data movements without any code to be created and executed. Some such implementations may provide processing via a runtime image that processes the XML rules and instructions for processing the application flow, external logic and data movement processing.

**[0027]** Additional advantages will be set forth in part in the description which follows, and in part will be obvious from the description, or may be learned by practice of the systems and methods described herein. The advantages of the disclosed systems and methods will be realized and attained by means of the elements and combinations particularly pointed out herein. It is to be understood that both the foregoing general description and the following detailed description are exemplary and explanatory only and are not restrictive of the invention as claimed below.

## BRIEF DESCRIPTION OF DRAWINGS

**[0028]** The accompanying drawings, which are incorporated in and constitute a part of this specification, illustrate various aspects of the disclosed systems and methods and together with the description, serve to explain and/or exemplify their principles.

**[0029]** FIG. **1** depicts an example graphical user interface for a typical modeler environment.

**[0030]** FIG. **2** depicts an exemplary import interface for an example modeler.

**[0031]** FIG. **3** depicts an exemplary data movement interface for an exemplary modeler.

**[0032]** FIG. **4** depicts an example rule set generated in a typical modeler environment.

**[0033]** FIG. **5** depicts a directory listing for a file repository in an exemplary modeler.

**[0034]** FIG. **6** provides a sample FTP interface from an exemplary modeler.

**[0035]** FIG. **7** is an exemplary Server Administration screen for server configuring and managing.

**[0036]** FIG. **8** depicts an exemplary TN3270 Client accessing a TN3270 Server (Emulator).

**[0037]** FIG. **9** depicts a potential Integrated Development Environment (IDE) for an exemplary modeler.

**[0038]** FIG. **10** is an exemplary FTP interface used in an exemplary modeler.

**[0039]** FIG. **11** is a block diagram of various components or functions of an exemplary modeler and server.

**[0040]** FIG. **12** is a graphic representation of a potential architecture for an exemplary modeler or server.

**[0041]** FIG. **13** is an example screen representing an IDE with MDI, drag and drop and dockable windows as used in some potential implementations.

**[0042]** FIG. **14** depicts a potential tool box window of an exemplary modeler.

**[0043]** FIG. **15** depicts a potential project explorer window of an exemplary modeler.

**[0044]** FIG. **16** depicts a potential property window of an exemplary modeler.

**[0045]** FIG. **17** provides a graphical representation of an example graphical diagram window.

**[0046]** FIG. **18** represents a browser window within exemplary modeler.

**[0047]** FIG. **19** depicts an exemplary test window.

**[0048]** FIG. **20** depicts an exemplary SOAP request window as used by the exemplary test window of FIG. **19**.

**[0049]** FIG. **21** depicts an exemplary SOAP response window used by the exemplary test window of FIG. **19**.

**[0050]** FIG. **22** represents an example output window.

**[0051]** FIG. **23** depicts a screen from mainframe sample application.

**[0052]** FIG. **24** depicts a sample data input into the example screen in FIG. **23**.

**[0053]** FIG. **25** depicts an exemplary modeler project representing a Web service to operate a mainframe application defined through graphical nodes.

**[0054]** FIG. **26** demonstrates the relationship between a property window and a node within an graphical diagram window in a particular example modeler.

**[0055]** FIG. **27** represents use of a drop-down menu for selecting a data sources for an exemplary node.

**[0056]** FIG. **28** represents use of a drop-down menu for selecting a data sources for an exemplary node.

**[0057]** FIG. **29** depicts a sample graphical Web service definition accessing a mainframe application.

**[0058]** FIG. **30** demonstrates graphical definition of a composite Web service within an exemplary modeler.

**[0059]** FIG. **31** depicts a typical deployment architecture for Web services using an exemplary modeler and server.

## DETAILED DESCRIPTION

**[0060]** Exemplary systems and methods are now described in detail. Referring to the drawings, like numbers indicate like parts throughout the views. As used in the description herein, the meaning of "a," "an," and "the" includes plural reference unless the context clearly dictates otherwise. Also, as used in the description herein, the meaning of "in" includes "in" and "on" unless the context clearly dictates otherwise. Finally, as used in the description herein, the meanings of "and" and "or" include both the conjunctive and disjunctive and may be used interchangeably unless the context clearly dictates otherwise; the phrase "exclusive or" may be used to indicate situation where only the disjunctive meaning may apply.

**[0061]** The hardware of a typical execution environment for one or more of the components supporting Web services definition, implementation and/or execution include a system

processor potentially including multiple processing elements, that may be distributed across the hardware components, where each processing element may be supported via a general purpose processor such as Intel-compatible processor platforms preferably using at least one PENTIUM class or CELERON class (Intel Corp., Santa Clara, Calif.) processor; alternative processors such as UltraSPARC (Sun Microsystems, Palo Alto, Calif.) and IBM zSeries class processors could be used in other implementations. In some implementations, Web services definition, implementation and/or execution (servicing) functionality, as further described below, may be distributed across multiple processing elements. The term processing element may refer to (1) a process running on a particular piece, or across particular pieces, of hardware, (2) a particular piece of hardware, or either (1) or (2) as the context allows.

**[0062]** Some implementations can include one or more limited special purpose processors such as a digital signal processor (DSP), application specific integrated circuits (ASIC) or a field programmable gate arrays (FPGA). Further, some implementations can use combinations of general purpose and special purpose processors.

**[0063]** The hardware further includes an SDS that could include a variety of primary and secondary storage elements. In one preferred implementation, the SDS would include registers and RAM as part of the primary storage. The primary storage may in some implementations include other forms of memory such as cache memory, non-volatile memory (e.g., FLASH, ROM, EPROM, etc.), etc.

**[0064]** The SDS may also include secondary storage including single, multiple and/or varied servers and storage elements. For example, the SDS may use internal storage devices connected to the system processor. In implementations where a single processing element supports all of the server/manger functionality and/or the modeler functionality a local hard disk drive may serve as the secondary storage of the SDS, and a disk operating system executing on such a single processing element may act as a data server receiving and servicing data requests.

**[0065]** It will be understood by those skilled in the art that the different information used in the systems and methods for Web service function definition, implementation, and/or execution as disclosed herein may be logically or physically segregated within a single device serving as secondary storage for the SDS; multiple related data stores accessible through a unified management system, which together serve as the SDS; or multiple independent data stores individually accessible through disparate management systems, which may in some implementations be collectively viewed as the SDS. The various storage elements that comprise the physical architecture of the SDS may be centrally located or distributed across a variety of diverse locations.

**[0066]** The architecture of the secondary storage of the system data store may vary significantly in different implementations. In several implementations, database(s) are used to store and manipulate the data; in some such implementations, one or more relational database management systems, such as DB2 (IBM, White Plains, N.Y.), SQL Server (Microsoft, Redmond, Wash.), ACCESS (Microsoft, Redmond, Wash.), ORACLE 8i (Oracle Corp., Redwood Shores, Calif.), Ingres (Computer Associates, Islandia, N.Y.), MySQL (MySQL AB, Sweden) or Adaptive Server Enterprise (Sybase Inc., Emeryville, Calif.), may be used in connection with a variety of storage devices/file servers that may include one or more standard magnetic and/or optical disk drives using any appropriate interface including, without limitation, IDE and SCSI. In some implementations, a tape library such as Exabyte X80 (Exabyte Corporation, Boulder, Colo.), a storage attached network (SAN) solution such as available from (EMC, Inc., Hopkinton, Mass.), a network attached storage (NAS) solution such as a NetApp Filer 740 (Network Appliances, Sunnyvale, Calif.), or combinations thereof may be used. In other implementations, the data store may use database systems with other architectures such as object-oriented, spatial, object-relational or hierarchical.

