Common Use Cases

This chapter covers the following topics:

- Introduction
- Case 1: ACI and ACI (including Units of Work)
- Case 2: JACI and ACI
- Case 3: ACI (via Web Server) and ACI
- Case 4: RPC Wrapper and RPC
- Case 5: Publisher (Natural Mainframe) and Subscriber (UNIX or Windows)

Introduction

This section provides common use cases of the basic concept of EntireX - achieving highly flexible interoperability of distributed application components. Each use case contains a

- business scenario
- table of interoperability, listing the major components selected for the use case
- diagram of the type of message flow resulting from the combination of these specific components
- stepped table describing the message flow depicted in the diagram.

The common use cases based on the EntireX components Broker and Developer’s Kit are provided to show the extent and limitations of the EntireX Broker.

The Developer’s Kit contains a set of interfaces for using applications written in various programming languages with EntireX Broker. Developer’s Kit enables application components to be "wrapped", i.e. encapsulated, thereby allowing them to behave like an object and be plugged-and-played as needed.

The ACI forms the layer upon which the various wrappers of the Developer’s Kit logically exist. This allows application programs to directly utilize the following industry-standard APIs that are exposed through the Developer’s Kit and EntireX Broker.

The common use cases in the table below are specific examples of how EntireX Broker provides highly flexible interoperability of application components in a distributed processing environment. The programming interfaces selected for the use cases below are organized by the two communication models exposed through EntireX Broker: client and server and publish and subscribe.

Case	Client	Server	Typical Use
Case 1	ACI	ACI	To integrate applications on separate platforms. (Persistent messaging is described.)
Case 2	JACI	ACI	To integrate applications on separate platforms, whereby the client's application interface is a subset of the ACI.
Case 3	ACI (via Web server)	ACI	To enable Web access to mainframe systems.
Case 4	RPC	RPC	To enable a UNIX or Windows application to access a Natural RPC program.
Case	**Publish**	**Subscribe**	**Typical Use**
Case 5	ACI	ACI	To enable a mainframe application to publish messages to UNIX or Windows subscribers.

Case 1: ACI and ACI (including Units of Work)

This case is typically used to integrate applications on separate platforms.

