(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
26 September 2002 (26.09.2002)

PCT

(10) International Publication Number
WO 02/075474 A2

(51) International Patent Classification[7]: G06F

(21) International Application Number: PCT/EP02/02766

(22) International Filing Date: 13 March 2002 (13.03.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
0106439.3 15 March 2001 (15.03.2001) GB

(71) Applicant *(for all designated States except US)*: **SMITHKLINE BEECHAM P.L.C.** [GB/GB]; New Horizons Court, Brentford, Middlesex TW8 9EP (GB).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: **HUNTER, John, Robin** [GB/GB]; Computer Resource Management UK Ltd., 24 High Street, Newmarket, Suffolk CB8 9TJ (GB).

(74) Agent: **DOLTON, Peter, I.**; GlaxoSmithKline, Corporate Intellectual Property (CN9.25.1), 980 Great West Road, Brentford, Middlesex TW8 9GS (GB).

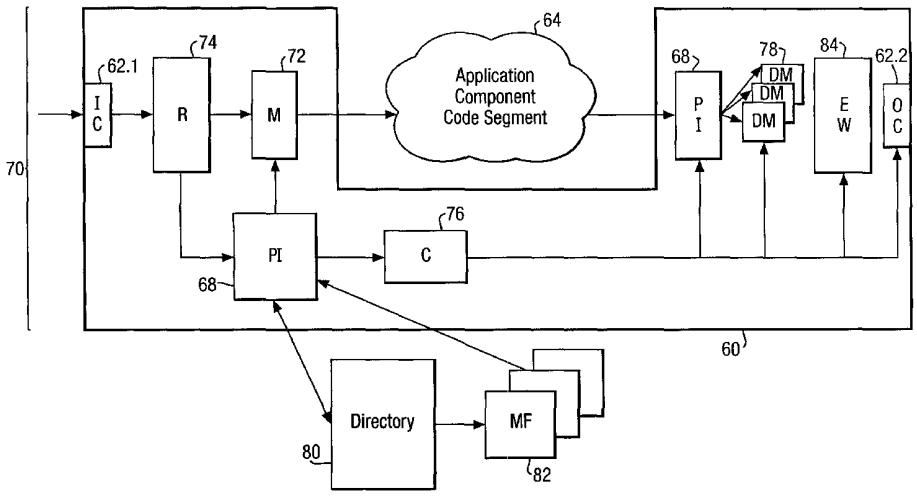
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: COMPUTER APPLICATION FRAMEWORK



(57) Abstract: An application framework supports at least one application that includes one or more application components. The framework includes a directory providing a logical definition of the application. The logical definition is referenced by an application context identifier and provides logical definitions of the application components that make ap the application. It further includes references to physical instances of the application components. An application component includes a harness that contains an application component code segment for a predetermined application context. The harness forms an interface between the application component code segment and an exterior of the application component. The harness is operable to apply a predetermined mapping for the application context to map between an external format for communication external to the application component and an internal format understandable to the application component code segment. In a particular example, LDAP is used as the directory protocol and XML is used to communicate between the framework and the application components and between application components.

WO 02/075474 A2

COMPUTER APPLICATION FRAMEWORK

BACKGROUND OF THE INVENTION

The invention relates to a framework for supporting computer applications that are based on application components.

Object oriented programming and other component-based software programming approaches are well established in the computing industry. A difficulty with any programming approach is the problem of ensuring that various parts of a potentially complex software system are compatible and inter-operable. This is particularly the situation where various programming languages may be used for different parts of the system and/or different parts are designed by different programmers. Indeed, it is the usual situation that the many software component parts of an application system will have been developed at different times, using different languages and/or platforms and/or by different programmers or programming teams. Also, as a result of continued development, the individual components may change with time as new versions become available.

The integration of components is very time consuming task and forms a major part of today's software development efforts. It would be desirable to reduce the amount of time needed to integrate software components to form a desired application and to reduce the total effort involved in maintaining applications through their life cycle. Accordingly, an aim of the present invention is to facilitate the integration of software components forming parts of software applications.

SUMMARY OF THE INVENTION

Particular and preferred aspects of the invention are set out in the accompanying independent and dependent claims. Combinations of features from the dependent claims may be combined with features of the independent claims as appropriate and not merely as explicitly set out in the claims.

An aspect of the invention provides an application framework for supporting at least one application that includes one or more application components. The framework comprises a directory providing a logical definition of the application. The logical definition is referenced by an application context identifier and provides logical definitions of the application components that make up the application. It further includes references to physical instances of the application components.

In this manner the framework is useable by a component of an application to reference a further component of the application with the same application context without needing to know about that further component. The use of a framework in accordance with the invention can provide the basis of an architecture that facilitates the building of componentized applications. It enables individual components to be linked to form an application without those components needing knowledge of context of their application. The context of the application will depend, for example, on the type of application. Thus, for example, the application context may be a word processing application, a spreadsheet application, a chemical database application, etc.

For example, in an embodiment of the invention, the directory can include an entry identified by the application context identifier, which entry references logical definitions of the application components for that application and physical references to physical instances of the application components. The reference to a physical instance of the application component can identify a physical location for the application component. The logical definitions of the application components and/or their location facilitate the independence and extensibility of an embodiment of the invention.

The logical definitions of the application can further identify information for mapping between a data format internal to the application component and a data format external to the application component. This means that format conversion information would not need to be held by the application component. Similarly, the logical definition of the application can identify a connection method to be used to connect to the application component. As a result of this, this information would not need to be held by the application component. These features help to avoid unnecessary replication of information and enhance the flexibility of the

invention. By holding such information in a directory, rather than in individual application components, the integrity of the system can be enhanced as less data needs to be changed when elements of a system are upgraded or moved. By collecting such information in the directory, management and maintenance of the information becomes much more straightforward and less prone to error.

In one embodiment of the invention, the directory is configured as a hierarchy of entries that define an order of operation of the application components. In the embodiment the directory is configured using the Lightweight Directory Access Protocol (LDAP).

In an embodiment of the invention communication between components via the framework is effected using, for example, a markup language, for example the eXtensible Mark-up Language (XML).

Another aspect of the invention provides a computer system comprising the application framework discussed above. The processing system could be a single processor with a memory in which the directory is configured. Alternatively, the system could comprise a plurality of computers connected via a network. In the latter case, multiple instances of the application framework could be provided in respective computers.

A further aspect of the invention provides an application component for such a system. The application component comprises a harness that contains an application component code segment for a predetermined application context. The harness forms an interface between the application component code segment and the outside of the application component. The harness is operable to apply a predetermined mapping for the application context to map between an external format for communication external to the application component and an internal format understandable to the application component code segment.

The use of the harness means that an interface can be provided between an application code segment and the outside world, minimizing any changes needed to an application code segment of an existing application for use with an embodiment of the invention. The use of the application context identifier provides a consistent link between the directory structure mentioned earlier and the application components.

The harness can be operable to extract an application context identifier for the application context from a received message and to obtain information for mapping between the external format and the internal format according to the application context.

The harness can be operable to identify and to obtain information for mapping between the internal format and the external format according to the application context for a message to be sent from the application component. It can further be operable to map data output from an application component code segment in the internal format to the external format and to add an application context identifier to the outgoing message. The application context identifier thus accompanies the message to the next recipient.

The harness can be operable to obtain the mapping information by accessing a directory external to the application component.

The harness could be operable to seek to obtain the mapping information for an application context from an internal cache using the application context identifier, and where the mapping information is not available and current, to obtain the mapping information from a directory external to the application component. This can reduce the number of directory look-ups needed, but does require the caching of information and the risk that locally held information may not be valid.

The application component code segment and the harness comprise computer code. This could be provided on a carrier medium, for example a storage or a transmission medium.

Yet a further aspect of the invention provides a method of supporting computer applications that include one or more application components, the method comprising:
defining an application framework for supporting the application, the framework comprising a directory providing a logical definition of the application;
referencing the logical definition using an application context identifier; and
providing logical definitions of the application components that make up the application to the application components including references to physical instances of the application components.

An aspect of the invention also provides a method of supporting computer applications that include one or more application components, the method comprising:
forming an application component in which the application component includes a harness that contains an application component code segment;
operating the harness as an interface between the application component code segment and an exterior of the application component; and
the harness mapping between an external format for communication between application components and an internal format understandable to the component.

An embodiment of the invention can thus facilitate the building of distributed, componentized applications. Individual components within the application do not need to have knowledge of the context of their application and the code in the component does not need to know who called it or who to call. The context is derived from the logical information, which could also be described as metadata, from a directory. As a result, the components are more easily re-usable, and more easily reconfigurable to respond to changing needs.

It can be seen that all of the various aspects of the invention mentioned above are linked by the use of the application context. This means that application components can be logically grouped even if they are distributed physically throughout a network. It further provides an efficient, reliable and low maintenance approach to supporting multiple applications over a non-homogenous network, that is readily adaptable and extensible as systems and applications develop over time.

Thus, an embodiment of the invention enables the flexible, adaptable, maintainable and scaleable applications to be developed that can use off-the-shelf and/or custom-developed software components. It effectively forms a glue layer that handles the interconnection of the code components, allowing application developers to concentrate on tackling the application problems and not have to worry about integrating the various parts of an application.

Development times can be reduced as a result of the invention reducing the difficulty of integrating software components. Maintenance problems caused by dependencies between different components can be reduced. Also, the improved approach to integration has the further advantage that resulting applications will be more supportable in the years after release. The invention is operable to enable the connection of software components on open platforms such as the Windows NT operating system, the Unix operating system, etc. and can enable the connection of components across platforms. It also allows the use of any of a wide variety of known protocols and protocols yet to be developed to be used for the interconnection of the application components, for example the protocols such as HTTP, IIOP, message queues, etc.

BRIEF INTRODUCTION TO THE DRAWINGS

Exemplary embodiments of the present invention will be described hereinafter, by way of example only, with reference to the accompanying drawings in which like reference signs relate to like elements and in which:

Figure 1 is a schematic representation of a computer workstation for an exemplary implementation of the invention;

Figure 2 is schematic block diagram illustrating an exemplary configuration of a computer workstation as shown in Figure 1;

Figure 3 is schematic representation of a network for a networked implementation of the invention;

Figure 4 is a conceptual representation of application components for an application context in an embodiment of the invention;

Figure 5 is a schematic representation of components related to the operation of a harness of the application component of Figure 4;

Figure 6 is flow diagram giving an overview of the operation of the application component of Figure 5;

Figure 7 illustrates an example of a directory structure;

Figures 8 to 10 are class diagrams illustrating classes of objects for an embodiment of the invention; and

Figures 11 – 15 are sequence diagrams illustrating in more detail aspects of the operation of the application component of Figure 4.

DESCRIPTION OF PARTICULAR EMBODIMENTS

Exemplary embodiments of the present invention are described in the following with reference to the accompanying drawings.

Figure 1 is a schematic representation of a computer workstation on which an exemplary embodiment of the invention can be implemented. As shown in Figure 1, a computer workstation 10 includes a system unit 12, user input devices, for example in the form of a keyboard 14 and a mouse 16, and a display 18. Removable media devices in the form, for example, of a floppy disk drive 20 and an optical and/or magneto-optical drive (e.g. a CD, a DVD ROM, a CDR drive) 22 can also be provided.

Figure 2 is schematic block diagram illustrating an exemplary configuration of a computer workstation 10 as shown in Figure 1.

As shown in Figure 2, the computer workstation 10 includes a bus 30 to which a number of units are connected. A microprocessor (CPU) 32 is connected to the bus 30. Main memory 34 for holding computer programs and data is also connected to the bus 30 and is accessible to the processor. A display adapter 36 connects the display 18 to the bus 30. A communications interface 38, for example a network interface and/or a telephonic interface such as a modem, ISDN or optical interface, enables the computer workstation 10 to be connected to other computers via, for example, an intranet or the Internet. An input device interface 42 connects one or more input devices, for example the keyboard 14 and the mouse 16, to the bus 30. A floppy drive interface 44 provides access to the floppy disk drive 20. An optical drive interface 46 provides access to the optical or magneto-optical drive 22. A storage interface 48 enables access to a hard disk 50. Further interfaces, not shown, for example for connection of a printer (not shown), may also be provided. Indeed, it will be appreciated that one or more of the components illustrated in Figure 2 may be omitted and/or additional components may be provided, as required for a particular implementation.

Although Figures 1 and 2 have illustrated a single computer on which an application of the present invention may be operable, more typically an embodiment of the invention will be operable on a distributed processing system, for example a distributed processing system such as that illustrated schematically in Figure 3. Figure 3 shows a network (for example, the Internet) 52 that provides connections between various stations 54, each of those stations communicating with each other via the network 52 by means of one or more protocols 56. The individual stations 54 may be stand-alone computers or, as illustrated at the top left of Figure 3, may in fact represent a network of computers

including a gateway 55 that provides a connection via an internal network 58 to a plurality of computers 59 or other devices. In such an environment, it is often the case that multiple protocols may be used at different parts of the system.

More particularly, different software components that may form a complex application may have been written using different programming languages and/or on different platforms and/or by different people at varying times. Accordingly, the exact format of data that any particular component expects to receive, and is able to transmit, may vary. Particularly where the communication between components is over a network as illustrated, for example, in Figure 3, this provides enormous complexity in order to provide integration of the various components of a complex application. The present invention is intended to address this.

To assist in an understanding of the following description, there follows a brief summary of some of the terms used herein.

An embodiment of the invention provides a "harness" around individual "application component code segments". The combination of an application component code segment and a harness is known as an "application component". An application component code segment is a segment of code that forms part of a user application. Examples of applications can be word processing applications, spreadsheet applications, chemical database applications, to name but a few. The harness buffers an individual application component code segment from the outside and takes care of identifying how to communicate with other application components and where to find them.

The harness includes only a small amount of code and is operable to reference a "directory" that holds or in turn references information needed by the application component code segments. The combination of the harness and the directory forms a "framework" for linking application component code segments.

An embodiment of the invention links respective elements by an "application context", that is information defining the application. The application context can link application components that together form, for example, a word processing application, a spreadsheet application, a chemical database application, and so on.

The application context is identified in the envelope of a "common messaging protocol" used between application components etc. and also forms entries in the directory, which is organised hierarchically in a preferred exemplary embodiment.

Figure 4 is a schematic representation of a number of application components 70 linked together within an application context, say as distributed components of a word processing application.

Figure 4 illustrates three application component code segments 64. It is assumed that each of these application component code segments provides different functions which, when combined together, provide the complete application for the application context.

It will be noted that each application component code segment 64 is provided with a respective harness 60 to form respective application components 70. As mentioned above, each harness 60 buffers its application component code segment 64 from needing to know about the "outside world" and thus forms an interface between that application component code segment 64 and the "outside world". Figure 4 illustrates connection components (hereinafter connectors) 62 within the harnesses 60. These connectors 62 provide messaging between application components 70 in accordance with a standardised protocol 66. In an embodiment of the invention a mark-up language, for example an eXtensible Markup Language (XML), is used as a standardised communication protocol 66 for communication between the connectors 62 of the respective application components 70.