**[0067]** Instead of, or in addition to, those organization approaches discussed above, certain implementations may use other storage implementations such as hash tables or flat files or combinations of such architectures. Such files and/or tables could reside in a standard hierarchical file system. Such alternative approaches may use data servers other than database management systems such as a hash table look-up server, procedure and/or process and/or a flat file retrieval server, procedure and/or process. Further, the SDS may use a combination of any of such approaches in organizing its secondary storage architecture.

**[0068]** The hardware components may each have an appropriate operating system such as WINDOWS/NT, WINDOWS 2000 or WINDOWS/XP Server (Microsoft, Redmond, Wash.), Solaris (Sun Microsystems, Palo Alto, Calif.), or LINUX (or other UNIX variant).

**[0069]** In one implementation the server or manager executes on a z/OS or VSE/ESA platform and the modeler or studio executes under a WINDOWS 2000 or WINDOWS/XP operating system. The server or manager executes as a rules-based processing application using XML based instructions created by the modeler or studio software. The modeler or studio is a graphical tool for building application flows to allow processing of non-SOAP and SOAP-based applications as SOAP-based composite applications.

**[0070]** In some implementations, a graphical user interface is disclosed that allows business analysts and programmers to form a collaboration to build a new business process centered on Web services. For example, users may be business analysts who require a method of interfacing to a mainframe application without an in-depth knowledge of the programming and/or application execution environment. Other users might be application developers or technical support personnel tasked with building SOAP Web services for use by application servers. This aspect can allow a developer to bridge between the application logic and the business process needed to provide a SOAP Web service.

**[0071]** In one implementation of the methods and systems described herein, a client uses a graphical interface to build a business process by defining inputs and expected outputs and then stepping through the application using graphical icons or nodes for each step. This graphical user interface allows the modeling of a Web service via graphical objects. The graphical objects are connected using connection points to form a flow through the various applications or methods needed to create the single process or composite process web service. As seen in FIG. **1**, the application flow through the various nodes controls the logic processing for several applications which provide the unique function of composite processing.

**[0072]** Each of the functions, (for example, business logic functions such as Link Point 3270 process and 3270 Point or Web Service Client point nodes, start and stop, logic flow, input and output movement nodes or XML/data remapping)

may be represented by a graphical icon or node. The modeler software is resident on the client's workstation and converts the client's input into processing rules in a single format, for example XML. The modeler provides the server with the rules required to navigate or otherwise invoke a business logic process, a transactional or conversational type application or even Web services which exist on the same or external servers.

**[0073]** The server or manager is a rules-based engine used to process rules generated as instructions from the modeler. The composite processing of applications provided by the server are a direct result of building the application flow using a graphical design tool. This process provides a simple yet powerful process for building mainframe-based SOAP or SOA applications. The graphical tool serves as one facet of the overall systems and methods described herein. Use of this tool allows no additional programming to be required once the modeler tool has deployed the rules to the server or manager software. The server processes the incoming SOAP request envelope, and then processes the business logic to build the SOAP response envelope for the returned SOAP packet. The server may further allow various processes of additional functions or business logic to form a complete response. Service requestors may communicate with the server to discover the defined Web services and import the WSDL that is created by the modeler to describe and define the processing of the service. The modeler or studio tooling builds the WSDL file automatically for the client or user of the system and removes the need to have the knowledge of building these interface files. The WSDL files can then be used by other third party products such as application design tools to build the interface modules called SOAP clients or proxies to access the applications that are orchestrated by the modeler/studio and server/manager.

**[0074]** In some implementations, existing business logic and/or application information can be imported into the systems and methods described herein. Existing business logic and/or application information may be supported through particular formats such as BMS or copybooks. An interface can be provided for such importation such as depicted in FIG. **2**. Imported business logic and/or application information can then be used to build Web services.

**[0075]** The importing process provides a way to communicate between various different system types, and the meta data collected through importation allow systems designers to communicate in a known language. Additional meta data may be created and/or renamed, mainly SOAP input and output meta data that is to be exposed by the Web service can be named or described using new meta data names, which can then hide the fact that the back-end system is not a Web service-based application.

**[0076]** The imported meta data is preferably normalized into an XML format to allow it to be processed using standard XML parsers instead of having to use a unique parser. The imported data, and/or other meta data, can be viewed in a tree fashion using tooling provided by some implementations of the modeler. FIG. **3** depicts such an example.

**[0077]** The graphical building of a Web service may use imported information to define communication with business logic and/or application processes. Each node has a unique function and properties that define the Web service and the operations performed by the service. For example, in a particular implementation, a Start node can describe the Web service and can serve the anchor or parent node for all other operations. A Start node can further represent multiple operations and/or configuration settings. It can, for instance, describe the location (environmental) attributes of the Web service that you are building.

**[0078]** Once the Web service operations have been defined the user may select the correct processing or point node. Each processing or point node defines the information required to access the target source. For example, for 3270 interface operations the target could be a CICS transaction code, or for a COMMAREA application the target could be a CICS program name. Additional data sources such as DL/I, IMS, DB2, VSAM, could also be used. As users build the diagram or model of the Web service they may connect the nodes to form the logic or processing flow. This flow will later be translated into instructions for the rules-based SOAP server process. The syntax of the diagram is verified each time a node connection is attempted to insure a valid logic path and that node connection rules are correct. The logic path is traced to insure that the connection operation is to a valid parent and is not crossing Web service operation paths or boundaries. Each Web Service Operation node defines the expected SOAP input and output for the service operation path. The WSDL is created from the properties entered for each of the nodes. This document is an XML description of the interface methods for the Web service being created.

**[0079]** The modeler build process may in some instance provide a verification of the Web service at the same time as it creates the server rules, WSDL and HTML to define the service.

**[0080]** The descriptive HTML may be static or converted to a dynamic XSTL template that will build the HTML dynamically based on the XML of the WSDL. The WSDL generated by the modeler build process can now be used, in some implementations, by an execution test and/or debug tool. Such a test and/or debug tool potentially provides for further verification of the service definition and the generated Web service interfaces. These tools can dynamically process the service information (WSDL code) to create the user interface required to verify the deployed Web service. In such implementations, the WSDL definition may serve as the interface point between Java J2EE and .NET processing. The WSDL may be processed by popular Integrated Development Environment ("IDE") products that supply the capability to automatically build the Java, C# or other language interface code for processing the Web service described by the WSDL.

**[0081]** The server side instructions (rules) contain a mixture of execution and data flow. These instructions can be represented in XML as a tree structure. The XML tree is derived from the project diagram and its associated settings including data movements and/or properties. The diagram itself is stored as a mostly flat entity (although it may be represented in XML) so the resultant server instructions are not required to have the same appearance as the diagram. The format of rules is preferably set to provide the highest performance, as these rules will be executed for each request of the Web service. The rules may be compressed and/or optimized to improve performance of execution. Each execution node within the server instructions contains children nodes, one of which will receive control once the parent has completed its processing. The choice of the next child to dispatch is determined at runtime by the rules engine, but the rules instructions allow all possible choices to be specified. FIG. **4** depicts a representation of the rules generated by an exemplary modeler in one particular implementation.