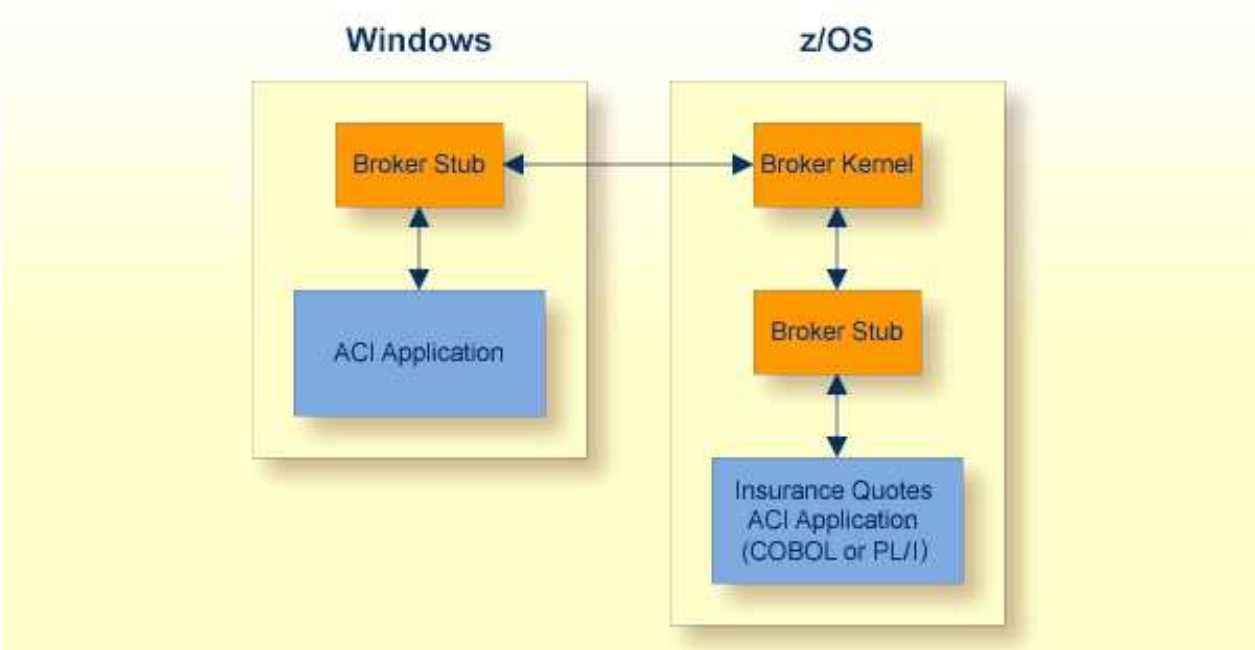
Business Scenario

An insurance company sells its own products as well as those of other insurers. It is company policy for its sales agents to give the most competitive insurance quotes possible to customers. The front-ends used by the sales agent are provided with GUI applications on Windows. To obtain insurance quotes from the back-end data as well as to update those data, the insurance agents must communicate information from/to various mainframe applications written in COBOL and PL/I.

Table of Interoperability

Application Component	Programming Interface	EntireX Component	Operating System	Language	Messaging Model
Client	ACI	Broker	Windows	Visual C	• Synchronous or asynchronous • Conversational or non-conversational
Server	ACI		z/OS	COBOL, PL/I	

Message Flow: ACI and ACI



Description of Steps in Message Flow

1.
    1. **Synchronous**
       The client program creates a request for information from a mainframe back-end and issues a call via the Broker stub to EntireX Broker.

       With conversational communication, a series of linked requests can be issued, allowing both the client and server to retain context between commands.

    2. **Asynchronous**

        - The client program wants to communicate updated information to the back-end system. It formulates one or more messages within a unit of work (UOW) and performs an asynchronous `SEND` from the stub to the broker.
        - The Broker writes the UOW to the persistent store, enabling the client program to know that the UOW will be processed.

2.
    1. **Synchronous**
       The server application issues an ACI call via the Broker stub in order to obtain the request from the client program.

    2. **Asynchronous**
       The server application issues a `RECEIVE` command, now or at a later time, in order to obtain the messages from the client program.

3.

1. **Synchronous**
   The server application processes the request and returns a message to EntireX Broker via the Broker stub.

2. **Asynchronous**
   The server program performs processing to update the data on the back-end system and, only afterwards does it acknowledge that the message has been processed.

4.

1. **Synchronous**
   The client program receives the reply to the ACI call, allowing the request to be satisfied.

2. **Asynchronous**
   The client program can query the status of its messages by `UOWID` in order to determine the status of the back-end processing.

Case 2: JACI and ACI

This case is typically used to integrate applications on separate platforms.