Figure 5 is a schematic diagram illustrating an example of functional elements that are associated with the operation of the application component 70 in more detail. It should be understood that the functional elements illustrated in Figure 5 do not necessarily correspond to separate physical elements of program objects, but relate to separate functions that are performed. Examples of software objects used in a specific example of the invention will be described in more detail with reference to Figures 8 to 15.

Figure 5 illustrates an input connector (IC) 62.1 that receives an external message (in the present example an XML message).

A receiver (R) 74 (which can be implemented by an object called "receiver" described with reference to Figures 8 to 15) unwraps a received message and extracts an application context from a message envelope. The application context can be held as XML data for an XML "application context" tag. The application context is passed to a protocol interface (PI) 68 (which can be implemented by an object called "SoftBrix" described with reference to Figures 8 to 15) that derives information for the application context, either from a local cache (C) 76 (if such information for the application context is already held in the cache) or from an external directory 80.

In a preferred embodiment of the invention the external directory is implemented under the Lightweight Directory Access Protocol (LDAP), which provides a hierarchical directory. The application context forms an entry in the directory, and access to that entry provides access via the hierarchy to further entries that provide references to information needed by the harness 60. This information includes links to mapping files 82 that are supplied to the protocol interface 68 and are used by a mapper (M) 72 to map data from the received message to a format required by the application component code segment 64. For example, if the application component code segment is implemented in Java, XML data received in the XML message can be mapped into a set of Java objects in a form suitable for the task being undertaken by the application component code segment 64. The same principle holds for other programming environments such as C++, Visual Basic, etc.

Information received back from the directory 80 and the mapping files 82 is cached 76 by the harness 60.

When the application component code segment has finished its processing, information is output to the protocol interface 68, which then uses application context information and data stored in the cache 76 and/or retrieved by accessing the directory 80, to instantiate a data formatter 78 to map data output by the application component code segment 64 into XML. An envelope wrapper 84 (which can be implemented by an object called "sender" described with reference to Figures 8 to 15) then wraps the output data (which could be described as a letter) into an envelope. The envelope includes addressing information and an "application context" tag that defines the application context so that this information is available for a subsequent recipient.

One or more output connectors (OC) 62.2 receive the wrapped message for transmission and transmit this to the appropriate recipients identified in the envelope.

Figure 6 is a flow diagram giving an overview of the operation of the application component 70.

In step 100, an XML message is received by the first connector 62.1.

In step 102, the XML message is unwrapped by the protocol interface 68. The first protocol interface 68 is operable to unwrap the XML message by removing an "envelope" to expose a "letter" therein, the letter containing XML data. The envelope includes an application context identifier (e.g. a context type identifier as XML data to an XML "application context" tag).

In step 104, the protocol interface 68 uses the application context identifier as a key for accessing an internal cache 76 to find application context information referenced by the application context identifier. If the data is not held in the cache or has expired because it is older than a given age, the protocol interface 68 then uses the application context identifier to reference an entry in the external directory 80 labelled by the same application context identifier. The directory 80 includes references to further entries in the directory or to mapping files 82 external to the directory to provide information for mapping data in the received message to a format acceptable to the application component code segment 64 according to the application context concerned. The information retrieved from the directory is used to update the cache 76. Optionally, the mapping files could be held in the cache 76, or alternatively, the cache 76 could merely hold links to the mapping files 82.

In step 106, the mapper 72 is instantiated for mapping the XML data in the received XML message to the format acceptable to the application component code segment 64. It does this using mapping information for the application context from the mapping files 82 referenced by the directory or cache entries, as appropriate, for the application context.

In step 108, the application component code segment performs its processing tasks (the processing that the application code segment does), dependent on the application concerned. After carrying out the appropriate processing in step 108, the application code segment 64 then outputs data to the harness 60.

In step 110, a data mapper 78 maps the output data into XML using mapping information retrieved from the mapping files 82 using the entries in the cache 76 for the application context concerned. One or more mappings may be employed if plural messages are to be sent. The one or more XML messages are then wrapped by the envelope wrapper 84. The envelope wrapper includes in the envelope an "application context" tag with the application context as data therefor.

In step 112, one or more output connectors 62.2 transmit(s) the XML message(s) using a transmission protocol such as, for example, the well-known CORBA IIOP protocol.

With further reference to Figure 6, three types of application components can be defined.

There are application components that only send data to other application components. These provide the functionality of steps 106 to 112. This type of

application component typically gathers data, often as user input, and is analogous to a client application in a client/server environment.

There are application components that only receive data. These provide the functionality of steps 100 to 108. This type of application component typically consumes or stores data, possibly also passing back results, and is analogous to a server application in a client/server environment.

There are also application components that both send and receive data. These provide the functionality of steps 100 to 112.

Figure 7 is a schematic representation of an exemplary directory 80, which, in one embodiment, is implemented using the Lightweight Directory Access Protocol (LDAP). In other implementations, another directory protocol could be used.

For illustrative purposes, the directory is used to implement a simple echo name application that is represented by the XML code segment in Table 1. The echo name application forms a test application that echos a name.

TABLE 1

```
<?xml version="1.0"?>
<XOML EJB="">
    <OBJECT TYPE="SoftBrix.test.EchoDataMap">
            <MAP FIELD="echoData" ELEMENT="ECHODATA"/>
    </OBJECT>
</XOML>
```

The directory is arranged as a hierarchy starting with a root node (Directory Root) 120. Different classes of nodes depend from the root node 120. An application node 122 has an application instance in the form of the EchoNameApplication node 124. "EchoNameApplication" is the name of this application and this is an example of an "application context identifier". Everything below this node 124 in the hierarchy is within the application context. An application context may also include references to other application context identifiers. This allows an application to consist of many aggregated applications and/or components. This particular example application context 124 includes nodes 126 and 128 defining two application components represented by a SendName node 126 and an EchoName node 128.

The SendName application component is an example of a component that inputs data, possibly via a GUI client. Items in the directory below the SendName node

126 represent outputs from this component. Hence SendName has a single output group, labelled default, which itself contains a single output definition 132 identifying EchoName 128 as the application component to receive data sent by the SendName Application Component (as represented by the dashed line 134).

The EchoName component node 128 in turn defines a link, represented by the dashed line 136, to an EchoNameService physical component node 140, that is an instance of a physical component 138. The EchoNameService node 140, which defines a physical software component for implementing the application components 70 includes logical link 142 to a physical machine that runs the software.

As shown in Figure 7, a server node 144 includes a particular instance in the form of a node 146 labelled hlwwo223181 that identifies a physical server on which the EchoNameService physical component software is run and the protocol used to send the XML message to it (in this example, as shown in Figure 7, the IIOP protocol).

Also shown in Figure 7 is another branch 148 of the tree can include information relating to other resources such as queues, people, databases, etc.

As will be appreciated from Figure 7, the nodes in the directory tree provide logical definitions of individual components. This can also be described as metadata for those components. Although this one simple example is given for illustrative purposes, it will be appreciated by one skilled in the art how the principles employed in this example can be employed to define the relationships for one or more complicated examples.

Figure 8 is an overview of different classes used by an exemplary embodiment of the invention and the relationship between those classes. Moreover, this identifies various object classes identified in the subsequent sequence diagrams forming Figures 11-15.

Thus, a SoftBrix object class 170 provides a head class for the harness (this can be compared to the protocol interface 68 of Figure 5). Instances of a sender object class 172 are responsible for packaging messages for sending from an application component 70. Instances of an envelope object class 174 provide envelopes to contain a letter (data) for transmission. Instances of a receiver object class 190 provide for reception of data. Instances of a listening class 182 provide listeners for listening for component parts of an XML message. Instances of a mapper class 194 provide mappers for mapping. Instances of a data mapper 184 and object mapper 188 classes enable mapping of data and objects, respectively. The object mapper class 188 and data mapper class 184

play different roles in the mapping of marked up data, such as XML. Within a markup language, such as XML, data of relevance to an application component code segment 64 is identified by markup tags. The application component code segment 64 may have interest in both individually marked up data and groups of marked up data. It is the responsibility of the data mapper 184 to extract individual marked up data items. It is the responsibility of the object mapper 188 to identify groups of marked up data of relevance.

Also shown in Figure 8 are a connection pool object class 178, a package of connection object classes 62 (see Figure 9), a component entry properties class 176, and a package of directory object classes 80 (see Figure 10), instances of which will be described later with respect to Figures 11 to 15.

Figure 9 illustrates various sub-classes of the package of connection classes 62 for providing instances of connections under different protocols. Thus, Figure 9 illustrates an HTTP connection class 621, an IIOP connection class 622, an entity connection class 623, an SMTP connection class 624, an MQ connection class 625 and a file connection class 626. Further sub-classes can be defined to accommodate such further connection protocols as are required for a given implementation.

Figure 10 illustrates in more detail the sub-classes related to the directory class 800. This includes the directory itself 80, a component entry class 176, a component output class 177, a component input class 179, a component class 640, an e-mail class 181, an application class 175, a queue class 183 and a host server class 185.

In the following, the operation of an embodiment of the invention will be described in more detail with reference to Figures 11-15. In those Figures, various instances of the classes identified in Figures 8-10 are shown. The same reference signs as used for the classes in Figures 8-10 are used to identify instances of those classes in Figures 11-15. It should, however, be appreciated that in Figures 11-15 instances of objects, rather than the classes themselves, are identified.

Figure 11 is a sequence diagram illustrating in more detail the handling of data output by an application component code segment 64.

In step 150, the application component 64 simply sends its output to the protocol interface 68 (the SoftBrix object 170) of the harness 60. The application component code segment 64 does not need to know anything about any other components of the application (i.e., within the application context). The SoftBrix object 170 then outputs the data to a sender object 172.

In step 152, the sender 172 creates an envelope 174 that identifies the application by including an application context identifier as described earlier. The data output by the application component is then held in the "letter" within the envelope 174.

In step 154, the sender 172 then asks which outputs are needed for the current application and component by querying the directory 80. The information from the directory 80 is cached and a component entry 176 is generated.

In step 156, the sender 172 further identifies how to make a connection, with details of this being cached from the directory 80.

In step 158, a connection pool 178 queries the directory 80 to find physical connection parameters, these parameters also being cached, and this providing a component output 180.

In step 160, the sender 172 makes a connection, adds the "TO" field to the envelope 174 and identifies a subject for the connection.

In step 162, the sender 172 obtains data from the envelope 174 and then writes to the connection 62.

In step 164, this operation loops for the next output. In step 166, the next application component now has all the information needed to proceed, and in step 168 the sender causes closing of any connections made.

Within the framework, all inter-component communication is handled by the framework. The passing of information between application components is achieved in the present embodiment using XML. Event-based XML parsing using a conventional XML parser known as SAX is used to interpret received XML.

An embodiment of the invention has been implemented to utilize a third party environment for the generalized receipt of messages from a network. While this implementation utilizes the SUN Microsystems Enterprise Java Beans (EJB) environment, the invention is not limited to this environment. It is envisaged that custom and commercially available environments could be used. Examples of commercial environments further include the Unix INETD subsystem, the Microsoft COM+ subsystem, the IBM MQSeries subsystem as well as others.

Figure 12 illustrates how the framework sets up the context for receiving an application component 70. In particular, Figure 12 illustrates how an application

context tag identifying an application context is handled. This includes application identification, but could also be modified to include security verification. An event-based parser (SAX) reads a received XML message (as part of the connector 62) and handles it appropriately.

In particular, following receipt of an object at 200, in step 202 a receiver 190 recreates an envelope 174 and sets as an application name the received application context identifier. In step 204 the protocol interface 68 (SoftBrix object 170) then loads the application details. It initially looks in the cache 76 of the harness 60 for the appropriate application details, but refers to the external directory 80 when the cache details are absent or have expired.

Figure 13 illustrates how the framework handles a "TO" tag within a received XML message. The "TO" tag identifies a component that should be called within the context of the specified application.

Thus, when an XML message is received at 208, an event-based parser (SAX) reads a received XML message and handles it appropriately.

In step 210, a receiver 190 identifies which component should receive the data from the "TO" tag. Details about the "TO" component entry for the current application context are identified from the harness cache 76, or from the directory 80 as has been described earlier for other cache/directory accesses.

In step 212, an application identifier 175 identifies the component entry 176 for the current application context. In a manner described earlier for other cache/directory accesses, the application identifier 175 initially looks in the local harness cache 76, to find the appropriate information. However, if it is missing or has expired, the application identifier 175 then looks up the component entry in the directory 80.

In step 214, a reference to the requested component description is identified from the component entry in the cache or directory 80 as appropriate.

In step 216, the application component is located within the directory 80 and then the application component 70 is loaded.

In step 218, a location (URL) for a mapping file that defines how to map XML for the required component 64, is obtained.

Finally, in step 220, SAX listeners are set up for each of the items identified within the mapping file and are associated with a mapper object 194.

Figure 14 illustrates how data is received by an application component. The present embodiment is implemented in a Java environment and a conventional Java reflection interface is used to map from received XML format into a dynamically created Java object. At the start of an element, XML attributes are mapped only. Accordingly, Figure 14 illustrates how data is received by an application component code segment 64.

In step 230, a receiver 190 finds if there is a listener for the current tag.

If there is a listener, the listener 182 returns a data mapper or an object mapper object at step 232.

In step 234, a current tag is used to find a target object class for the listener. An object factory (a conventional Java function) is used to create an object of the target object class. This object is then set as the target to receive the incoming data.

In step 236, it is assumed that a new object within the XML data has been identified which needs to be mapped. The object now becomes active for incoming XML thanks to an object mapper 188.

In step 238, attributes are set for dynamically created objects and in step 240, Java reflection is used to invoke a method to set an attribute invocation method 186 to set the attribute.

Finally, in step 242, the object mapper 188 or data mapper 184 is pushed onto a stack. This allows items to be popped off in a hierarchical-matching XML tag order.

Figure 15 illustrates how the framework processes the end of an element within an incoming XML message. At this stage, any XML elements can be mapped to dynamically created objects.

In step 250, the listeners 182 for the current tag are removed.

In step 252, the previous tag is popped off the element stack 189.

In step 254, the XML element is mapped to the current object using the data mapper 184.

In step 256, Java reflection is used to set an attribute within the application component code segment 64 by means of the element invocation method 192.

Accordingly, there has been described an example of a flexible environment that enables application components to exist independently, yet still to work together to form an integrated application.