**[0082]** The model designer has complete control over server processing logic flow. There are no ambiguous execution flows within a Web service operation; the model designer details completely how the server is to operate. Data flow is completely described within the modeling diagram process. The modeler creates the server side instructions via an "n-pass" algorithm applied to the diagram XML based on the number of paths created by the client in the diagram. Server processing is a single pass of the XML server rules instructions tree. The modeler pre-notifies the server, possibly via the rules, of any data movements required to be saved for later usage to optimize the server performance. This design places the complexity burden upon the modeler to be highly optimized in its rule creation resulting in performance benefits in the server rules engine.

**[0083]** The server may be a SOAP server based on the HTTP and SOAP protocols. The particular server may support processing standard HTTP and/or secure HTTPS requests. The repository may be a Hierarchical File System based on a CICS standard VSAM KSDS file; this unique function provides support for a UNIX/Windows based file system without the need for Unix System Services on CICS. The file repository may contain a command line processor for management of the file system via a standard CICS transaction. Command examples include, but are not limited to creation of file systems in the repository, directories and data files as depicted in FIG. **5**. The file system repository can also be managed from an FTP server provided such as with the SOAP server and/or via the FTP client supplied with the modeler. FIG. **6** provides an example of screen of management via an FTP server. The HTTP server and FTP server can follow the standard Internet RFCs for those protocols.

**[0084]** The server can provide the ability to map transactional 3270 and program-based applications into SOAP methods or objects. The heart of this process is the rules engine that is used to process the output rules instructions from the modeler. The server processes each node starting from the initial start point of the diagram. In a preferred implementation, each diagram contains the Web service name and the operations or methods the service provides. Once the method is selected from the SOAP Request envelope, the selected Web Service Operation node becomes the parent node and the logic tree that results will all branch from this common parent. For example, each node of the rule instruction set may cause the server to dispatch the function to handle the node operations. The node operations are optimized into the correct code page for the mainframe session.

**[0085]** In a preferred implementation, the HTTP server provides the base protocol support on top of CICS Web Services. This HTTP server provides administrative utilities to manage the SOAP services and FTP servers. An example administrative screen is seen in FIG. **7**. All discoveries of WSDL files and management of the HTML documentation can be stored in a Hierarchical File System (as part of the SDS).

**[0086]** Some implementations of the described systems and methods may incorporate a debugger or testing tool to read in the WSDL and extract all the operations that can be performed. The debugger lists the operations in a pull-down list. Once the user selects an operation from the list the input field, meta data will be used to build a tree view of the required input fields. The debugger or test tool may be used to input data for the SOAP Request envelope. For complex arrays the user may first define how many occurrences will be entered, and then the debugger will provide input area in the tree display for the occurrences of the complex type. The debug option may be added to the client modeler application and the server service provider. The debug operations may be a two-way communication path that will allow the client to know what step has been executed on the mainframe. The client notifies the server of the debug request by sending additional headers in the HTTP request to show the debugging client machine. The user may open a TCP/IP socket to listen for requests. As the server starts execution of the rules the server will send status information to the client. The modeler will show the current step in the diagram and will provide for breakpoints and other standard debug commands, such as looking at storage, setting new data values.

**[0087]** Some implementations of the described methods and systems incorporate an emulator. The emulator can be written in such a way to provide .NET access to the 3270 emulation via browser object tags, the browser may be used as the container for the emulator product. This may also extend beyond the browser into an API that allows programmatic control over the 3270/5250 applications. This will be a pure .NET solution and as such will allow any .NET language on a Microsoft platform be used to build new interfaces. The emulator can take advantage of performance improvements placed into the Windows .NET object used to build applications.

**[0088]** The emulator can be installed in classic windows fashion, or via a browser interface. An exemplary emulator screen is seen in FIG. **8**.

**[0089]** In some preferred implementations, the modeler process provides for building Web services by importing Basic Mapping Support macros ("BMS") and building a data structure in XML that matches the Application Data Structure ("ADS"). For example, in one implementation the model may provide a method for importing the CICS COMMAREA into an XML format much like the BMS macro layout. This provides a common layout and structure for the various application data structures that will be used.

**[0090]** In some preferred implementations, a user builds a business process by defining inputs and expected outputs and then stepping through the application using graphical icons for each step. Each of the functions, for example business logic functions, starts and stop, logic flow, or XML remap, is represented by an icon. The end result is a graphical design of the application from the Web service inputs to the final Web service response. The graphical models may be self-documenting or may also provide for process documentation to be entered using "sticky notes". The modeler interfaces with an IDE, which processes the menus and windows for user selections. The general design in a preferred implementation may include dockable and movable windows and the use of Multiple Document Interface ("MDI"). (See FIG. **5**.) Menus and screen literals may optionally be based in resource files to allow for support of languages other than English. FIG. **9** depicts an exemplary screen.

**[0091]** The modeler will allow users access to multiple applications in a single Web service by combining application available functions, in contrast to conventional systems that require having to make several calls. In order to improve compatibility with other systems, all or some files created by the modeler, including the project files, can be stored using an XML format. The modeler may incorporate functions to import copybooks that are converted to an easy to understand XML format, which allows for easy expansion of the resulting meta data dictionary. The modeler will also incorporate a

BMS macro importer to convert the BMS source into XML format. An FTP client can be provided to pull copybooks and/or BMS macros from the mainframe or other computer. An exemplary interface screen is depicted in FIG. **10**. The FTP client may include a meta data import function. Such a function supports ease of use by clients when importing meta data located on a remote computer system.

**[0092]** The modeler provides the server with all the rules engine information needed to process SOAP operations.

**[0093]** Manager software, for example, mainframe service routines, uses XML data collected via the modeling software to process the business rules defined in the graphical model by the user. Input to the manager will be the processing rules from the modeler and the SOAP packet. The input process will fire the manager, and it will process the business logic building a result for the returned SOAP packet.

**[0094]** In one preferred implementation, the server is a rules-based engine that processes the XML server instructions from the modeler. Each node is converted into a set of rules that allow the various processes of a Web service to be applied to information and procedures resident on a server, for example the CICS TS server or on IMS-based server processes.

**[0095]** The Start node is the initial setup and logical start of the Web service being created. The Start node defines the name of the Web service and various WSDL options such as the URI to invoke the Web service and the target namespace for the SOAP input/output fields that will be defined. The processing type of RPC (Remote Procedure Call) and Document is requested at this node.

**[0096]** The Web Service Operation node provides the logical name for the method or operation and provides the SOAP input and output structures. Each input field and its type is described at the Web Service Operation node. The output structure expected from the Web service is also defined at this node.

**[0097]** The LINK Point Node provides the interface point between the server and the Web service being created. The link point defines the name of the program to execute and the location where it should execute. CICS dynamic routing rules may be used when processing.

**[0098]** Some implementations may incorporate a 3270 Process node and a Point node. The Process node may cover all the setup information for the transaction code and the information required by CICS to start the 32370 process. The Point node is used to supply the current BMS mapset and map name so that data movements can be created.

**[0099]** Data movement nodes provide a tree structure for creation of move rules. Some examples of data movement nodes include Move to LINK, Move to 3270, and Move to Output. The movement process is based on a common code base so that all movement nodes have a similar function. The Move to LINK node, for example, may provide the method to set the initial values for the program that will be invoked. The Move to 3270 node may allow for moving data to the BMS map in order to provide input for the 3270 screen operation. In one implementation, the data that could be used for these move operations could be from static values, SOAP input request envelope, and any previously accessed 3270 map or CICS COMMAREA.