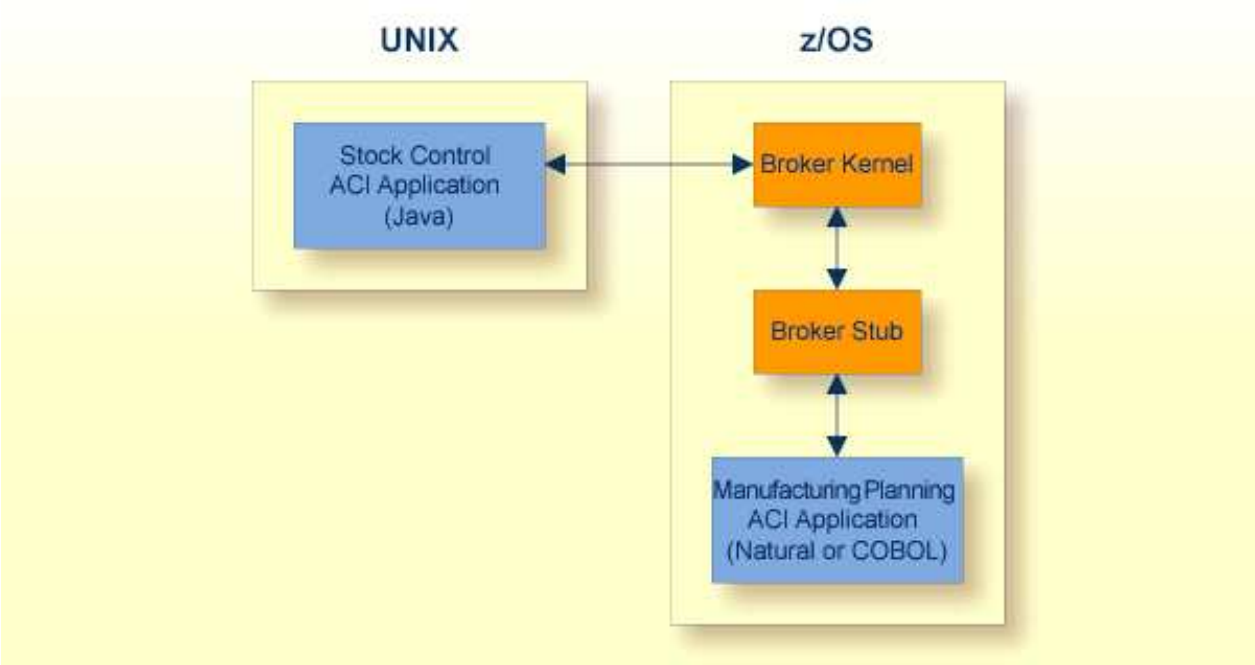
Business Scenario

An organization wants to integrate a UNIX-based stock control system with its existing mainframe-based manufacturing planning systems.

Table of Interoperability

Architecture	Programming Interface	EntireX Component	Operating System	Language	Messaging Model
Client	JACI	Broker	UNIX	Java	• Synchronous • Conversational or Non-conversational
Server	ACI		z/OS	Natural	

Message Flow: JACI and ACI



Description of Steps in Message Flow

1. The client program creates a request and issues a JACI call to EntireX Broker.
2. The server application issues an ACI call via the Broker stub in order to obtain the request from the client program.
3. The server application processes the request and returns a message to EntireX Broker via the Broker stub.
4. The client program receives the reply to the ACI call, allowing the request to be satisfied.

Case 3: ACI (via Web Server) and ACI

This case is typically used to enable Web access to mainframe systems.

Business Scenario

A brokerage has an application which processes orders of personal customers to buy and sell securities. All incoming orders are executed on a back-end system, and some orders are executed at a later time. The incoming orders are in the form of internet communication.