The described environment provided by an embodiment of the invention enables application component code segments (user code) to be created that do not have to know all about the overall environment and/or other application component code segments of the application. Each application type is assigned an application context, and communication takes place within the system using application context identifiers. The use of the application context identifiers, which are passed in messages between application components, enables a recipient application component to retrieve appropriate information for mapping the received data to be compatible with an application component code segment of the application component.

Each application component is formed from an application component code segment and a harness that forms a buffer between the code segment and the outside world. The application context identifiers are used by the harness to access appropriate information via an external directory. The use of the external directory means that the harness can be kept compact. A common communication protocol (for example a mark-up language such as XML) is used for communication between application components.

In connecting components, an embodiment of the invention provides an infrastructure that can enable the location and connection of components across an environment, which is potentially distributed, using a directory service (for example configured under LDAP). A protocol is provided that enables communication with components that can be user defined within the directory.

An embodiment of the invention can support a wide range of protocols including, for example, HTTP, CORBA/IIOP, MQSeries, SMTP, etc. Transformation of the external format (e.g., XML) to a format that an application component code segment can understand can be effected in a flexible and efficient manner through the storage of processing components accessible by referencing the directory service. With this configuration, the individual components can exist totally independently of one another. As effectively only one copy of the processing components is retained, this provides for easy maintenance and reliability of those components. It also provides for compact and efficient transformation engines in the application components.

Although, in the above description, reference is made to a single directory, it would also be possible for the directory to be mirrored at various locations throughout a network, in order that local access to the directory may be effected.

Typically, as well as mirroring the directory, the individual processing components would also be mirrored at the various locations. Such mirroring is well known in the network arts, and can be effected efficiently and at appropriate times to ensure that all mirrored versions are consistent with one another. Thus, although in such an environment there will be multiple copies of the processing components and the directory, they are effectively the same, as they are merely direct copies of one another.

As indicated above, the interconnection of the transformation engines and the connectors is specified using data accessible via the directory. The directory specifies where a component is found, what data should be sent to it, and what connection method should be used to connect to it. Providing such information centrally in the directory means that it is quick and easy to change the connection details. For example, the system can be configured to provide a form for effecting this. The application code within the various transformation engines does not need to know who is calling it or who to call next, this information being automatically derived from the directory service.

The invention may be implemented using any appropriate programming language. One embodiment of the invention is implemented using Enterprise Java Beans. However, other programming languages could be used. Also, software that is wrapped in a Java Bean can be in many different languages, including, for example, Javacode C/C++ code, Microsoft COM/ActiveX components, etc. By using connection methods such as SOAP, HTTP and XML, an application of the invention can be effectively neutral to a hardware platform, operating system, etc. Although specific references to programming languages and platforms have been made, the present invention is not limited thereto and could be implemented in a wide variety of languages for a wide variety of platforms. Computer code for implementing the invention can be provided on a carrier medium, for example a storage medium (e.g., solid state memory, a magnetic, optical or magneto-optical disk or tape, etc.), or a transmission medium (e.g., a wired or wireless medium such a telephone line, broadcast waveform, communication channel etc.).

Although particular embodiments of the invention have been described, it will be appreciated that many modifications/additions and/or substitutions may be made within the spirit and scope of the invention as defined in the appended claims.

CLAIMS

1. An application framework for supporting at least one application that includes one or more application components, the framework comprising a directory providing a logical definition of the application, the logical definition being referenced by an application context identifier and providing logical definitions of the application components that make up the application and references to physical instances of the application components.

2. The application framework of claim 1, wherein the directory comprises an entry identified by the application context identifier, which entry references logical definitions of the application components for that application and physical references to physical instances of the application components.

3. The application framework of claim 2, wherein said reference to a physical instance of the application component identifies a physical location for the application component.

4. The application framework of claim 2 or claim 3, wherein the logical definition of the application identifies information for mapping between a data format internal to the application component and a data format external to the application component.

5. The application framework of any of claims 2 to 4, wherein the logical definition of the application identifies a connection method to be used to connect to the application component.

6. The application framework of any preceding claim, wherein the directory is configured using LDAP.

7. The application framework of any preceding claim, wherein communication between application components is effected using a markup language.

8. The application framework of any preceding claim, further comprising at least one application component, wherein the application component comprises a harness that contains an application component code segment, the harness forming an interface between the application component code segment and an exterior of the application component.

9. The application framework of claim 8, wherein the harness is operable to map between an external format for communication between application components and an internal format understandable to the component.

10. The application framework of claim 9, wherein the harness is operable to extract an application context identifier from a received message and to obtain information for mapping between the external format and the internal format according to the application context.

11. The application framework of claim 9 or claim 10, wherein the harness is operable to identify and to obtain information for mapping between the internal format and the external format according to the application context for a message to be sent from the application component, to map data output from an application component code segment in the internal format to the external format and to add an application context identifier to the outgoing message.

12. The application framework of claim 10 or claim 11, wherein the harness is operable to seek to obtain the mapping information for an application context from an internal cache using the application context identifier, and where the mapping information is not available and current, to obtain the mapping information from the external directory.

13. A computer system comprising the application framework of any preceding claim.

14. The system of claim 13, comprising a processor and memory.

15. The system of claim 13 or claim 14, comprising a plurality of computers connected via a network.

16. The system of claim 15, comprising multiple instances of the application framework.

17. An application component for the system of any of claims 13 to 16, the application component comprising a harness that contains an application component code segment for a predetermined application context, the harness forming an interface between the application component code segment and an exterior of the application component, the harness being operable to apply a predetermined mapping for the application context to map between an external format for communication external to the application component and an internal format understandable to the application component code segment.

18. The application component of claim 17, wherein the harness is operable to extract an application context identifier identifying the application context from a received message and to obtain information for mapping between the external format and the internal format according to the application context.

19. The application component of claim 17 or claim 18, wherein the harness is operable to identify and to obtain information for mapping between the internal format and the external format according to the application context for a message to be sent from the application component, to map data output from an application component code segment in the internal format to the external format and to add an application context identifier to the outgoing message.

20. The application component of claim 18 or claim 19, wherein the harness is operable to obtain the mapping information by accessing a directory external to the application component.

21. The application component of claim 18 or claim 19, wherein the harness is operable to seek to obtain the mapping information for an application context from an internal cache using the application context identifier, and where the mapping information is not available and current, to obtain the mapping information by accessing a directory external to the application component.

22. The application component of any of claims 17 to 21, wherein the application component code segment and the harness comprise computer code.

23. A method of supporting computer applications that include one or more application components, the method comprising:

- defining an application framework for supporting the application, the framework comprising a directory providing a logical definition of the application;
- referencing the logical definition using an application context identifier; and
- providing logical definitions of the application components that make up the application to the application components including references to physical instances of the application components.

24. The method of claim 23, wherein providing of logical definitions includes providing a mapping between a data format internal to the application component and a data format external to the application component.

25. The method of claims 23 or claim 24, wherein communication between application component is effected using a markup language.

26. A method of supporting computer applications that include one or more application components, the method comprising:

- forming an application component in which the application component includes a harness that contains an application component code segment;
- operating the harness as an interface between the application component code segment and an exterior of the application component; and

- the harness mapping between an external format for communication between application components and an internal format understandable to the component.

27. The method of claim 26, wherein the harness extracts an application context identifier from a received message and obtains information for mapping between the external format and the internal format according to the application context.

28. The method of claim 26 or claim 27, wherein the harness identifies and obtains information for mapping between the internal format and the external format according to the application context for a message to be sent from the application component, maps data output from an application component code segment in the internal format to the external format and adds an application context identifier to the outgoing message.

29. The method of claim 27 or claim 28, wherein the harness seeks to obtain the mapping information for an application context by accessing a directory external to the application component.

30. The method of claim 27 or claim 28, wherein the harness seeks to obtain the mapping information for an application context from an internal cache using the application context identifier, and where the mapping information is not available and current, obtains the mapping information by accessing a directory external to the application component.

31. An application framework substantially as hereinbefore described with reference to the accompanying drawings.

32. A computer system substantially as hereinbefore described with reference to the accompanying drawings.

33. An application component substantially as hereinbefore described with reference to the accompanying drawings.

34. A method of supporting computer applications substantially as hereinbefore described with reference to the accompanying drawings.

Fig.1.

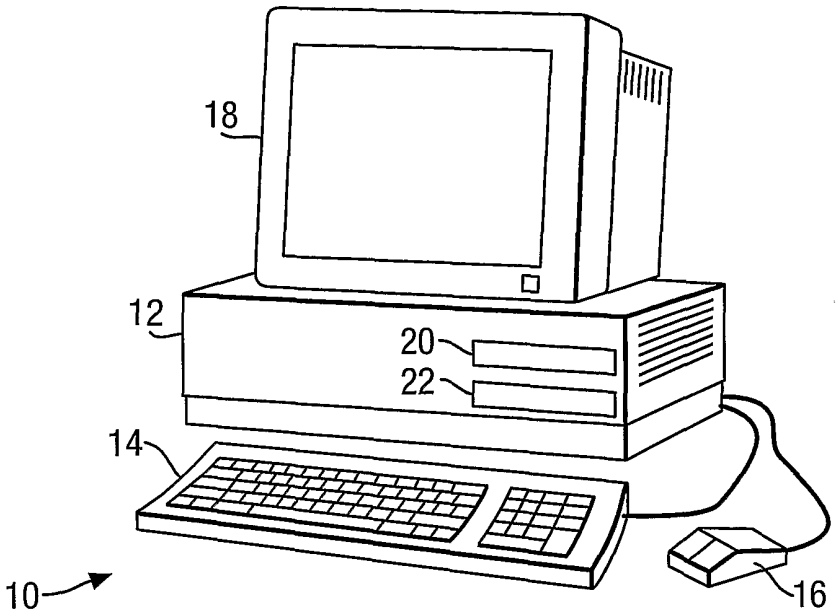


Fig.2.

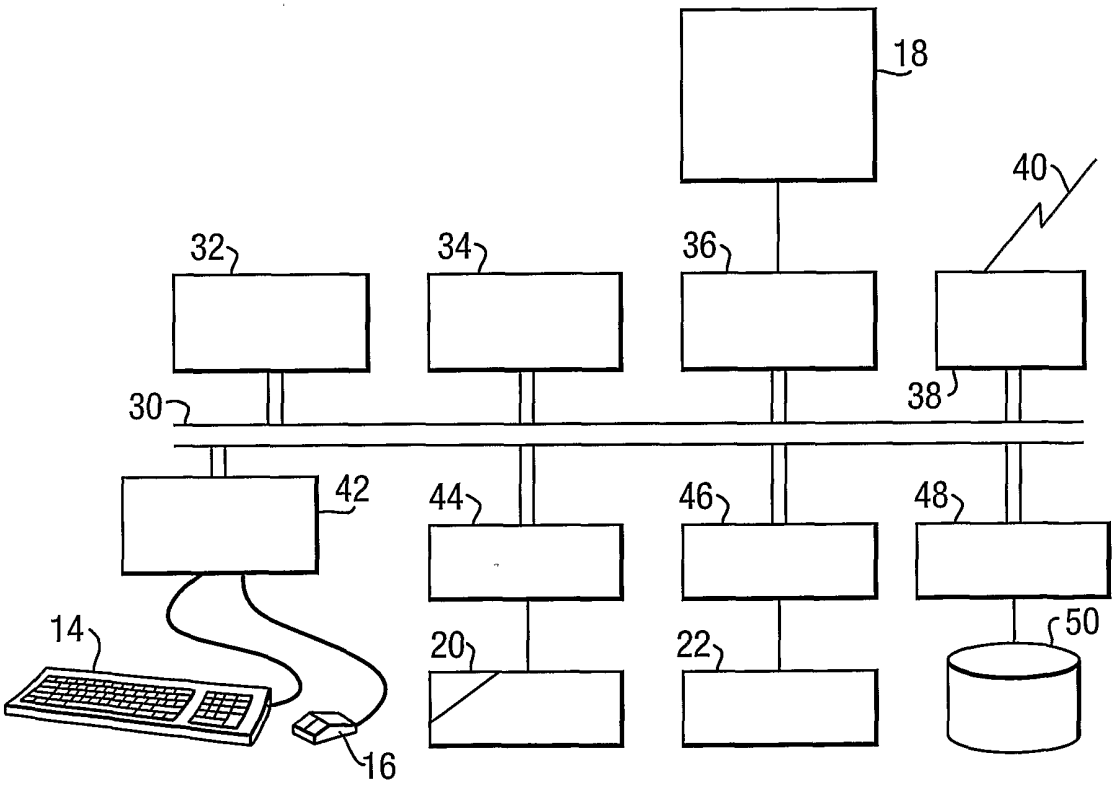


Fig.3.

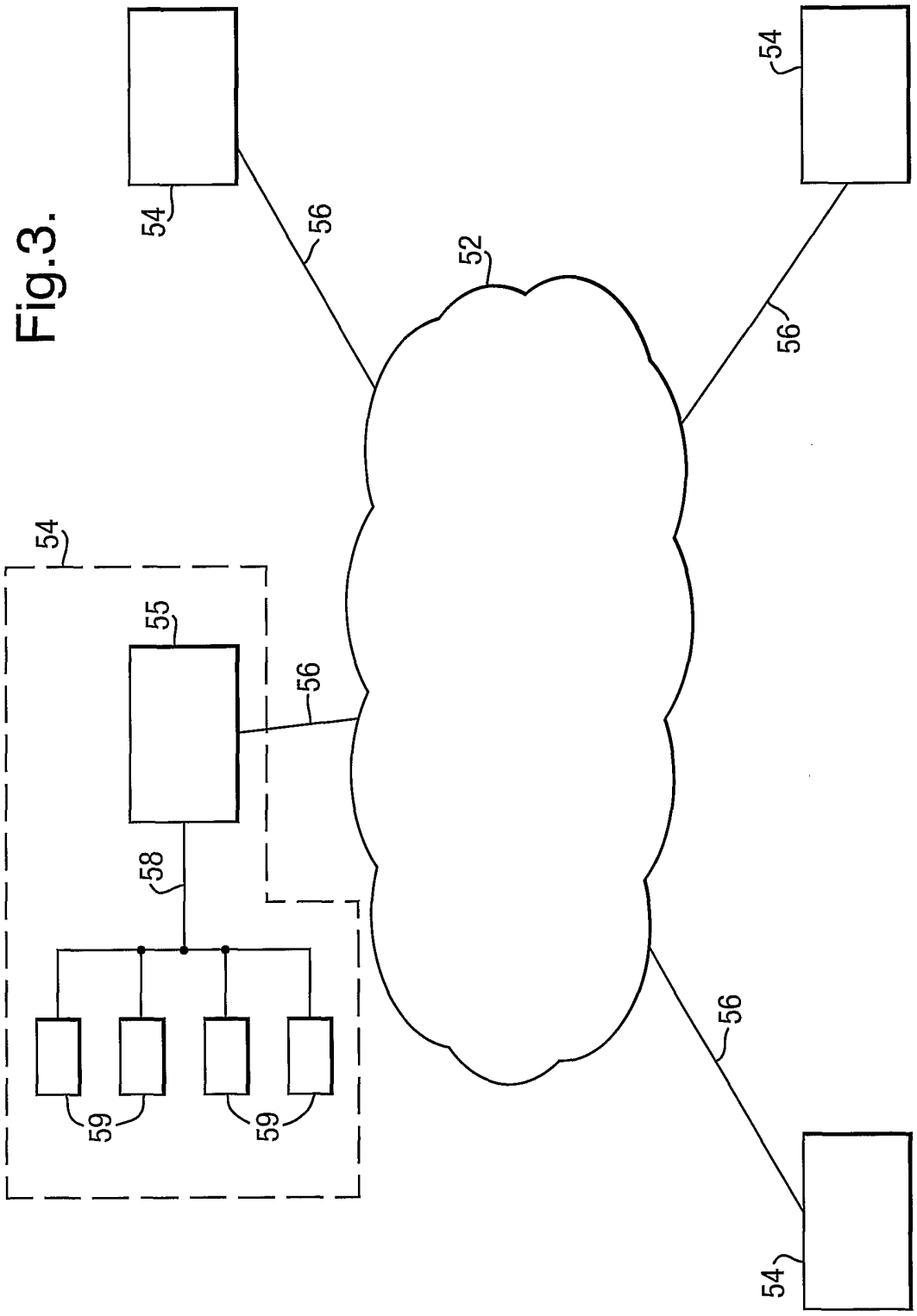


Fig.4.