**[0100]** The Decision nodes provide the ability to add logic that will change the flow through the model. These nodes may be processed by the server and take action on data from any previous point node under the same Web Service Operation node. Complex operations may be defined using the Decision nodes in conjunction with Loop nodes. Decision nodes can target the current point node process for all comparison operations.

**[0101]** Logic Decision nodes may also be placed in the flow. These nodes allow the data to be examined and the result to modify the flow of the server rules to provide 2 paths for each decision node, so the logic paths or flows increase 2*n where n=the number of decision nodes. These nodes may provide the method for the logic flow to form a tree format. In conjunction with Decision nodes, Connector nodes may be incorporated to allow consolidation of logic paths. The Connector node may provide the method for a branch of logic to return to any logic path under the parent Web Service Operation.

**[0102]** Loop processing is also used within decision tree processing; a Loop node provides a method for returning to a previous node within the parent Web service operation. At times logic of applications running in CICS will require that they be executed using multiple iterations. The Loop node will allow the logic flow to return to a parent node. For 3270 BMS this might be an operation that is scrolling through several screens. For a COMMAREA application, it might be a program that requires more than one pass to collect all the returned data.

**[0103]** The Move to Output node allows for data to be moved from any previous point node with the same parent Web Service Operation node. The target of the movement data will be the SOAP response envelope.

**[0104]** The Connector nodes provide a very important process that allows several logic paths to branch out and then return to the main line logic flow.

**[0105]** The Operation End node is a logical placeholder to signal that the Web service operation has completed its task. All Web Service Operation node paths can connect to a single Operation End node.

**[0106]** A Calculation node will provide for mathematical operations to be added to the logic flow of the modeler. These nodes will process data from the application and will be used to modify the result set returned to the client. For example, the 3270 application may have a total account balance for the persons account. A Calculation node may provide support for the user who desires to process all accounts and return to a single total.

**[0107]** A Data Source node provides access to external file systems and database data, for example, DL/I, IMS, DB2, and other ISV databases. Each Data Source node will have unique options for each of the different Data Source databases or file systems supported. These will be added to the product on an as needed or on demand basis.

Exemplary Manager or Server Function

**[0108]** The following description lists some general features of an exemplary manager or server function that may be used along with a modeler or studio function to provide application flow processing and orchestration of applications. The manager or server processes all SOAP requests and handle the processing of application flow as described by the rules created by the modeler or studio process. Application orchestration provides a method to modify application flow without the need to modify the existing application code. FIG. **11** shows the various functions of the manager/server in one of the variations in which the process can exist. The studio process is the modeler that builds the rules used by the rules

processor to orchestrate application flows. The manager/server and modeler/studio architecture is depicted graphically in FIG. **12**.

**[0109]** The server process or manager process contains a set of code instructions to manage the various processing engines and may in some implementations automatically switch states between the various processing functions as needed to process an application flow or to orchestrate a composite application consisting of one or more of the application area or processes shown in the example.

**[0110]** The SOAP clients can be any .NET, J2EE, standard Java, third party or user created web service functions. The manager/server can itself call another manager/server running the same or different projects to complete a true distributed function. Functions may further be called from user written applications (not shown). During this process the manager or server is acting as an agent of the application and not that of a SOAP request. The processing may occur via API meta data structures instead of SOAP WSDL meta data XML files.

**[0111]** The following is a list of functions provided by a typical manager server for processing the application flow rules created by the modeler or studio functions.

**[0112]** Rules processing engine for processing execution rules provided by the modeler/studio software for orchestration of application processes along with handling all the dynamic movement of data to and from SOAP XML meta data to application specific meta data layouts. The algorithm used for application flow processing may work in conjunction with the memory management algorithm to insure that only the storage required is saved during the execution orchestration process.

**[0113]** HTTP or HTTPS processing of incoming SOAP requests, optionally the requests could be dropped onto an MQSeries queue for processing by the rules engine. This may include the security checking and processing required for verification that the process is valid for the passed security information.

**[0114]** The application meta data mapping and data transformation algorithm may be based on the imported meta data information collected from the modeler or studio execution rules instructions. This algorithm includes routines for processing data transformations between the various encoding methods of ASCII to/from EBCDIC. The server/manager may also process all the various COBOL application data types of, but not limited to: COMP, COMP-2 COMP-3, COMP-4 and POINTERS. String and decimal number processing can be processed by the data mapping functions to insure the correct decimal alignment options are processed.

**[0115]** Storage processing may be handled by management modules that insure only the data needed to process the movements in the project rules files are saved during the execution of the SOAP or MQSeries request.

**[0116]** A URI mapping module may control the processing options and security used for various SOAP requests based on any suitable technique; in one preferred implementation, a pattern matching algorithm is used.

**[0117]** The manager may also process WSDL discovery requests as a standard SOAP server using the ?WSDL option, but it processes the documentation of the WSDL into an HTML file that can be served out using the ?INFO request which displays the information collected by the modeler/studio functions into a human readable form of HTML so that other application developers can read and understand the requirements of the WSDL process. This can be advantageous in the debugging and analysis of the Web service.

**[0118]** The rules engine may be implemented a single common resource. In some implementations, this rules engine may be completely re-entrant so that it can handle multiple inbound requests.

**[0119]** The manager/server may also, in some implementations provide trace functions which create an XML description document that traces the functions of the last SOAP request call. The trace can be used to debug operations or look for problems.

**[0120]** The Callable Services Operation (CSO) module may provide an API to an application program which communicates with the CSO module via COMMAREA interface. The CSO module may operate to convert COMMAREA input requests to SOAP requests, execute the requested Web services, and deliver the results in COMMAREA output area.

Exemplary Modeler or Studio Function

**[0121]** The following description lists some general features for an exemplary modeler or studio that may be used along with a manager or server function to provide application flow processing and orchestration of applications. The modeler or studio function typically includes a high quality graphical display process to achieve the easy graphical building of application flow process. Icons and images are used to associate tasks and functions required to build or orchestrate an application or set of application flows. FIG. **13** depicts a sample screen of one example interface with features as more fully described below.

**[0122]** Multi-document window interface for controlling several containers of information in a single User Interface (UI).

**[0123]** Drag-and-drop dockable windows allow customization of the UI to match the clients or users preferences that support one or more features such as:

**[0124]** View or hide a window within UI

**[0125]** Resize a window within UI dynamically.

**[0126]** Place a window in different location within UI dynamically.

**[0127]** Tab between windows.

**[0128]** Each window is provided with scroll bars so that the information within the window can be viewed using the scroll bars if needed.

**[0129]** Remember the current user preference and configure the windows the same way next application start up.

**[0130]** Graphical toolbox of icons used to manage the various nodes required for building application flows and composite application processes.

**[0131]** Under certain conditions, the particular nodes made inactive depending on the application types that a developer builds.

**[0132]** The nodes will be placed in a logical order that a developer might be using. For an example, all COMMAREA related nodes are grouped together, placed in the order that they are typically used. All 3270 related nodes are grouped and placed in the order that they are typically used. All Web services related nodes are grouped and placed in the order that they are typically used.

[0133] The toolbar may be fully customizable to allow clients to place important or high use items on toolbars for easy access.

[0134] Action bar menus may be provided to allow processing of all functions, important or higher usage functions will be represented by icons to allow placement on the toolbar.