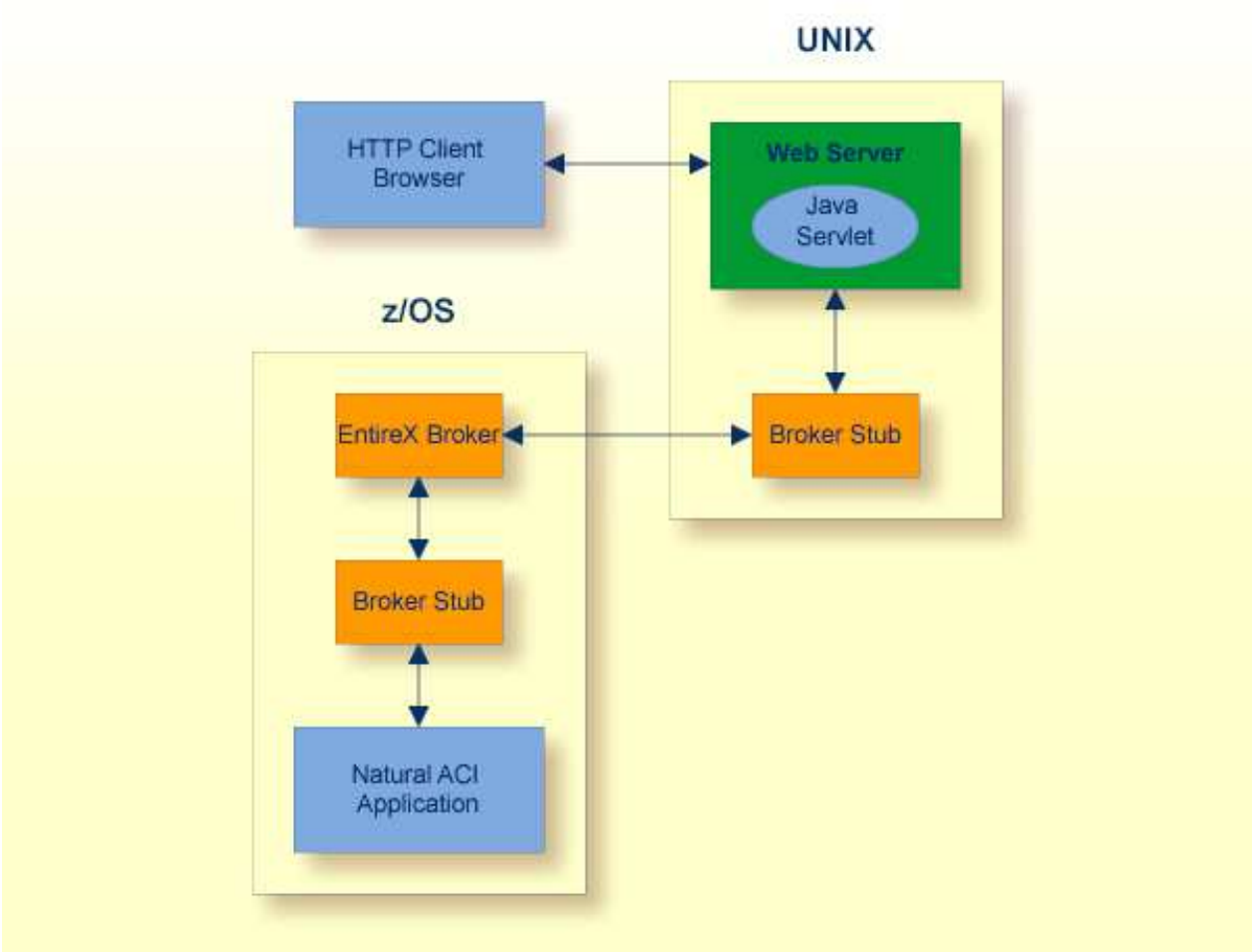
IT Environment

The brokerage uses a Web server as the point-of-entry for incoming orders. These orders are executed either synchronously or asynchronously on a separate back-end system. Located on the brokerage's Web server is an application which is a client to EntireX, which functions as a proxy and provides information to the brokerage's EIS (Enterprise Information System). Because of the critical nature of the orders, units of work are employed to guarantee delivery of the incoming information to the back-end system. This system is robust and can be restarted after failure without loss of data.

Table of Interoperability

Architecture	Programming Interface	EntireX Component	Operating System	Language	Messaging Model
Client	JACI	Broker	UNIX	Java Servlet	• Synchronous or Asynchronous • Conversational
Server	ACI		z/OS	Natural	

Message Flow: ACI and WebSphere MQ



Description of Steps in Message Flow

1. The Web browser sends an HTTP request to the Web server.
2. The Web server instantiates a Web page containing the script (ASP).
3. The script creates a request and issues an ACI call via the Broker stub to EntireX Broker.
4. The back-end application issues an ACI call via the Broker stub in order to obtain the request from the script.
5. The back-end application processes the request and returns a message to EntireX Broker via the Broker stub.
6. The script receives the reply to the ACI call, allowing the execution of the Web page to be completed.
7. The Web server returns the information to the Web browser via HTTP, where the Web page is displayed.

Case 4: RPC Wrapper and RPC

This case is typically used to enable a UNIX or Windows application to access a Natural RPC program.

Note:
This use case is the most common within EntireX; it employs the EntireX Broker together with the Developer's Kit.

Business Scenario

An organization actively using Software AG technology - including Adabas and Natural - wants to expand use of Software AG technology in order to build new applications accessible to clients executing under UNIX or Windows. To achieve this, the organization runs a client written to use RPC, which makes calls to EntireX Broker. The client, which is written in either Natural, Java or a 3GL language, will invoke any of these three variants:

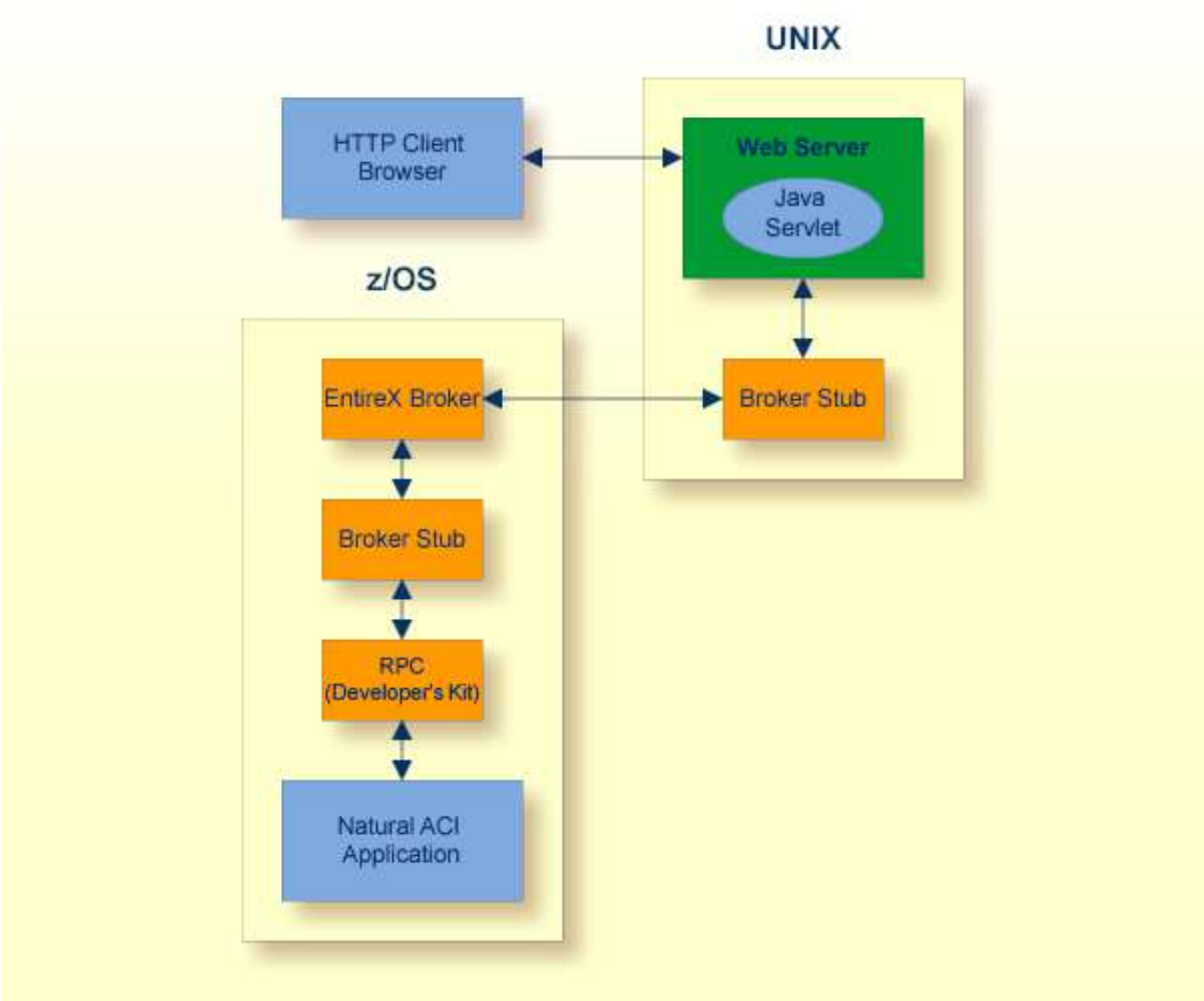
- **(A)**
  RPC programs written in Natural and executing under Natural on z/OS (RPC is available through Natural on z/OS);
- **(B)**
  RPC programs written in Java and executing under the Java RPC Server on UNIX;
- **(C)**
  3GL RPC programs executing under the C RPC Server on Windows.

Table of Interoperability

Application Component		Programming Interface	EntireX Component	Operating System	Language *	Messaging Model
(A)	Client	RPC	EntireX Broker and Developer's Kit	C	Visual Basic	• Synchronous • Conversational or Non-conversational
	Server	RPC		z/OS	Natural	
(B)	Client	RPC		Windows	Natural	
	Server	RPC		UNIX	Java	
(C)	Client	RPC		UNIX	Java	
	Server	RPC		Windows	C (=3GL)	

Message Flow: RPC Wrapper and RPC

This diagram represents variant (A) in Table of Interoperability above.



1. The client application ACI application initiates an RPC request through the SDK: synchronous/conversational or synchronous/non-conversational.
2. Broker stub communicates this request to the broker kernel.
3.
    1. **Natural**
       The broker kernel communicates this request to Natural nucleus, which behaves like an RPC server for Natural-written applications programs.
    2. **Java**
       Broker communicates this request to RPC server.
    3. **C**
       Broker communicates this request to RPC server.
4.

1. **Natural**
   Natural nucleus invokes the RPC server program.

2. **Java**
   RPC server invokes the server application program.

3. **C**
   RPC server invokes the server application program.

5.

