Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE @ DIRECT•

Computer Standards & Interfaces 25 (2003) 329–343

COMPUTER STANDARDS & INTERFACES

www.elsevier.com/locate/csi

CODEX: Content-Oriented Data EXchange Model on CORBA

Yue-Shan Chang[a,*], Ruey-Shyang Wu[b], Kai-Chih Liang[b], Shyan-Ming Yuan[b], Magic Yang[c]

[a] *Department of Electronic Engineering, Ming-Hsin University of Science and Technology, 1 Hsin-Hsing Road, Hsin-Fong, Hsinchu 304, Taiwan, ROC*

[b] *Department of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan, ROC*

[c] *W&Jsoft Inc., Taiwan, ROC*

Abstract

Asynchronous Data Exchange (ADE) is useful in Electronic Data Interchange (EDI). It needs a well-defined mechanism to support data buffing, dispatching, and filtering. ADE is more flexible, extensible and scalable than the synchronous one. CORBA Event Service, announced by OMG, offers an easy-to-use, abstract and generic communication mechanism, while providing message delivering, dispatching and buffering by way of introducing the concept of event channels. However, it lacks some practical data exchange functions.

This paper proposes the design and implementation of a system named CODEX (stand for Content-Oriented Data EXchange system), which combines the XML features and the functionality of CORBA (which stands for Common Object Request Broker Architecture) Event Service. The CODEX system equips functions of primitive data type checking, message classification, content filtering and source/destination screening. Application developers can focus more on business functions, instead of caring about the detail of data exchange. The concept and architecture of the content-oriented data exchange, provided by CODEX, can be applied to other practical computing fields in the large-scale, loosely coupled distributed computing environments.

Keywords: Asynchronous data exchange; Event Service; XML; CORBA

1. Introduction

Asynchronous Data Exchange (ADE) is naturally useful in the Electronic Data Interchange (EDI) field. The data providers and data consumers work independently while exchanging data with each other. They do not need to wait the peer receiving the data. ADE requires mechanisms to support data buffing/queuing, dispatching, and filtering in the system. It has much more flexibility, extensibility, and scalability than synchronous data exchange. There are many commercial products and industrial standards have adopted this concept, such as TIBCO TIB/Rendezvous, IBM MQSeries, Microsoft Message Queue, CORBA Event Service and Java JMS. Besides the mechanism, the content and the exchange format are also significant issues in an EDI system. The structu-

* Corresponding author. Tel.: +886-3-557-2930; fax: +886-3-559-1402.

E-mail addresses: ysc@must.edu.tw (Y.-S. Chang), ruey@cis.nctu.edu.tw (R.-S. Wu), kcliang@cis.nctu.edu.tw (K.-C. Liang), smyuan@cis.nctu.edu.tw (S.-M. Yuan), magic@wnjsoft.com (M. Yang).

ralized, not-complicated and well-defined format of data exchange is one of the critical success factors of EDI applications. Without a well-defined information format, applications cannot communicate with each other properly. For highly flexible applications, contents are the key criteria to the EDI system to perform proper data exchange. Different meanings for content requires different exchange routing.

1.1. Challenges

ADE is critical in large-scale systems, especially when not all applications are located in a single place. Frequent communication and large message size normally occur among application components. Under such conditions, the challenges are:

- *Network bandwidth minimization*. There are many application components exchange information through the network. In order to make the whole system operate properly, each component should not consume too much bandwidth. Unnecessary information should be screened out before delivering.
- *Security*. Some systems may have to exchange confidential information. To guarantee the information security, messages should only be received by authorized applications.
- *Integration*. Applications may be built on heterogeneous environment. To facilitate data exchange across heterogeneous environments, suitable mechanism has to be provided to integrate applications over different platforms.
- *Message validation*. Wrong messages may break the whole system integrity. To prevent the situation, message should be validated before being accepted by the target applications.
- *Robustness*. The data exchange platform must be robust enough to support large volume and high frequency traffic.

A suitable platform has to be chosen to build such a data exchange environment.