[0135] Import functions may be supported for the following:

[0136] Copybooks and/or source code to pull in COMMAREA or IMS application layouts

[0137] BMS macro importer to pull in 3270 meta data information—This may include other 3270 sources in the future.

[0138] Pull copybooks and/or source code from input source including local computer file system and remote computer file system.

[0139] Profile dialogs for manager server deployment, UDDI client, FTP client and 3270 client application processing. Profiles include information needed to create and maintain a connection to a host process.

[0140] Profile dialogs for management of internal/external tooling, namely access to applications created outside the product. For UDDI processing the dialogs may describe the various methods for importing WSDL files created by other functions or by the tool itself.

[0141] Project Explorer or file management process to show files associated with the project or file member to store all the execution flow processing XML information, may allow double clicking to process options, or right-click to show menu of processing options.

[0142] The application flow window or diagram window may be part of an MDI interface that provides a customizable grid layout process to allow nodes to be dragged to the diagram processing area and dropped. The dropping or selection process may activate a property window (part of a customizable dockable window interface). Once activated the property window can display the default or saved options for each node depending on the state of activation.

[0143] Certain nodes may provide collections of information about meta data or data movements. The collections may be accessible via a mouse click or keyboard selection of the collection expansion process. The collections may be presented in tree processing dialogs to allow for ease of processing when selecting or building meta data information.

[0144] The graphical Diagram window may contain rule processing to allow the flow engine to process correctly. A rigid and expansive set of ruling and verification may take place each time a node or tool is connected or disconnected to reduce errors. The flow processing may require validation in some implementations before a connection is allowed; this provides the client with immediate feedback as to the validity of the diagram, model or application flow. When meta data items are renamed or moved in the collection properties the change should be propagated through all nodes attached to the project.

[0145] The Properties window may display the required meta data information when a node is selected. When it is required, each meta data may cause additional windows to gather information. As an example, Web Service Client Point Node needs the WSDL location of the Web service. When the WSDL location is clicked, WSDL Discovery window is displayed so that it can gather information about the WSDL and populate the required information on the property window.

[0146] When the project is deployed to the manager or server, the status of build, verification or processing rules, and the status of upload process of processing rules of application and generated WSDL may be displayed.

[0147] In addition, some implementations include a build or verification process that can provide a secondary check of the properties and connections between the nodes to insure the flow processing rules are complete and valid before being deployed (uploaded) to the manager/server process.

[0148] After the completion of the build process, the generated files (e.g., WSDL files, rules processing files, HTML files) describing the application flow are stored. In some implementations, they are stored in the same directory structure of the computer system's file system.

[0149] A test utility window may contain areas to view and change WSDL location, operation name, URI location, HTTP version, timeout value, user ID, password, SOAP Request parameters, execution of the request, and/or display of the test result /SOAP Response. The interface in some implementations may include one or more of the following features:

[0150] Remember the current project information from the graphical Diagram window and pre-fill the information including WSDL and URI locations, operation name, and HTTP version.

[0151] Understand the current project information including the list of operations, the SOAP request and SOAP response parameters.

[0152] Allow test data to be entered

[0153] Save test data and reuse the saved test data.

[0154] View host-generated traces in a window.

[0155] A full help processing system may be provided in some implementations with the system to provide details on usage and processing of the various functions.

[0156] The following is a list of potential nodes in this exemplary implementation and a short description of its function:

[0157] Start—project or flow starting point: contains URI, WSDL location, descriptions and encoding styles and various other information for the rules processing engine. It also defines the name of the Web service.

[0158] Web Service Operation (WSO)—contains the collections for SOAP input/output and work field processing, defines the operation name for the web service defined in the start node. Allowing multi-operations is a significant feature for the mainframe application based version of the described systems and methods.

[0159] Callable Service Operation (CSO)—contains the collections for building an API layout to allow client applications to call the server/manager for processing rules based functions via application code instead of via SOAP based calls to activate composite SOAP applications.

[0160] 3270 Process—defines the 3270 BMS transaction codes and parameters needed to execute a transaction; this may contain processing of any 3270 application screen and need not be restricted to BMS only. After a process node there is normally a 3270 Point node as there is an implied 3270 action after the rules processing of the manager flows through a 3270 Process node.

**[0161]** 3270 Point—defines the map information for the 3270 process so that the correct meta data information is used for the following movement processing.

**[0162]** Move to 3270—collections of movements of data using meta data information from any previously access information.

**[0163]** 3270 Action—the action node defines the 3270 AID or key that should be pressed at the current time. After an action, another 3270 action or a collection of 3270 Point nodes are typically seen. The rules engine internally processes logic flow decisions to select the correct 3270 Point node based on the 3270 image or map that is currently active for the process. After each 3270 action, any number of different screen display options may be provided.

**[0164]** LINK Point Node—contains information to process a direct application call via CICS, IMS or other application platforms accessed via call or flat meta data layouts. Another feature is the ability to process several referenced storage areas in addition to just the single flat storage data area. This referenced storage process provides a method to pass very large data areas based on pointer or address reference points, similar to container or channel processing used by CICS TS 3.1. The meta data API area can be one single area or several areas.

**[0165]** Move to LINK—this is used as the collection point for movements of meta data from all previously referenced storage to the API area used to communicate to the application. Previous storage could be SOAP input, other applications called via LINK Point, 3270 Point or Web Service Client Point Nodes. The processes described herein may optimize the processing of storage based in part on the rules processor's knowledge at design time of all the data that may be moved or modified.

**[0166]** Execute Link Point Node—this node is mainly a marker in the logic flow of the diagram to show when the process is actually called. It allows for easy identification of the application flow being processed.

**[0167]** Web Service Client Point Node—identifies the processing of external Web service functions, they could be other Web services created by the present systems and methods or Web services created by other automated or manual methods. The process may provide an import function at this node to collect WSDL files for building the SOAP client needed to access the process. The process may build this dynamically via the modeler and creates a static call in the rules engine on the manager/server that will process the dynamic input and output for the referenced Web service operation. Special usage work variables can be processed as any object property, for example using the @@fieldname; option. This process allows any static option to be modified at runtime by the contents of the named work field.

**[0168]** Move to Web Service Client—this movement is like all other movement nodes with the exception that a WSDL is used to define the field names required for the operation.

**[0169]** Execute Web Service Client—this node marks when the requested call is placed by the manager/server rules processing engine. The SOAP client is dynamically built at runtime to process the Web service operation.

**[0170]** Move to Output—this node may be executed as many times as needed to move the entire SOAP packet to the requestor. The client has control over when and where these are placed in the diagram flow, as stated only one is needed, but as many as required may be coded. Any application data collected in the application flow (continuous connection points) is available to be moved to the SOAP Output, or API communication area depending on the type of service created.

**[0171]** Move to Work Variable—as discussed, work variables are meta data fields used to store information that can be used in the product runtime options or as storage for data that may or may not be needed for processing the application, for example switches to modify logic flow in a set of rules in a diagram.

**[0172]** Connector—connectors do just what the name implies, as decisions are used to create branches in the logic flow the connector can be used to bring many nodes into a single logic flow again. There are fewer rules about the placement of Connector nodes than any other node. In a preferred implementation, the connector cannot be used to join different operations in a project; it can only connect branches of an operation back to the same operation.

**[0173]** Decision—the logic flow modifier, clients can check work fields, SOAP input, application data from any of the various point nodes or for checking the ranges of boundary of arrays. The systems and methods described herein may support an array processor approach that allows for dynamic array processing for SOAP operations. Application arrays are normally static in nature, but can be processed using the same array boundary checking as for SOAP fields.