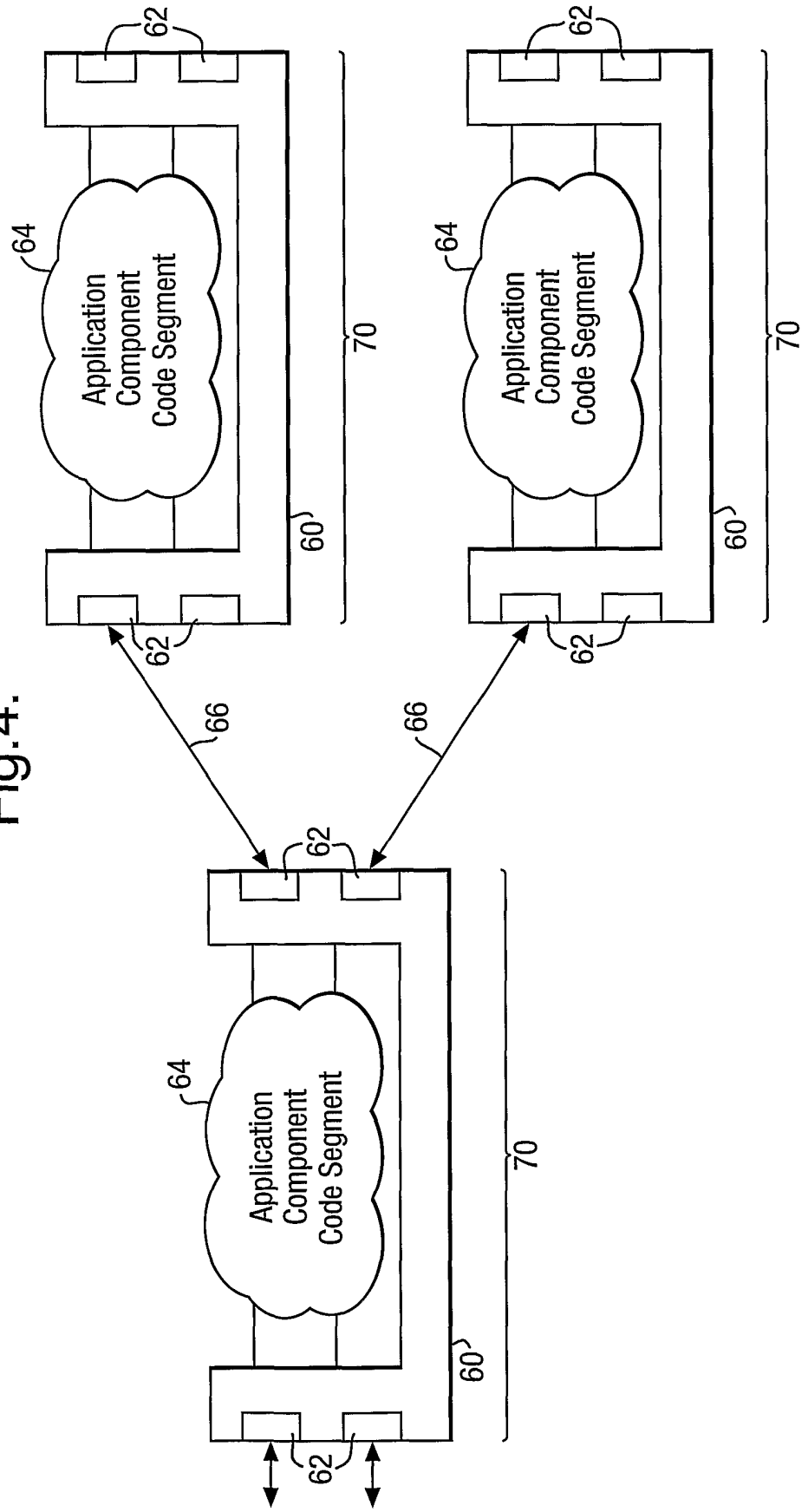


Fig.5.

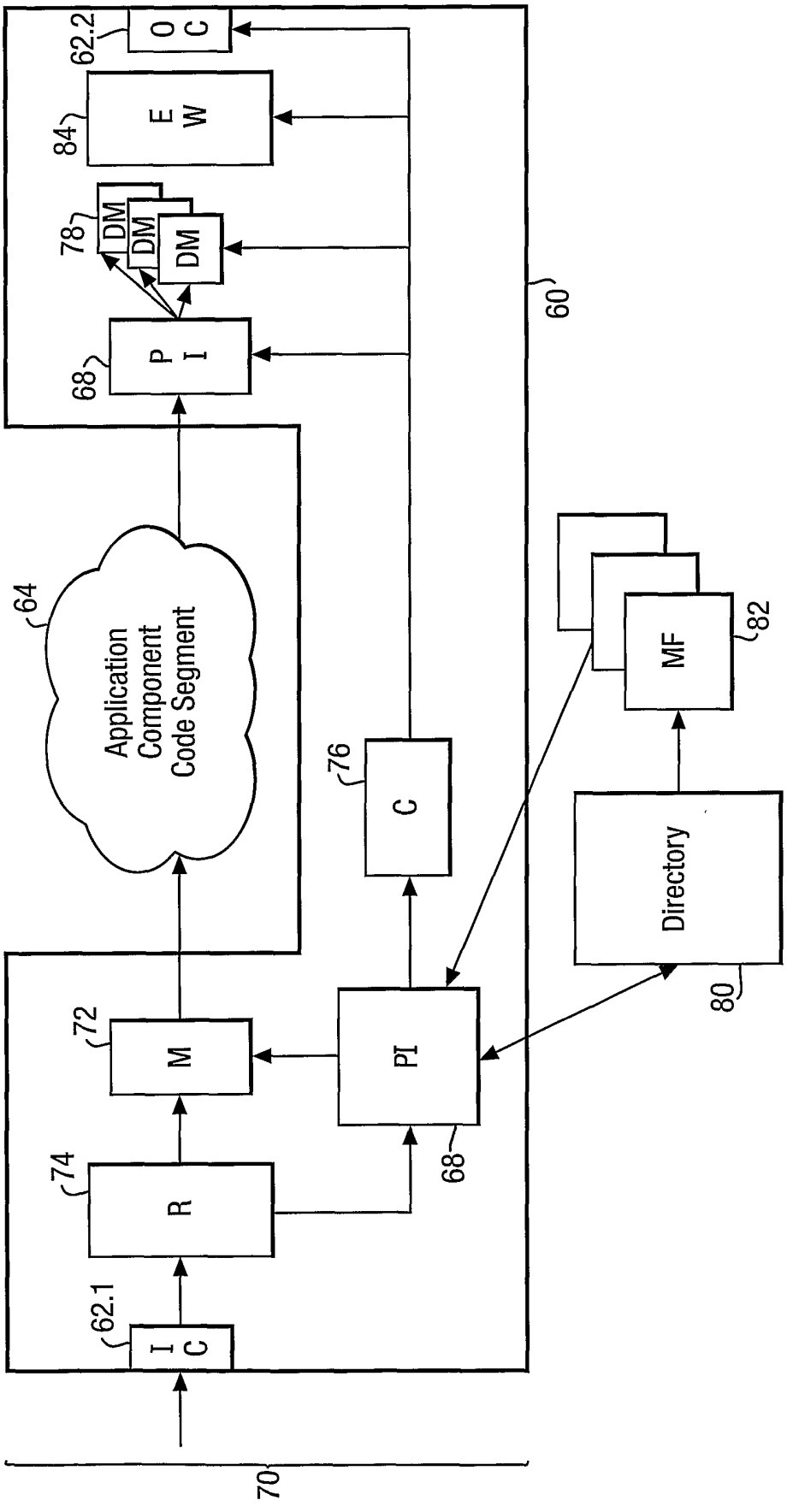

62.2
O C
84
E W
78
DM
DM
DM
68
P I
60
64
Application Component Code Segment
76
C
MF
82
Directory
80
72
M
PI
74
R
68
62.1
I C
70

Fig.6.

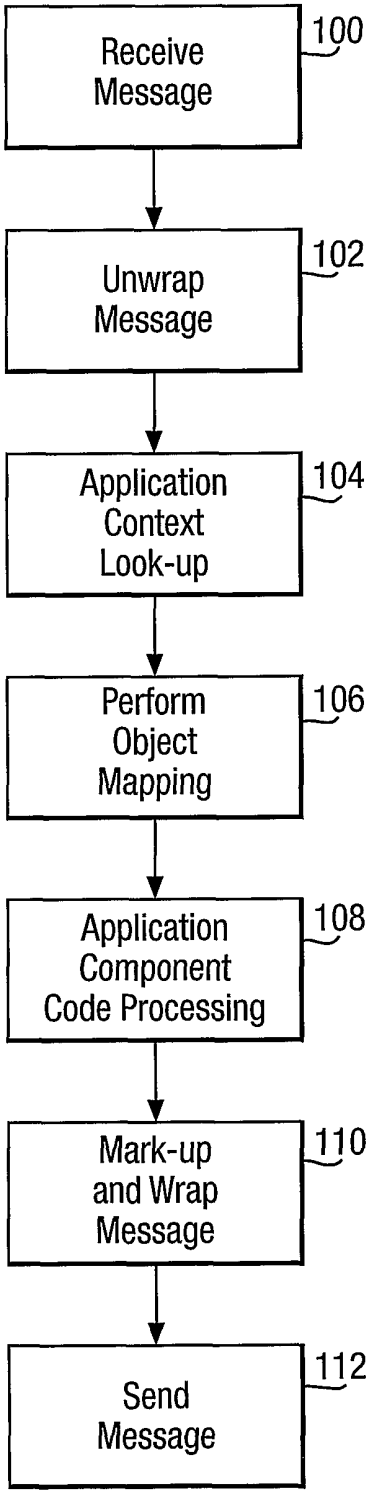

Receive Message
100
Unwrap Message
102
Application Context Look-up
104
Perform Object Mapping
106
Application Component Code Processing
108
Mark-up and Wrap Message
110
Send Message
112

Fig.7.

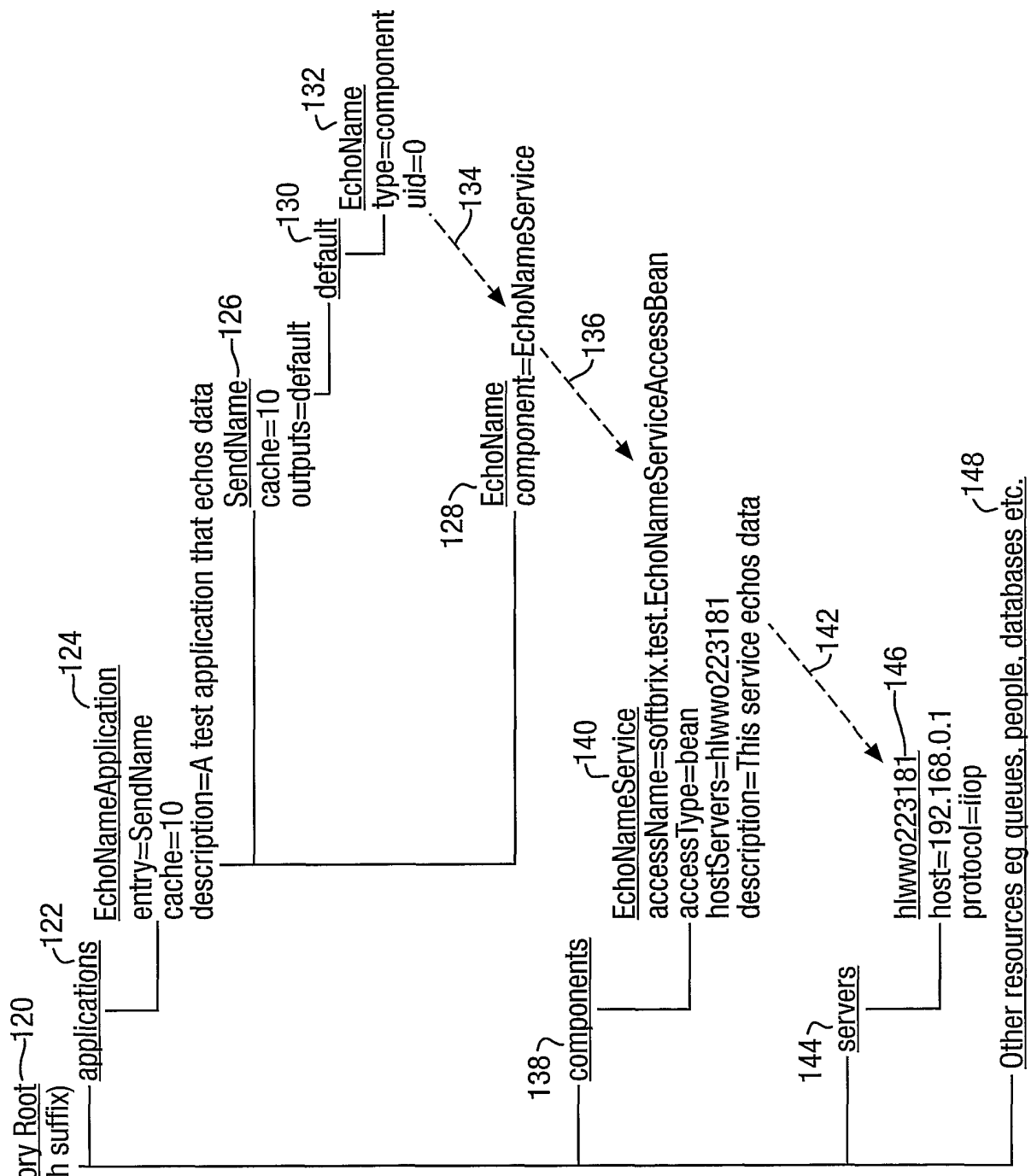

Directory Root
(search suffix)
120
applications
122
EchoNameApplication
124
entry=SendName
cache=10
description=A test application that echos data
SendName
126
cache=10
outputs=default
default
130
EchoName
132
type=component
uid=0
134
EchoName
128
component=EchoNameService
136
components
138
EchoNameService
140
accessName=softbrix.test.EchoNameServiceAccessBean
accessType=bean
hostServers=hlwwo223181
description=This service echos data
142
servers
144
hlwwo223181
146
host=192.168.0.1
protocol=iiop
Other resources eg queues, people, databases etc.
148

Fig.8.