1.2. CORBA Event Service and Notification Service

The CORBA architecture [2] announced by Object Management Group (OMG) is becoming an industrial standard. The architecture, in the Common Object Service Specification (COSS) [1], also proposes the *Event Service*, which is an asynchronous data exchange framework. It separates the data supplier and data consumer while providing typed and un-typed event channels. The major advantage of the *Event Service* is that it offers an easy-to-use, abstract and generic asynchronous message delivery, dispatching and buffing mechanism across heterogeneous platforms [1,10,11]. However, it still lacks some of the necessary data exchange features [12,13], for example, *data security*, *message classification*, *content structure* and *type validation*, *data source/destination screening* and *content filtering*.

To compensate for the flaws of the *Event Service*, OMG also proposes the *Notification Service* [1,3,4]. It improves the event filtering by introducing a new event type, *structured event*. Structured event supports well-defined data structure. However, it is still insufficient for large-scale and complex data exchange. Event Service still lacks *data security*, *data validation* and *data source/destination screening*.

1.3. Related works

There are some commercial asynchronous data exchange systems.

TIB/Rendezvous [18] is an infrastructure developed by TIBCO. It exploits a three-level hierarchical event dispatcher for creating and maintaining large, distributed and event-based applications. TIB/Rendezvous is a subject-based addressing technology that helps messages reach their destinations without involving programmers in the details of network addresses, protocols, packets, ports and sockets. It offers several interesting features including reliable and scalable distribution of events. In subject-based addressing technology, the subject string also has a filtering mechanism. For example, the filter *economy.exchange.*.MSFT** will select all the notifications whose subject contains *economy* in its first position followed by *exchange* in the second position, any string in third position, and a fourth string that starts with *MSFT*.

Java Event-based Distribution Infrastructure (JEDI) [19,20] was developed by Italy CEFRIEL research center. JEDI is an object-oriented distribution infrastructure used in data exchange and is developed in Java programming language. In JEDI, every client has an Active Object (AO). An AO is an autonomy object. Each AO plays the role of exchanging events between every application. The event in JEDI is produced by AO, and is a special type of message like *open(foo.c, read)*. The component that dispatches the event is called event dispatcher (ED). The AO announces which event it wants to receive by calling event subscription procedures. Also, it can stop receiving some events by calling an event unsubscription procedure. JEDI has been used to implement a significant example of distributed systems, namely, the OPSS Workflow Management System (WFMS).

Asynchronous Message Exchange System (AMES) [17] is based on CORBA's Event Service [7]. It provides an asynchronous publish/subscriber communication model and can communicate and integrate with legacy systems easily. AMES extends the traditional CORBA Event Service from channel-based to a content/subject-based communication model by adding multiple layers based on Event Service. It can send different types of events using one event channel. We develop the management component to make up the lack of management mechanism in CORBA's Event Service. AMES also proposes hierarchical event advertise/subscribe/propagate architecture. It was developed based on Java programming language and Inprise Visibroker for Java product families.

Java Message Service (JMS) [21] provides a solution for Java programs to create, send, receive and read an enterprise messaging system's message. The service is defined by Sun Microsystems and integrated into Java 2 Enterprise Edition (J2EE) [22] architecture. JMS provides several message delivery strategies and many important functions for enterprise. It also contains simple message-parsing and filtering features based on JMS message objects. The message receiver can only get the interested message when the message objects contain the selector objects.

1.4. CODEX

From the previous experiences, we propose a Content-Oriented Data EXchange model named CODEX, which combines features of eXtensible Markup Language (XML) [5,14,15] and capabilities of CORBA Event Service [16]. XML is good for document modeling. Based on XML data type definition (DTD) or XML Schema [9], applications can perform the document validation. In addition, CODEX invents *primitive data type checking*, *message classification*, *source identification* and *content filtering* features based on the XML model. To facilitate the transportation of the CODEX XML model, in order to support content exchange, CODEX chooses CORBA Event Service as the infrastructure because the CORBA standard can easily facilitate heterogeneous system integration. Besides, several standard services are defined on the CORBA standard. It is good for enterprise applications to reuse such standard services while modeling content-oriented data exchange on CODEX system. These functional requirements of CODEX system are depicted as follows:

- *Content-oriented data exchange*. In the real data exchange scenarios, the routing of data may depend on the contents in the message. The design of the CODEX system has to provide the function wherein CODEX has to validate the structure of message contents and figure out certain key data fields. If some conditions are met, CODEX will allow the routing of the message; otherwise, CODEX will block the message.
- *Primitive data type checking*. It may result in unexpected behavior when the application receives incorrect data from data exchange. The design of CODEX has to allow the CODEX system to check the data type of certain key fields.
- *Message classification*. The design of the CODEX system has to provide the data security level classification mechanism. All data exchange in the CODEX has to be classified to a certain security level. When a consumer application is registering to the CODEX system, it has to negotiate the access security class. CODEX will check the security class of the content before it sends the data to the consumer.
- *Identification and screening*. A comprehensive data exchange system has to provide the function that can precisely transfer the message to proper destination, instead of blind broadcasting. In the