**[0174]** Loop—the rule execution flow processor may have the ability to loop through applications (calling repetitively) to process large amounts of information. The node has controls to limit loop processing to a defined amount or allow for decision processing to control the loop functions. Loops can also be used to process arrays of data in SOAP or application meta data.

**[0175]** Operation End—signals to the rules engine that all the processing that has taken place to this point is complete and the response should be returned. If the SOAP Fault option is turned on, the Operation End will create a SOAP fault that will be returned to the calling SOAP client; this is normally used for error processing, but could also signal to the calling application the process completed

**[0176]** Sticky Note—used to add documentation to the logic flow diagram, this is only informational data and is not used by the rules engine.

### Tool Box Window of One Exemplary Modeler or Studio Function

**[0177]** A graphical toolbox of icons may be used to manage the various nodes for building application flows and composite application processes. Such an exemplary interface is depicted in FIG. **14**.

**[0178]** When a node is depressed or selected, it can be dragged and dropped on a Diagram window.

**[0179]** In some implementations, the graphical toolbox can be context sensitive. Particular node types can be made inactive depending on the application type that a developer builds. As an example, FIG. **14** shows that Callable Service

Operation node is made inactive because the developer is not building a Callable service application.

**[0180]** The nodes are preferably placed in a logical order.

**[0181]** All applications typically start with a Start node followed by one or more Web Services Operation nodes, and end with one or more Stop nodes. Therefore, the Toolbox window may start with the Start node followed by various operational nodes including the Web Services Operation node and Callable Service Operation node.

**[0182]** All 3270 related nodes may be placed in a logical order and grouped together. For example, the 3270 Process, 3270 Point Node, Move to 3270, 3270 Action are ordered and grouped accordingly.

**[0183]** All LINK related nodes may be ordered logically and grouped together. For example, the LINK Point Node, Move to LINK, and Execute LINK nodes are ordered and grouped accordingly.

**[0184]** All Web Services related nodes may be placed in a logical order and grouped together. For example, the Web Service Client Point Node, Move to Web Service Client, and Execute Web Service Client nodes are ordered and grouped accordingly.

**[0185]** After executing an operation node (3270 Action, Execute LINK, or Execute Web Service Client), the resultant data from the operation is available. Using the available data, the data can be moved, checked and/or a decision is made to loop or connect to other logical place, and/or loop. Therefore, Move to Output, Move to Work Variable, Connector, Decision, and Loop nodes may be placed after the last execute node and grouped together.

**[0186]** Since the last logical operation is typically the Operation End node, it may be listed as the last item of the logical operation node.

**[0187]** Since the Sticky Note is typically used as a documentation tool, it may also be placed after the last logical operation node Operation End node.

| Component | Description |
|---|---|
| 3270 Action | Defines the AID operation that is required to continue (move to the next screen). |
| 3270 Point Node | Identifies the BMS mapset and map name available to lower level nodes. |
| 3270 Process | Defines the data and operations required for a 3270 Bridge process. Server will process the data collected by this node to invoke the 3270 transaction and start the data collection or processing actions that follow. |
| Callable Service Operation | Identifies inputs, outputs and work variables for the Callable service. |
| Connector | Provides the means to return to the main flow path after branching into multiple paths. The connector eliminates the need for complex or redundant tree structures. Note: Every connection into a Connector must originate from the same nearest parent point node. |
| Decision | Allows you to make comparisons between data values from point nodes, SOAP inputs, work variables, or static text. |
| Execute LINK | Performs the LINK to the program defined in the previous LINK Point Node, using the data that has been passed in the COMMAREA with one or more Move to LINK nodes. |
| Execute Web Service Client | Calls the Web service operation defined in the Web Service Client Point Node. Following this node, you can use the SOAP outputs of the Web Service Client Point Node as data sources for any subsequent movements or decisions. |
| LINK Point Node | Defines the CICS application program, as well as the input and output copybooks for the current operation. |
| Loop | Defines a point in the Web service operation where a loop must occur to collect all the required information. |
| Move to 3270 | Defines the movement of data to the nearest parent 3270 point node (BMS map) from valid data sources, including 3270 field, copybook field, SOAP input, work variable, or static text. |
| Move to COMMAREA | Defines the movement of data to the COMMAREA associated with the input copybook in the nearest parent LINK Point Node from valid data sources, including 3270 field, copybook field, SOAP input, work variable, or static text. |
| Move to Output | Defines the movement of data to a SOAP output (defined in the parent Web Service Operation node) from valid data sources, including the nearest parent point node, work variable, or static text. |
| Move to Web Service Client | Defines the movement of data (defined in the parent Web Service Operation node) to the SOAP inputs of the Web Service Client Point Node. |
| Move to Work Variable | Provides the ability to assign values to work variables that have been defined in the parent Web Service Operation node. This allows you to store information for later use in a Decision node or other movement node. |
| Operation End | Signals the end of the Web service operation. |
| Start | Marks the beginning of the diagram and identifies the URI of the Web service operation. |
| Sticky Note | Allows you to add comments to the diagram to make it more understandable; it does not perform any processing or data collection. |

-continued

| Component | Description |
|---|---|
| Web Service Client Point Node | Allows you to browse the Web for a WSDL document, which defines the available services and operations as well as the SOAP inputs and outputs for each operation. After you select a WSDL, you select the Web service and Web service operation you want to call. |
| Web Services | Defines a single operation within the Web service. There can be one or more Web service operations connected to the Start node in each Ivory project. This node also defines the SOAP inputs, SOAP outputs, and work variables for this operation. |

Project Explorer Window of One Exemplary Modeler or Studio Function

**[0188]** Some implementations may include a project explorer window that may be dockable within an MDI environment. An exemplary project explorer window is depicted in FIG. **15**. Project Explorer may display the contents of the project folder. When a project is created, a folder is specified where the project will reside. This folder may contain files that are not included in the studio project as explained below.

**[0189]** In some implementations such as depicted in FIG. **15**, files that are included in the project appear in bold type, and files that are not included appear in normal type.

**[0190]** When a file is "included" in a studio project, the file is available to be associated with a point node in the diagram. Being included in the project does not necessarily mean that the file is (or will be) used by the project.

**[0191]** "Included" files may be used to define project information that is external to modeler or studio, such as BMS mapsets and/or COBOL copybooks. When a developer imports a mapset or copybook, it can be automatically included in the currently open project.

**[0192]** Various features of preferred implementations are outlined as follows:

**[0193]** Allow manual inclusion of mapset and copybook files that have been copied into the project folder. In some implementations, the interface supports inclusion of a file through a right-click context menu.

**[0194]** View the file content by double-clicking in a separate modeler or studio display window.

**[0195]** Allow view of the 3270 map image if a map is clicked in a separate modeler or studio 3270 Emulator window.

Properties Window of One Exemplary Modeler or Studio Function

**[0196]** Various implementations may further provide a Property interface. One exemplary interface is depicted in FIG. **16** and described further below.

**[0197]** The Properties window in such implementations may display the required meta data information for a selected node. The operation of this exemplary interface is outlined as follows:

**[0198]** Once a node is activated, such as from a Diagram window as detailed below, the property window will display the default or saved options for each node depending on the state of activation.

**[0199]** The meta data required for the node is displayed. For an example, the Start node will require the node ID, Web services name, URI, WSDL location, encoding style, description, use URI prefix option, and server side tracing option.