Fig.8.

Fig.8.	Fig.8 (cont i).	Fig.8 (cont ii).

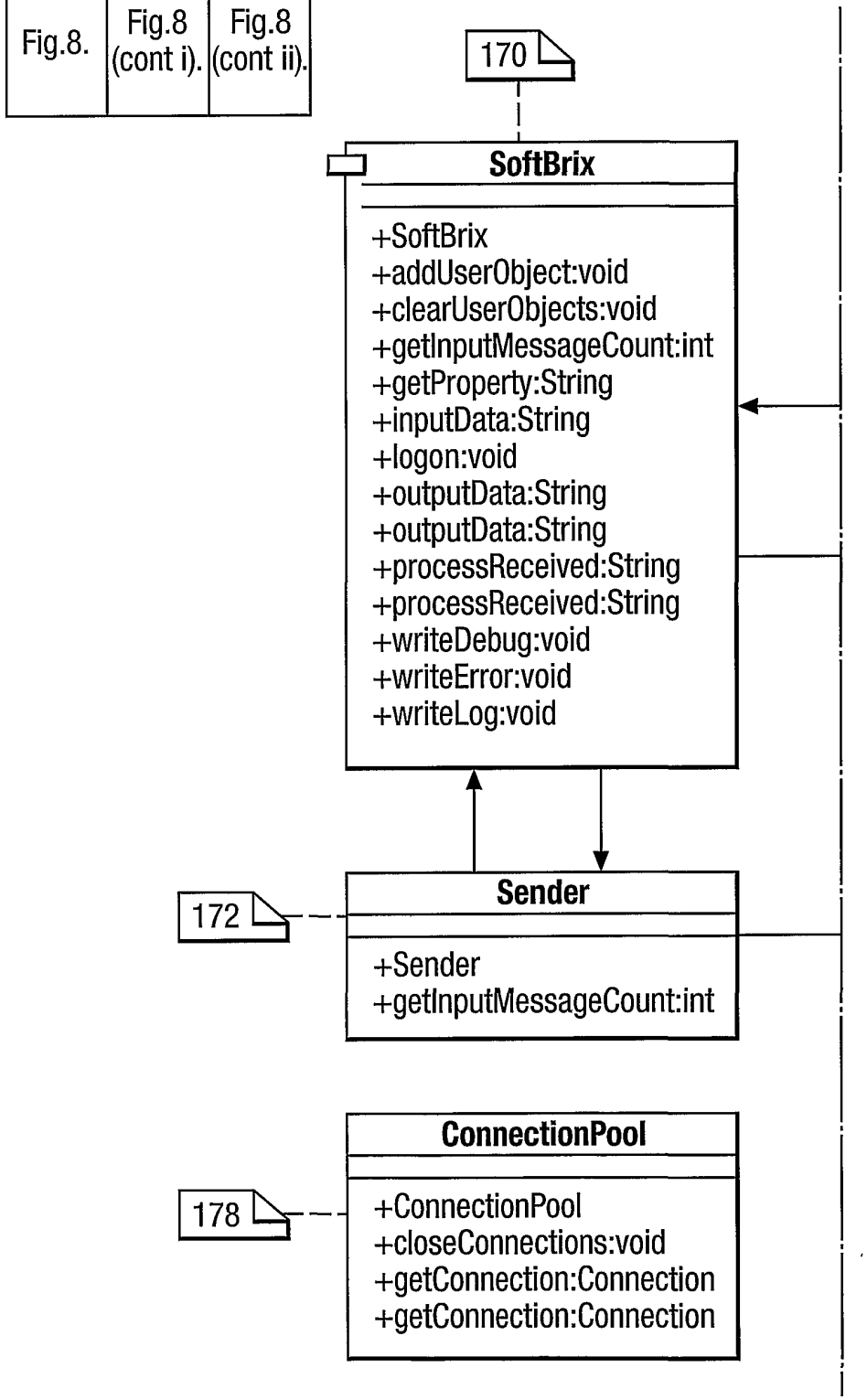


Fig.8 (cont i).

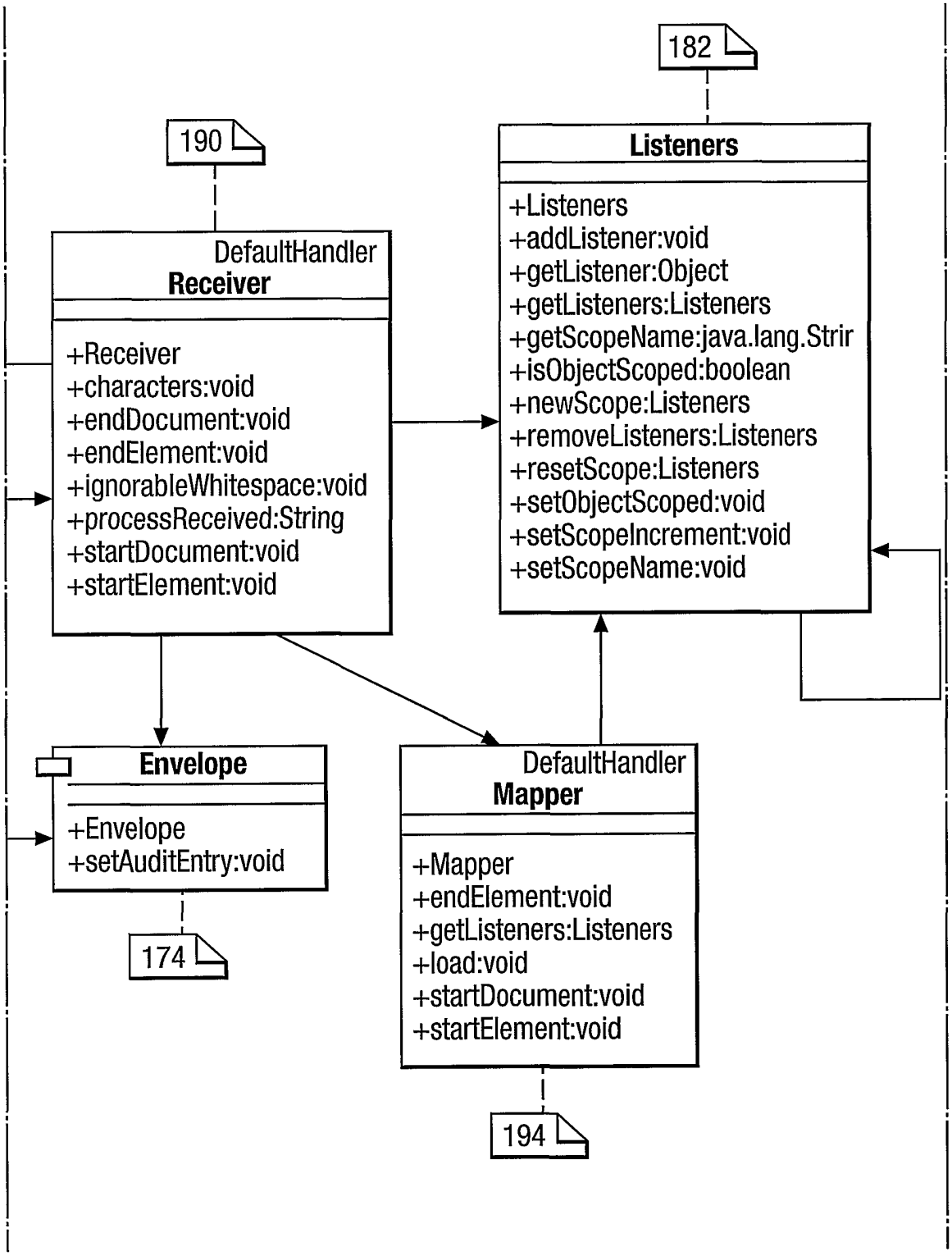
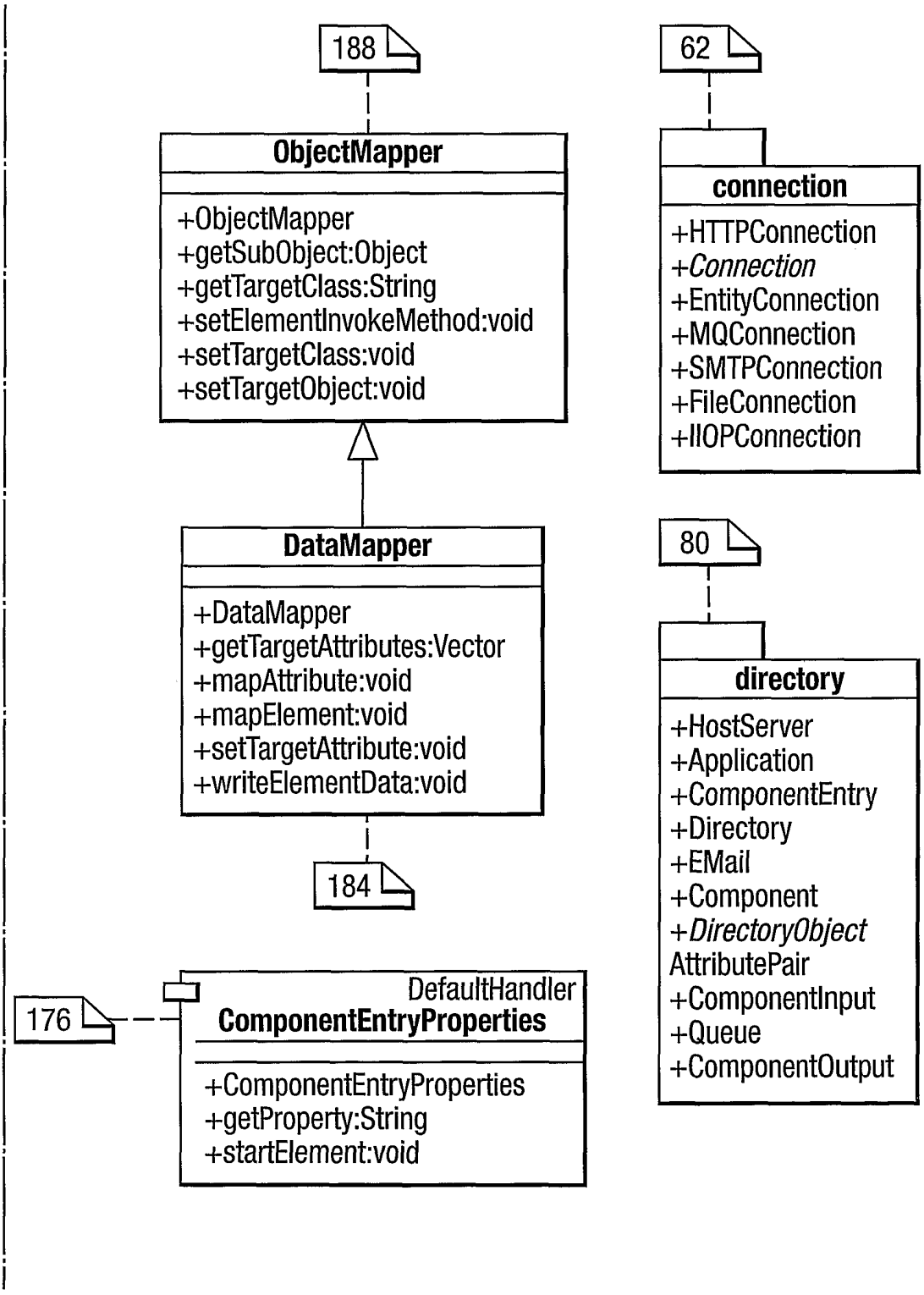

182
190
Listeners
+Listeners
+addListener:void
+getListener:Object
+getListeners:Listeners
+getScopeName:java.lang.Strir
+isObjectScoped:boolean
+newScope:Listeners
+removeListeners:Listeners
+resetScope:Listeners
+setObjectScoped:void
+setScopeIncrement:void
+setScopeName:void
DefaultHandler
Receiver
+Receiver
+characters:void
+endDocument:void
+endElement:void
+ignorableWhitespace:void
+processReceived:String
+startDocument:void
+startElement:void
Envelope
+Envelope
+setAuditEntry:void
174
DefaultHandler
Mapper
+Mapper
+endElement:void
+getListeners:Listeners
+load:void
+startDocument:void
+startElement:void
194

Fig.8 (cont ii).