1. **Natural**
   Natural nucleus returns the request to EntireX Broker.

2. **Java**
   RPC server returns the request to EntireX Broker.

3. **C**
   RPC server returns the request to EntireX Broker.

6. Broker passes the request to the ACI application.

Case 5: Publisher (Natural Mainframe) and Subscriber (UNIX or Windows)

This case is typically used to enable a mainframe application to publish messages to UNIX or Windows subscribers.

Business Scenario

A government department publishes details of various construction projects for which contractors are required. Companies are then able to bid for the contracts.

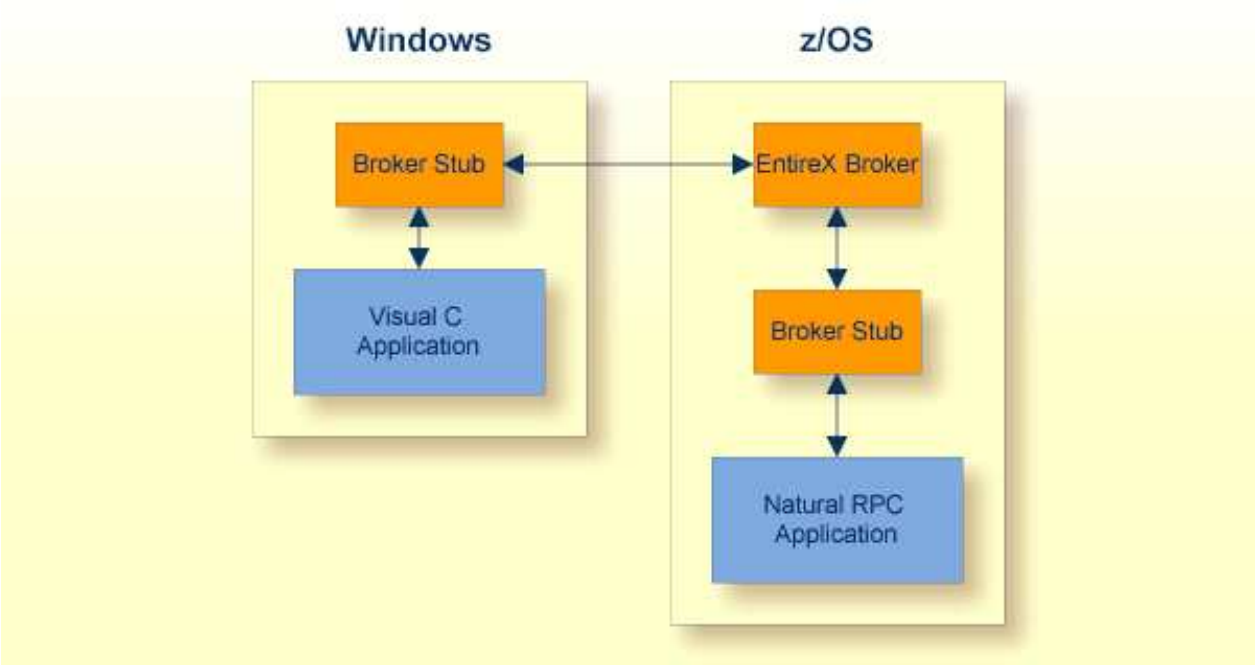
IT Environment

The government application consists of two pieces: a publisher and a subscriber component. An application running on z/OS publishes details for each new construction project. Publications are sent asynchronously with a logical topic name in accordance with the type of construction project required, for example freeways, minor roads, bridges. Approved contractors are given access to the subscriber component of the application which runs under Windows. Here the contractors can subscribe to the project types of interest and can receive details of projects for the specified project types at their convenience.

Table of Interoperability

Architecture	Programming Interface	EntireX Component	Operating System	Language	Messaging Model
Publisher	ACI	Broker	z/OS	Natural	• Asynchronous
Subscriber	ACI		Windows	Visual C	

Message Flow: Publisher and Subscriber



Description of Steps in Message Flow

1. The publisher component is executed when new publication messages are to be sent, using an ACI call via the Broker Stub to EntireX Broker.
2. EntireX Broker stores these publication messages into the persistent store, where they are available after a system restart.
3. The subscriber component is executed asynchronously, issuing an ACI call via the Broker stub to obtain published messages from EntireX Broker.
4. The subscriber repeats step (3) until all published messages have been received.