**[0200]** When a particular meta data item is selected, the help information is displayed so that a developer knows what to do.

**[0201]** When required, each meta data may cause additional windows to be displayed to aid gather information. As an example, Web Service Client Point Node needs the WSDL location of the Web service. When the WSDL location is clicked, WSDL Discovery window is displayed so that it can gather information about the WSDL and populate the required information on the property window.

**[0202]** When required, each meta data may list the options that a developer can choose from. For an example, in a Start node, the Use URI Prefix During Deploy will list True or False options when selected.

Diagram Window of One Exemplary Modeler or Studio

**[0203]** Some implementations may include a Diagram window that may also be referred to as an application flow window. Such a Diagram window may be included as part of an MDI interface. An exemplary diagram window is depicted in FIG. **17**. Such an exemplary interface may include one or more of the following features.

**[0204]** A customizable grid layout process to allow nodes to be dragged to the diagram processing area and dropped from the Toolbox window.

**[0205]** The dropping or selection process will activate a Properties window. Once activated the properties window will display the default or saved options for each node depending on the state of activation.

**[0206]** A grid to allow proper alignment of diagram or graphical representation of a application.

**[0207]** A zooming capability to be able to resize the diagrams in 25, 50, 75, 100, and 200% of normal diagram size.

**[0208]** A copy and paste function to be able to duplicate the area of logic in other locations in the application flow or to be able to duplicate in other application flow or project.

**[0209]** An ability to move a diagram node or group of diagrams to anywhere in a canvas area.

**[0210]** An ability to flip connection points when right mouse is clicked on a node and select on the option. For an example, the typical Decision node connections are to have the YES connector on right side of the Decision node and NO connector on the left side of the Decision node. When a decision node is selected and right mouse is clicked, it

will display an option to flip nodes. If the flip node is activated, the YES and NO will be reversed.

**[0211]** When an application flow or project is saved reopened, the diagrams on the window should displayed as it was when it was saved.

Browser Window of One Exemplary Modeler or Studio

**[0212]** Some implementations may include a Browser window. An example of which is presented in FIG. **18**. The Browser window may be included as part of the MDI or be launched separately. The Browser window can display any documents including Internet site and text document. Such an interface may provide one or more of the following features:

**[0213]** Go back to display the previously displayed information.

**[0214]** Go forward to display the next information (previously displayed information).

**[0215]** The same functionality as the commercially available Internet browsers like Microsoft Internet ExplorerBrowser including ability to display any Internet documents.

**[0216]** When View Trace is selected from the Test Utility window, display the trace information by downloading trace information from the manager or server.

Exemplary Test Utility Window

**[0217]** Some implementations may provide a Test Utility that can allow the unit testing of Web Services which are to be deployed to the manager or server. An exemplary interface is provided in FIG. **19**. Such a test utility window may support one or more of the following features:

**[0218]** View and change WSDL location, service name, service operation, HTTP version, URI, and SOAP Action.

**[0219]** Remember the current project information from the graphical Diagram window and pre-fill the information including URI.

**[0220]** Access the WSDL specified in the WSDL location and populate the information like service name, service operation, HTTP version, URI, and SOAP Action.

**[0221]** Select a service operation from list of operations that the WSDL supports.

**[0222]** Select an HTTP version from the list of HTTP versions that the WSDL supports.

**[0223]** Set Proxy URI.

**[0224]** Set Timeout value so that the execution will stop after a certain time.

**[0225]** Set user ID and password for security checking if security is activated on the server or manger.

**[0226]** Execution of test by sending SOAP request parameters.

**[0227]** Save test data and reuse saved test data at a later time.

**[0228]** Loading of test data from a local or remote files system to be used in the test.

**[0229]** View host generated traces in Browser window.

**[0230]** View generated SOAP Request parameters.

**[0231]** Access the WSDL and parse the WSDL to find required SOAP Request parameters.

**[0232]** Generate and display the SOAP request parameters so that the tester can enter data. An example of these parameters is depicted in FIG. **20**.

**[0233]** View received SOAP response from the server or manager

**[0234]** Parse the WSDL to determine the SOAP Response parameters.

**[0235]** Generate and display the SOAP response parameters so that the tester can view returned data. An example of these parameters is depicted in FIG. **21**.

Exemplary Modeler Output Window

**[0236]** Some implementations may support an Output window. The Output window may be a dockable window supported in an overall MDI. FIG. **22** depicts an example Output window. An output window may support one or more of the following features:

**[0237]** When the project or application flow is built, the status of the build process is displayed. If there is any error during the build process, the error is displayed on this window to allow the debugging of the logic.

**[0238]** When the project or application flow is deployed to the manager or server, the status of build, verification or processing rules, and the status of upload process of processing rules of application and generated WSDL to be displayed.

Example Web Service Development Using One Exemplary Modeler or Studio

**[0239]** As an example, the development of a 3270 CICS application that searches for a name (first and last name) and displays information about the account including the account number, address, status, and account limit is described. Using this information, the 3270 operator, a helpdesk, could raise the account limit if the account status is active,or reopen the account if it is closed.

**[0240]** Similar programs have existed for decades in the mainframe environment. Those mainframe programs developed over many years cannot be replaced overnight, even though they may be cumbersome to use. For example, the following procedure outlines what a 3270 operator would typically do to find account information using a mainframe application:

**[0241]** From a 3270 terminal, the 3270 operator types the transaction code "IVYS" on a blank 3270 screen and presses the Enter key.

**[0242]** The program called by the IVYS transaction runs and displays a blank screen such as depicted in FIG. **23**.

**[0243]** Then the 3270 operator types the last name and first name of an account holder on the screen and presses the Enter key.

**[0244]** The program runs and displays account information, such as presented in FIG. **24**.

**[0245]** The3270 operator reads the displayed information and saves it by either writing notes or printing the screen.

**[0246]** The3270 operator studies the displayed information, for example to determine the account status. Based on that information, the 3270 operator performs a different task.

**[0247]** This is an example of a simple task, but it requires multiple 3270 operations to accomplish the task. In order to save this information, or to share it with other computer platforms, such as a PC, the saving or sharing must be performed manually. However, if this operation is converted to a Web service, this information can be shared with other platform computers and other applications.

**[0248]** In many cases, the people who wrote the decades-old mainframe applications are no longer available. Consequently, it is not easy or cost-effective to alter the original application program.

**[0249]** Using the modeler or server, the existing program and business process can be converted to a Web service through a graphical definition, and no changes to the original program are required. An exemplary graphical flow for this process is depicted in FIG. **25**.

**[0250]** The graphical diagram defines a Web service which represents the application or business process that the 3270 operator had to perform manually:

**[0251]** Typing the "IVYS" transaction ID and pressing the Enter key.

**[0252]** When a 3270 map is displayed, typing the account holders last name and first name, and pressing the Enter key.

**[0253]** When the response is displayed, reviewing the information to determine the next action.

In order to convert the sample application into a Web service, a developer drags a node from the Toolbox window and drops the node on the Diagram window. When the node is selected on the Diagram window, the Properties window displays current and default information about the node, which is modified by the developer to provide the information needed by the application. An exemplary interface is depicted in FIG. **26**.

**[0254]** First, describe the Web service environment.

**[0255]** The Start node provides the name of the Web service, the location of the WSDL, URI, encoding style and its description.