188
ObjectMapper
+ObjectMapper
+getSubObject:Object
+getTargetClass:String
+setElementInvokeMethod:void
+setTargetClass:void
+setTargetObject:void
DataMapper
+DataMapper
+getTargetAttributes:Vector
+mapAttribute:void
+mapElement:void
+setTargetAttribute:void
+writeElementData:void
184
62
connection
+HTTPConnection
+Connection
+EntityConnection
+MQConnection
+SMTPConnection
+FileConnection
+IIOPConnection
80
directory
+HostServer
+Application
+ComponentEntry
+Directory
+EMail
+Component
+DirectoryObject
AttributePair
+ComponentInput
+Queue
+ComponentOutput
176
DefaultHandler
ComponentEntryProperties
+ComponentEntryProperties
+getProperty:String
+startElement:void

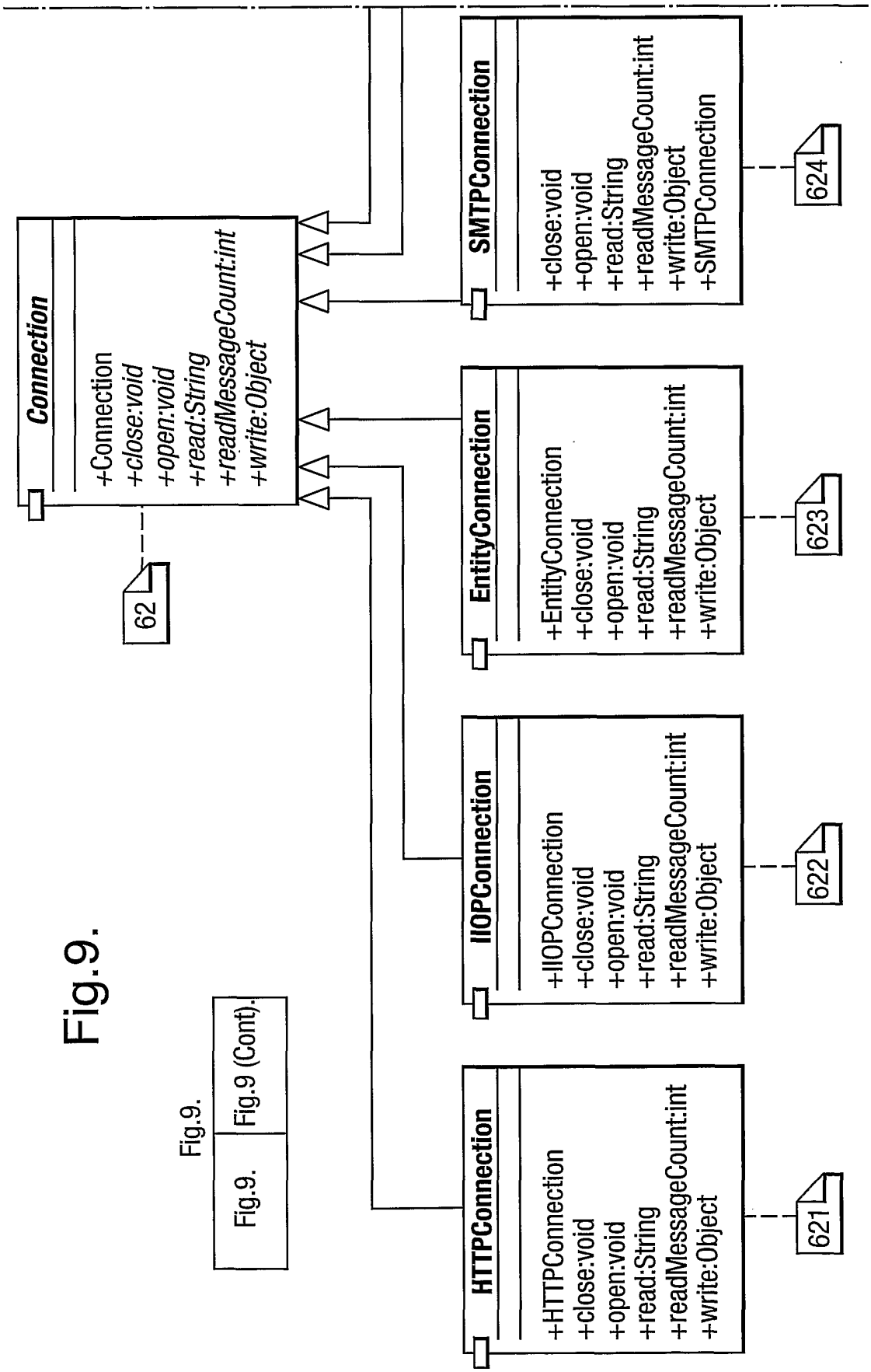


Fig.9.

Fig.9. | Fig.9 (Cont).

Fig.9 (Cont).

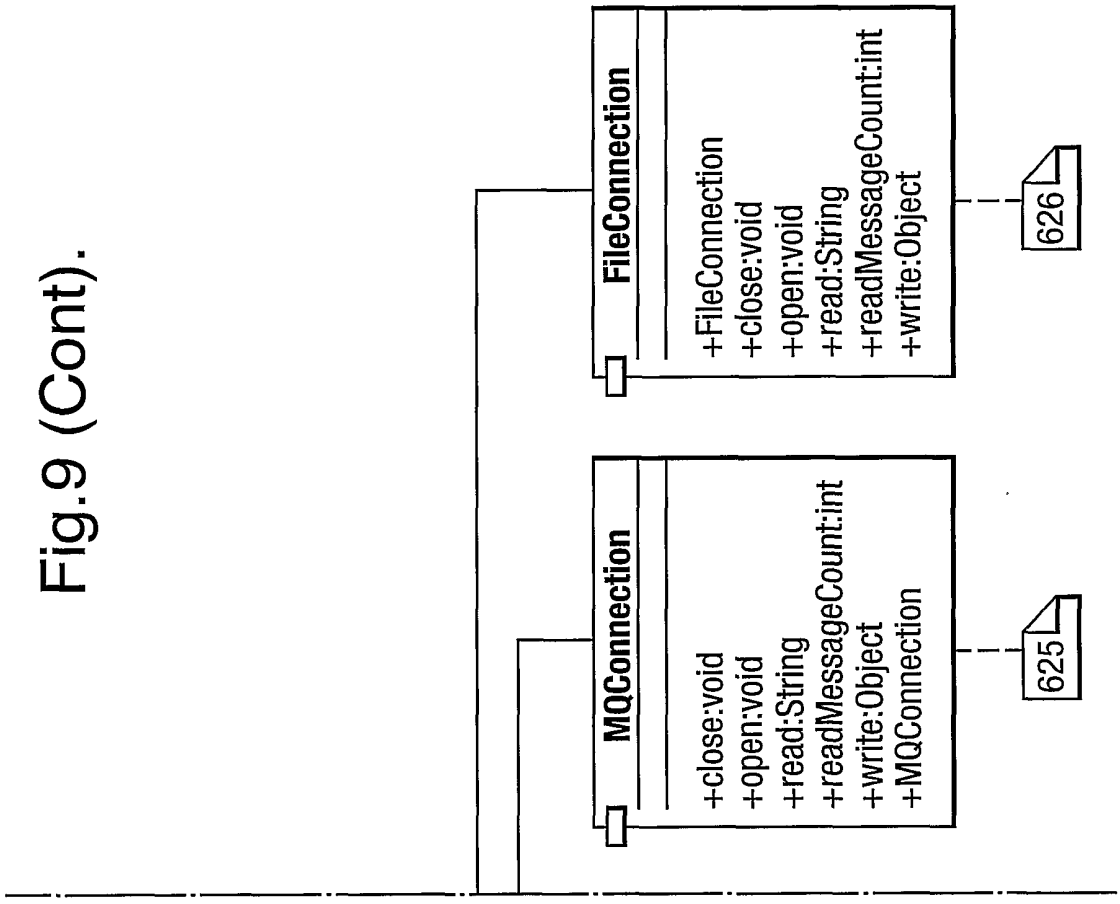

FileConnection
+FileConnection
+close:void
+open:void
+read:String
+readMessageCount:int
+write:Object
626
MQConnection
+close:void
+open:void
+read:String
+readMessageCount:int
+write:Object
+MQConnection
625

Fig.10.

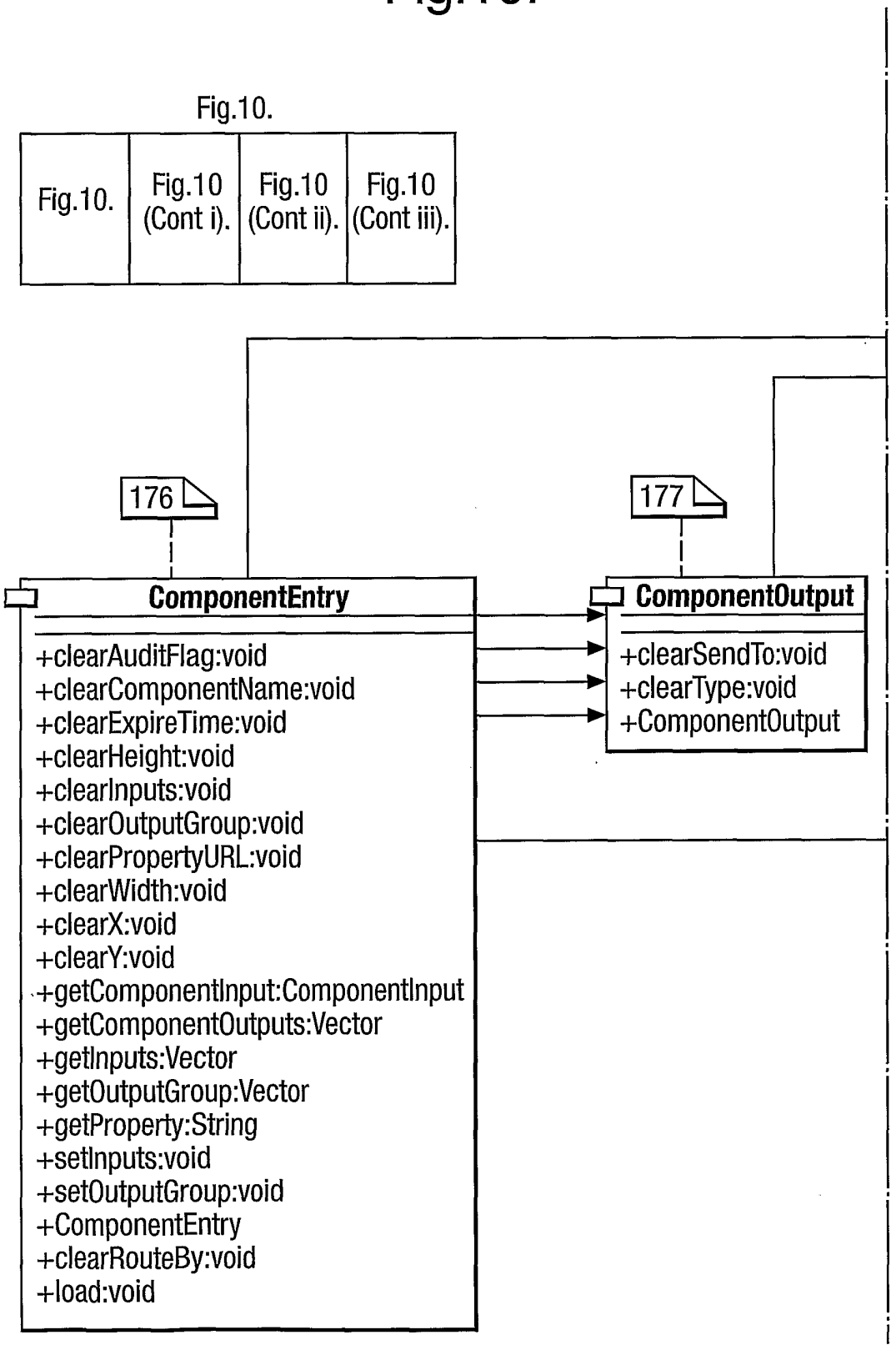


Fig.10.
Fig.10.
Fig.10 (Cont i).
Fig.10 (Cont ii).
Fig.10 (Cont iii).
176
177
ComponentEntry
+clearAuditFlag:void
+clearComponentName:void
+clearExpireTime:void
+clearHeight:void
+clearInputs:void
+clearOutputGroup:void
+clearPropertyURL:void
+clearWidth:void
+clearX:void
+clearY:void
+getComponentInput:ComponentInput
+getComponentOutputs:Vector
+getInputs:Vector
+getOutputGroup:Vector
+getProperty:String
+setInputs:void
+setOutputGroup:void
+ComponentEntry
+clearRouteBy:void
+load:void
ComponentOutput
+clearSendTo:void
+clearType:void
+ComponentOutput

Fig.10 (Cont i).

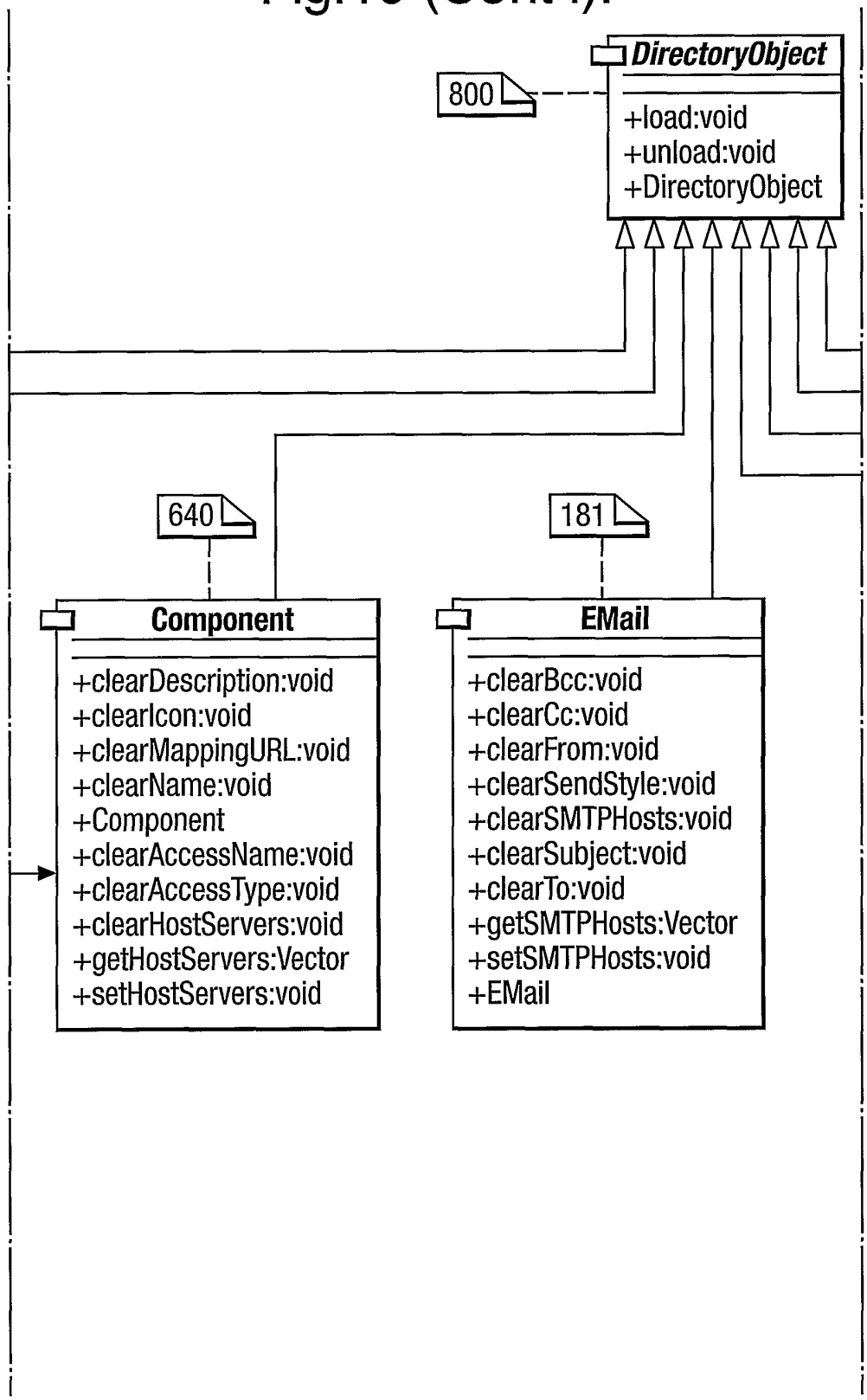

DirectoryObject
800
+load:void
+unload:void
+DirectoryObject
640
181
Component
+clearDescription:void
+clearIcon:void
+clearMappingURL:void
+clearName:void
+Component
+clearAccessName:void
+clearAccessType:void
+clearHostServers:void
+getHostServers:Vector
+setHostServers:void
EMail
+clearBcc:void
+clearCc:void
+clearFrom:void
+clearSendStyle:void
+clearSMTPHosts:void
+clearSubject:void
+clearTo:void
+getSMTPHosts:Vector
+setSMTPHosts:void
+EMail

Fig.10 (Cont ii).

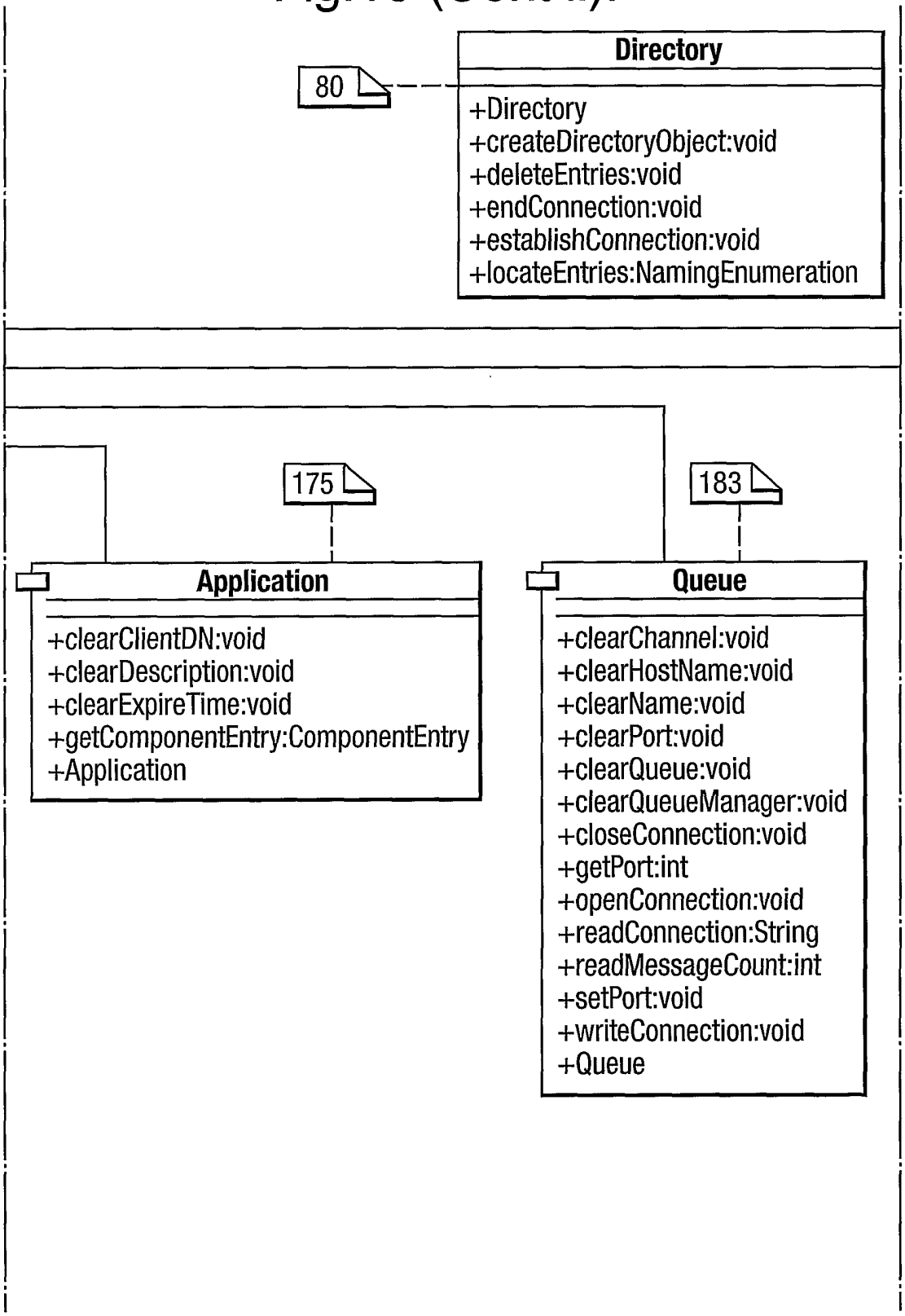


Fig.10 (Cont iii).

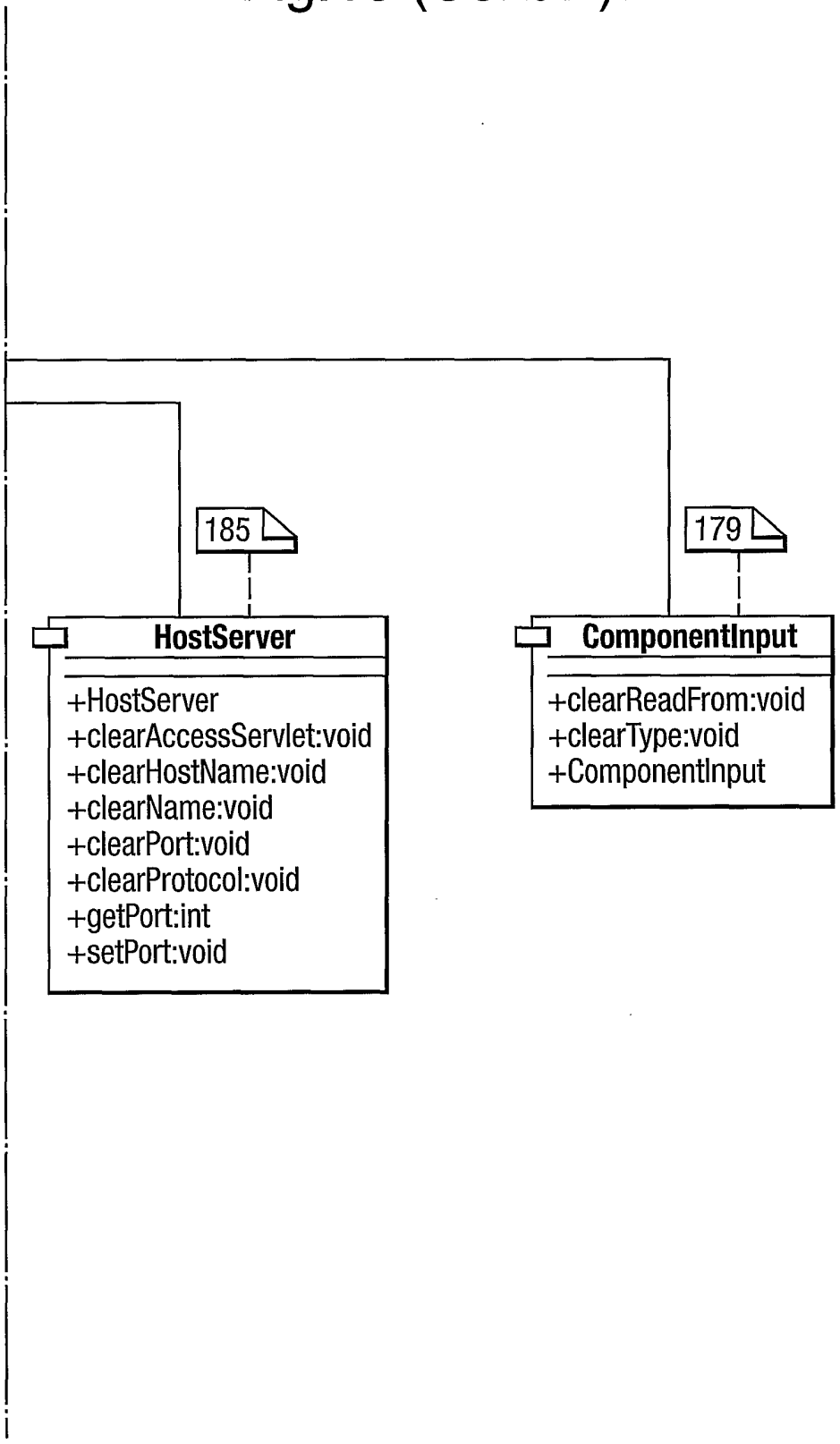


Fig.11.

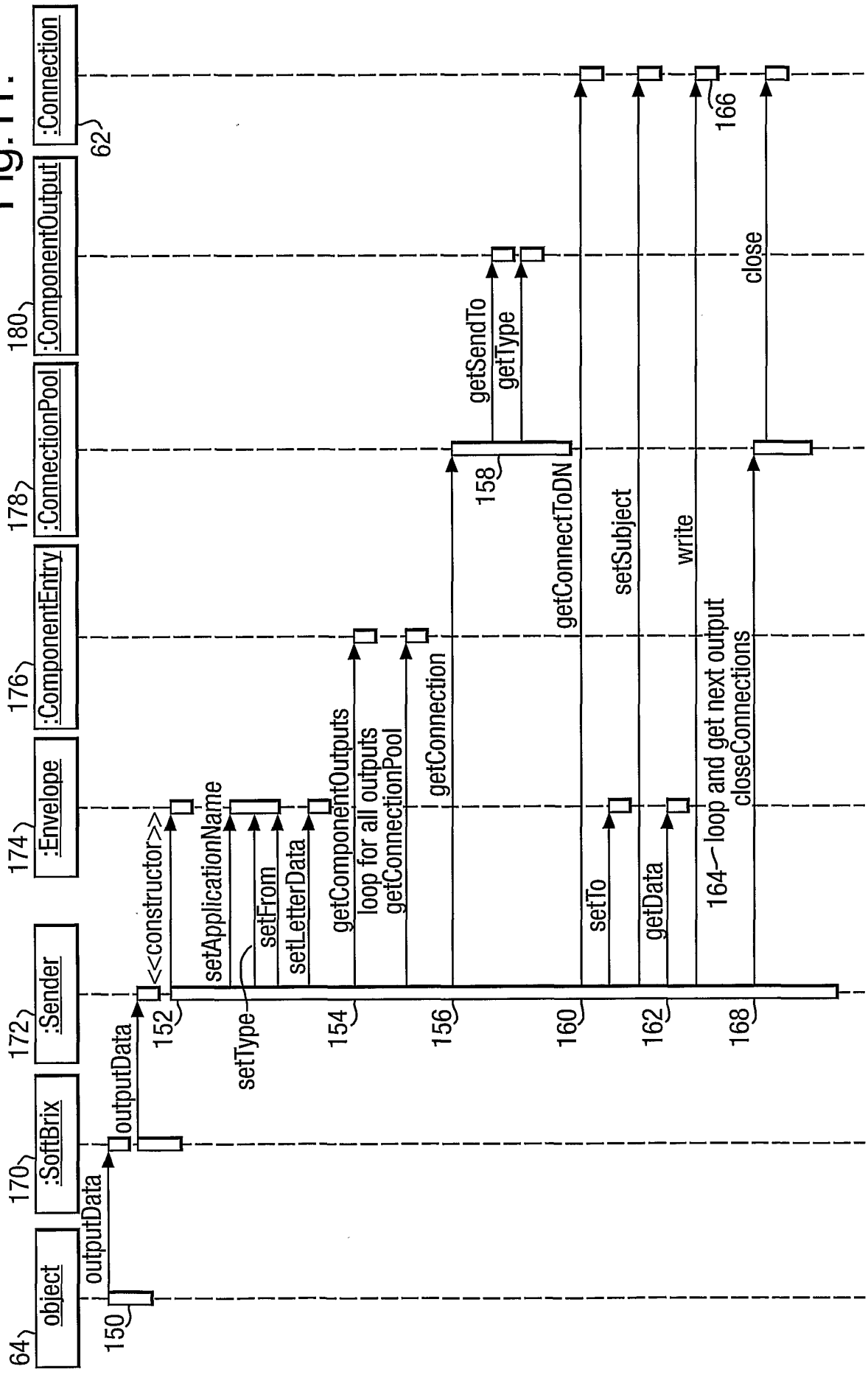
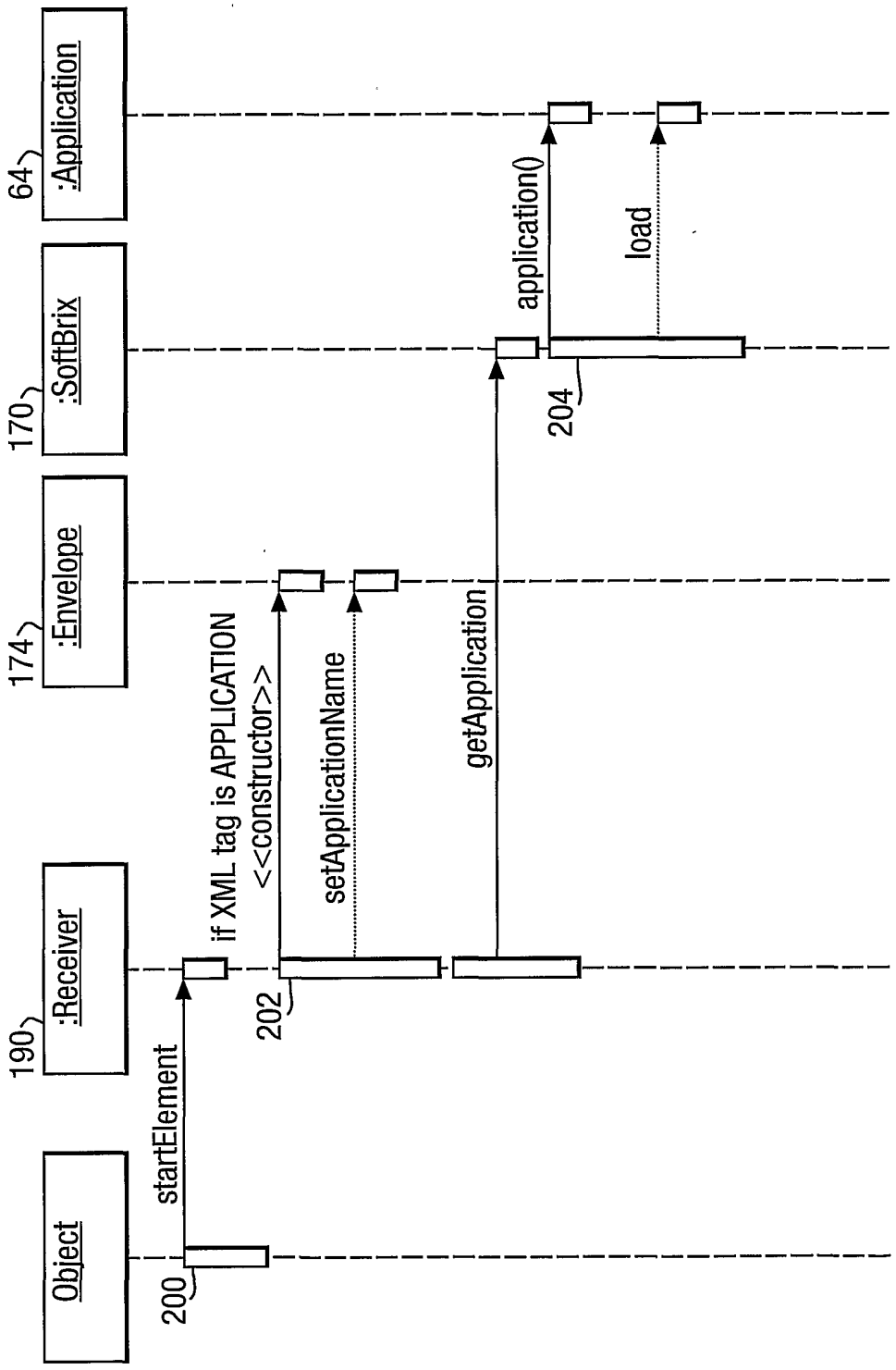

64
object
150
outputData
170
:SoftBrix
outputData
172
:Sender
152
<<constructor>>
174
:Envelope
setApplicationName
setType
setFrom
setLetterData
176
:ComponentEntry
154
getComponentOutputs
loop for all outputs
getConnectionPool
156
getConnection
178
:ConnectionPool
158
180
:ComponentOutput
getSendTo
getType
62
:Connection
160
getConnectToDN
setTo
162
setSubject
getData
166
write
164
loop and get next output
closeConnections
168
close

Fig.12.