**[0256]** The Web Service Operation node provides the name of the operation, SOAP input and output parameters—and describes the operation or business process that it will perform below this node. It is possible for a single Web service to have multiple Web service operations. Using our example, the SOAP input parameters will be the last name and first name of the account holder. The SOAP output parameters will be the account number and the account status (Active, Closed, Inactive, or unknown).

Next, the developer begins to diagram the flow of the operation.

**[0257]** With the old mainframe application, the 3270 operator typed the "IVYS" transaction code on a 3270 screen and pressed the Enter key to begin. To mimic this action in the Web service, the developer drags a 3270 Process node to the Diagram window. In the Properties window for the node, the developer enters the IVYS initial transaction code and defines "Enter" as the 3270 AID key. This replaces the user action of typing the transaction code and pressing Enter.

**[0258]** As a result of pressing the Enter key on the mainframe application, a 3270 screen was displayed. To produce this result, the developer drags a 3270 Point node onto the Diagram window. In the Properties window, the developer identifies the name of the mapset and map that it should expect to receive.

**[0259]** When the 3270 mainframe screen was displayed, the 3270 operator would type the last name and first name of the account holder. The developer now drags a Move to 3270 node onto the Diagram window. The Move to 3270 node allows the movement of data from any source that was defined at a higher level than this node. The possible data source is described below. In the example, the developer will use the SOAP input to move the account number, and Static Text to set the "demographic" option. A drop-down menu such as seen in FIG. **27** could be used to specify this.

**[0260]** After entering the name, the 3270 operator pressed the Enter key. To duplicate this manual operation, the developer drags a 3270 Action node to the Diagram window.

**[0261]** After the 3270 operator pressed the Enter key, the 3270 map with the account number information was displayed. To provide the same function, the developer drags a 3270 Point Node to the Diagram window. The node properties in the Properties window describe the expected 3270 mapset, and map name to receive.

**[0262]** When the information was displayed, the 3270 operator had to read the information on the 3270 screen and validate the account number. In the Web service diagram, the developer does the same by dragging a Move to Output node to the Diagram window, saving the account number information. The Move to Output node allows the data movement to SOAP output parameters from any data source that was referenced prior to this node. In this example, the developer will pull the information from the 3270 field and save or move the account number to the SOAP output field. The use of the 3270 field can be selected via a pull-down menu such as presented in FIG. **28**.

**[0263]** Next, the 3270 operator determines whether the account is Active, Inactive, Closed, or Unknown (error). Based on the status of the account, the 3270 operator will perform different operations. The Web service uses a series of Decision nodes to check the account status. Each Decision node allows checking a value against a data source. Based on the result of the check, it will generate a YES or NO answer. The first Decision node will check whether the account is Active. If the answer is YES, it will move information to the SOAP output area indicating that this account is an active account. If the answer is NO, then the next Decision node is checked for Inactive status, and so on. Typical use of a Decision node is to direct flow of Web service logic and to find errors, end a process, or select the next operation. The Decision node can check whether any data was returned from the name look-up function. Then, based on the reply, it will build a different reply buffer. A Move to Output node is used to define data fields that will be returned in the SOAP response. Any data previously collected can be marked as information to be included in the SOAP reply.

**[0264]** The WSDL output fields are the typical method to return data to the requesting application. Notice that not all of the information available on the 3270 screen was of interest to the 3270 operator. Therefore, Web service developer provided only the data required by the operation. This reduces the storage requirement during processing and data transfer between machines reducing the network traffic requirements.

**[0265]** The series of manual operations that were required to gather simple account information was automated without writing a single line of code. The automation was done using the graphical representation of the 3270 operator's operations.

**[0266]** What if the 3270 operator needed to raise the credit limit, perform other tasks based on the account information. The 3270 operator would have to go through another series of manual operations. This process can also be automated through a Web service developed using the modeler or studio.

**[0267]** Continuing with the prior example, logic can be added to update or raise the credit limit based on the credit limit by executing additional 3270 BMS transactions. For example, FIG. **29** depicts such an enhanced graphical definition. This new task can reuse the original Web service in a new Web service.

**[0268]** In another example depicted in FIG. **30**, the original Web service, account look-up by name, was called as a Web Service Client Point Node and new logic was added to update the credit limit. The ability to consume an internal or external Web Services allows:

**[0269]** Simplification of logic.

**[0270]** Reuse of an existing Web services.

**[0271]** Interaction with a business partner with a different computer system.

**[0272]** Reuse of an external machines and programs, such as Windows NT and Unix.

**[0273]** If, for example, there is a need to validate a driver's license number, this Web service can consume a Web service published by a state agency to validate the driver's license number.

**[0274]** Going back to the original example, if the 3270 operator needs to execute a transaction on a Microsoft Windows application which publishes a new credit card after updating a credit limit, it can be automated by adding a Web Service Client Point Node to execute the Microsoft Windows-based Web service from the mainframe. Combining the 3270 operator's tasks to run as a single task can save time and money. This can all be done without writing a single line of mainframe program.

**[0275]** A mainframe Web service can be generated through diagraming a business process, such as getting insurance policy information for a user. A mainframe, for example, may have a separate application or business process for each type of insurance policy: home, car, and health. The described systems and methods can provide the ability to combine these separate business processes into a single business process implemented as one or more Web services.

**[0276]** The described systems and methods can include design and runtime functions. The design component can include a modeler and/or an interface to a mainframe application.

**[0277]** The modeler may provide the ability to change logic flow and the path of the original mainframe application. The runtime function component can include a mainframe Web service. The Web service interface is built around mainframe applications that reside on the mainframe. The described systems and methods can verify that the original mainframe application and Web service are in sync.

**[0278]** To summarize the functions, FIG. **30** provides an overview of the modeler and server in some of the systems and methods described herein. These systems and methods improve Web service development along several axis including one or more of the following:

Design Time:

**[0279]** Using the modeler/graphical IDE, the application flow and process is defined by dragging components to the Diagram window.

Deploy Time:

**[0280]** After building the Web services with the modeler/studio, instructions for the processing the Web services and the WSDL are uploaded to the server repository.

Run Time:

**[0281]** During the execution time, the server takes in SOAP request from applications written in .NET, J2EE, JAVA, or other programming languages, processes the Web services processing instructions, and returns the results to the application in a SOAP Response.

**[0282]** Throughout this application, various publications may have been referenced. The disclosures of these publications in their entireties are hereby incorporated by reference into this application in order to more fully describe the state of the art to which this application pertains.

**[0283]** The examples described above are given as illustrative only. It will be readily appreciated by those skilled in the art that many deviations may be made from the specific examples disclosed above without departing from the scope of the inventions set forth in this application and in the claims below.

**[0284]** The indentations and/or enumeration of limitations and/or steps in the claims that follow are provided purely for convenience and ease of reading and/or reference. Their usage is not intended to convey any substantive inference as to parsing of limitations and/or steps and/or to convey any substantive ordering of, or relationship between or among, the so indented and/or enumerated limitations and/or steps.

What is claimed is:

**1**. A system to facilitate the definition of web services, the system comprising:

a) a modeler which is operable to interface with a user to enable the user to generate an execution flow via a graphical interface for building one or more flow rules to build composite or single use web services; and

b) a manager that parses the flow rules to generate a programmatic implementation of the web services.

**2**. A method for graphically defining a web service, the method comprising the steps of:

a) providing an interface with the graphical display device via which the user can graphically define a web service using the user input device;

b) displaying on the graphical display device a graphical representation of at least a portion of a web service definition; and

c) storing in the system data store a digital representation of a graphically defined web service.

* * * * *