:Application
64
:SoftBrix
170
:Envelope
174
:Receiver
190
Object
startElement
200
if XML tag is APPLICATION
<<constructor>>
setApplicationName
202
getApplication
application()
load
204

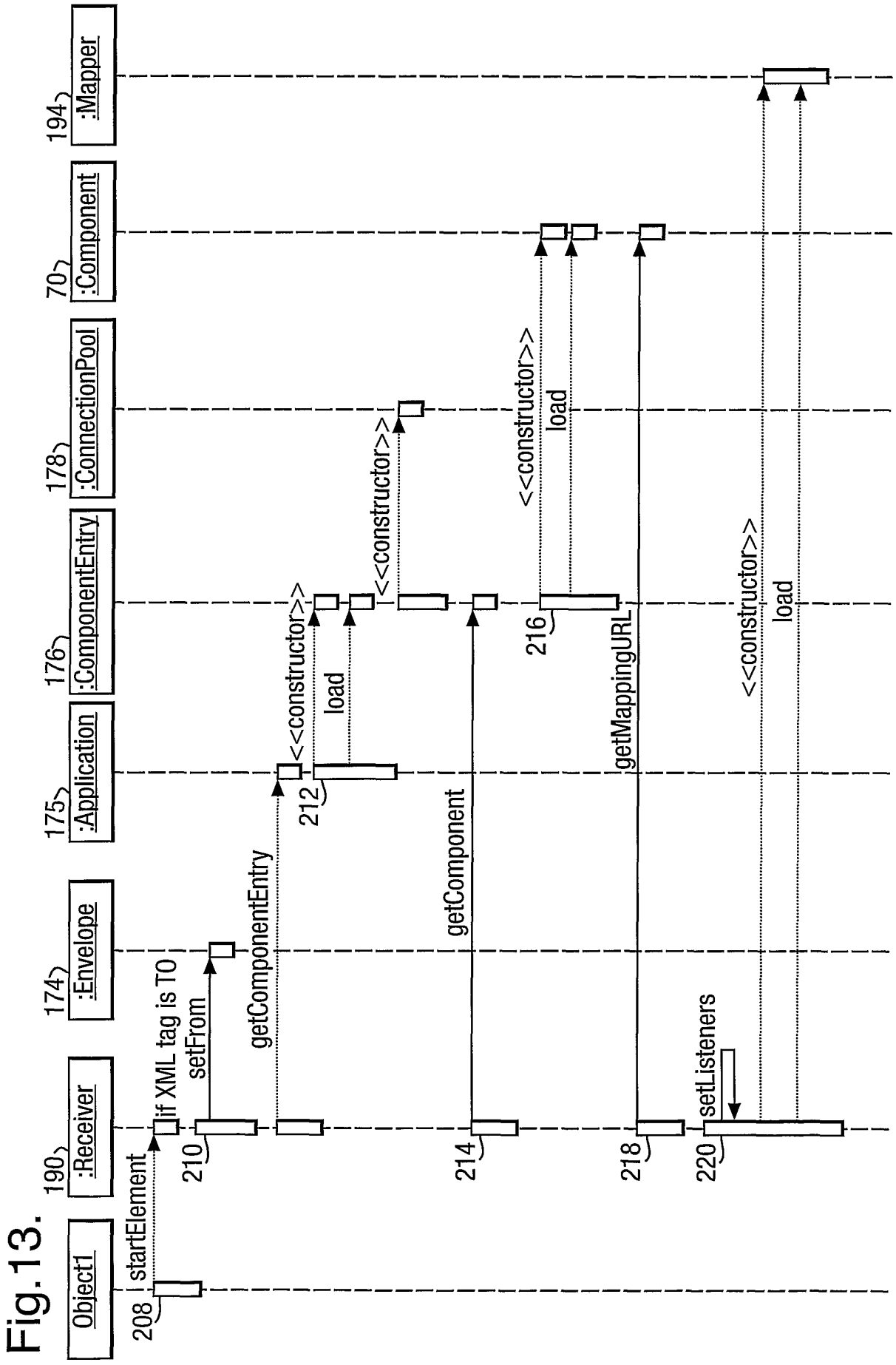

Object1
:Receiver
190
:Envelope
174
:Application
175
:ComponentEntry
176
:ConnectionPool
178
:Component
70
:Mapper
194
startElement
208
if XML tag is TO
setFrom
210
getComponentEntry
<<constructor>>
load
212
getComponent
214
<<constructor>>
load
216
getMappingURL
218
<<constructor>>
load
setListeners
220

Fig.13.

Fig.14.

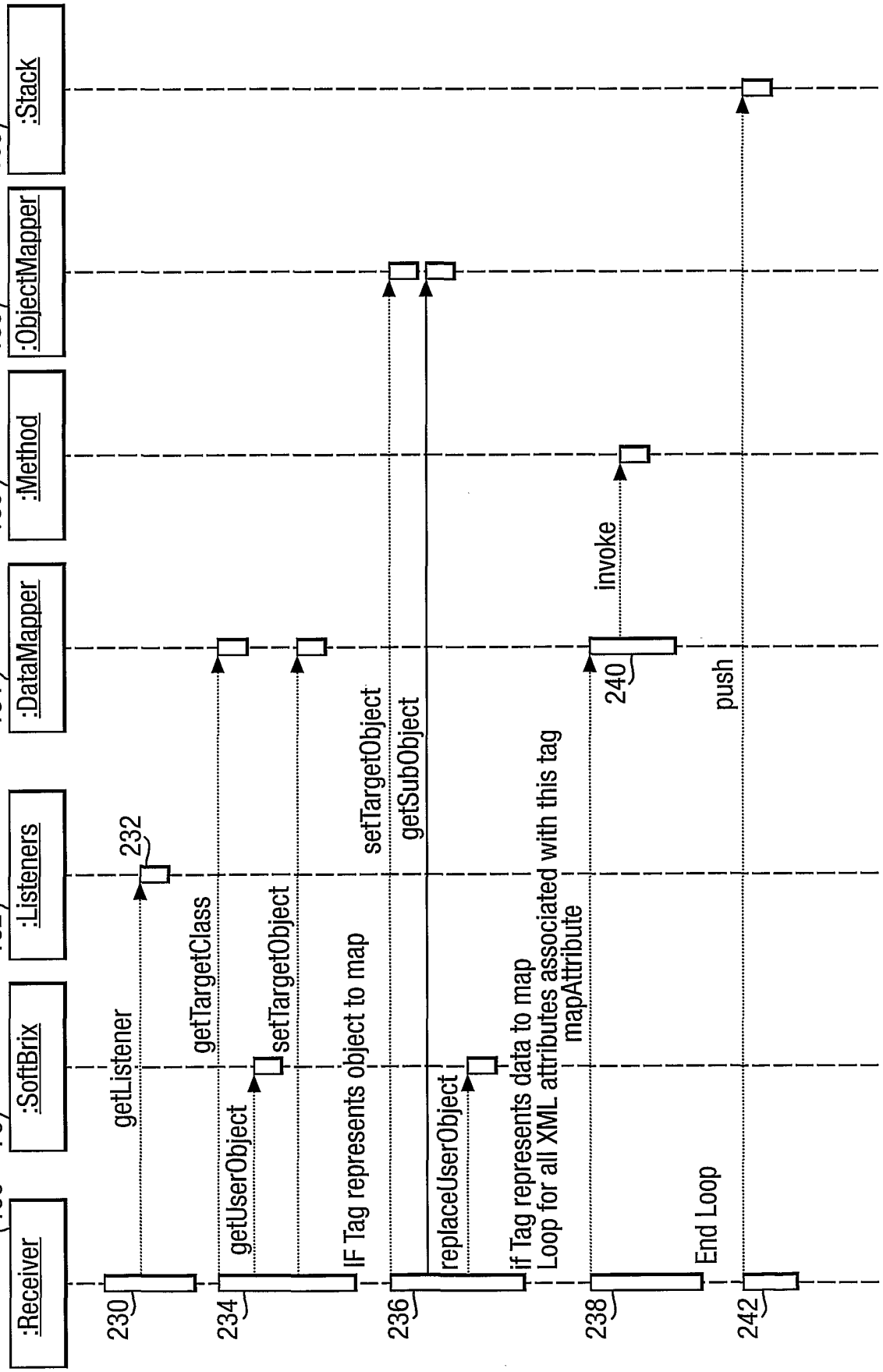
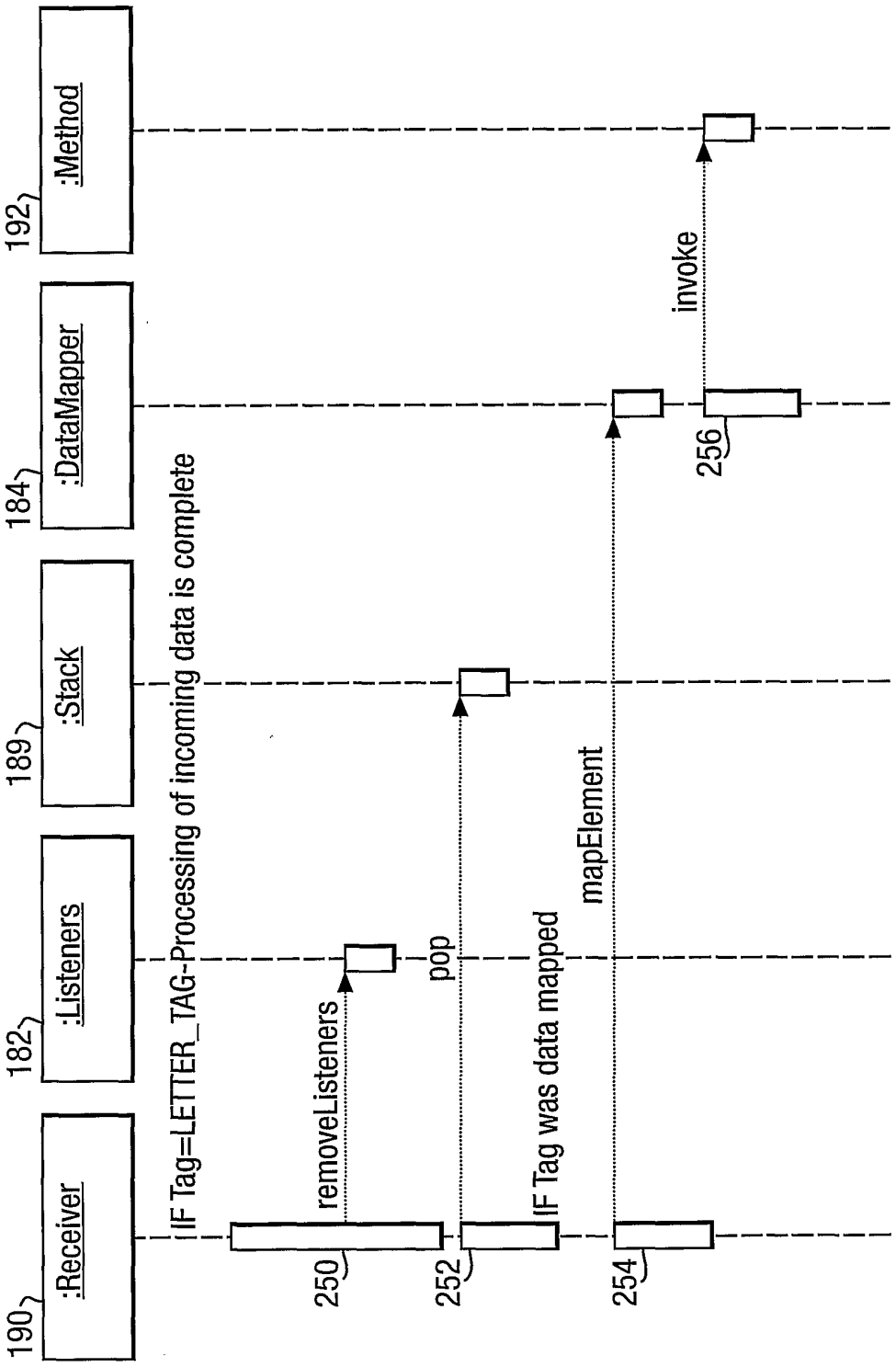

:Receiver
190
:SoftBrix
70
:Listeners
182
:DataMapper
184
:Method
186
:ObjectMapper
188
:Stack
189
getListener
232
getTargetClass
getUserObject
setTargetObject
IF Tag represents object to map
setTargetObject
getSubObject
replaceUserObject
if Tag represents data to map
Loop for all XML attributes associated with this tag
mapAttribute
invoke
240
push
End Loop
230
234
236
238
242

Fig.15.


:Receiver
190
:Listeners
182
:Stack
189
:DataMapper
184
:Method
192
IF Tag=LETTER_TAG-Processing of incoming data is complete
removeListeners
250
pop
252
IF Tag was data mapped
mapElement
254
invoke
256