CORBA Event Service, it is only separating the supplier and consumer, and cannot identify the data source and data target. The design of CODEX has to offer the functionality to distinguish the source and target applications by introducing the identification mechanism. Based on this mechanism, CODEX has to provide the screening function which screens unnecessary traffic to or from the identity.

- *Data type and identity repository*. To support the above requirements, it is necessary to design a repository to hold all the necessary information in the CODEX system. The repository has to keep the data/document type definition as well as the source/target identity information to facility the correct operations of the CODEX system.

2. CODEX architecture

Fig. 1 shows the CODEX system architecture. The left side is the data supplier and right side is the data consumer. The CODEX is a data exchange server that consists of the *event channel object*, *supplier filter*, *consumer filter*, *Data Repository*, and *EXManage*. The detail of each components are depicted as follows.

2.1. Channel

The CODEX system was based on the Event Service of the Inprise VisiBroker. The Event Service on VisiBroker is a generic event channel. The exchanged data will transform *any* object into CORBA before entering it to the channel (in supplier filter) and will be transformed back to the original DOM object after leaving the channel (in consumer filter).

2.2. Supplier filter

An XML document will be processed by the supplier filter before entering the event channel. Each supplier filter will have a specific XML type (DTD) and connect to just one channel. Many filters can connect to the same channel. All the messages entering a channel will be validated. The validation includes:

- *XML Type Checking*. The XML parser is responsible for this work.
- *XML Validation*. The XML parser will validate the incoming document by referring the definition of XML document (DTD) stored in the Data Repository.
- *Data Type Checking*. Check all types of CODEX_Type tag attribute in XML document.

The supplier filter will discard the XML document if any error occurs. Otherwise, it will construct a tree data structure (DOM object) in memory after parsing.

2.3. Consumer filter

Just before the consumer receives an XML document from the event channel, the consumer filter will perform post-processing. A channel can connect many consumer filters, but each consumer filter can connect to just one channel. The consumer filter is the part of executing data filtering functions in the system. These filtering functions include data source

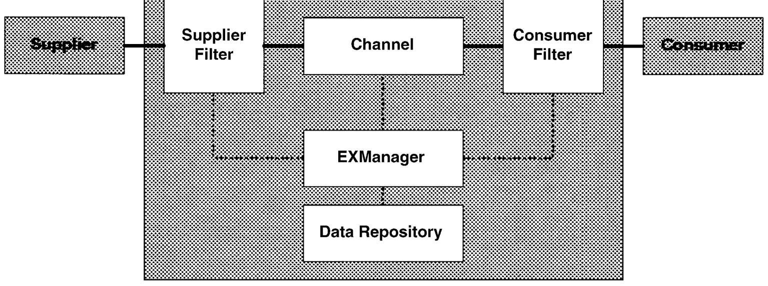

Fig. 1. CODEX system architecture.

filtering, data destination filtering, and data classification filtering.

- *Source filter*. The data consumer registers its data source filtering rule. The rule will be referred by corresponding filter. When a document is coming, the consumer filter validates the source of the XML document to see if the document was requested by the consumer or not.
- *Destination filter*. When a data supplier is publishing data, the data destination filtering rule will be appended. When the data gets to the consumer filter, the consumer filter will validate the attached data destination rule to see whether the consumer is allowed to receive the data or not.
- *Security filter*. All legal identity in CODEX will be assigned a priority. Higher priority identity can access equal or lower priority message. The consumer filter compares the identity priority and classified data that is in the *CODEX_Classification* tag attribute in the XML document. If the value of identity priority is larger than the value of *CODEX_Classification*, then the message will be delivered to the consumer.

Another important function of the consumer filter is format translation. This function is achieved by using the XSLT translation code registered by the data consumer. The registered translation code is kept in the consumer filter. The consumer application developer can compose the filter context using XSLT code for filtering the content. For the XSLT syntax, please refer to the XSLT specification in Ref. [6].

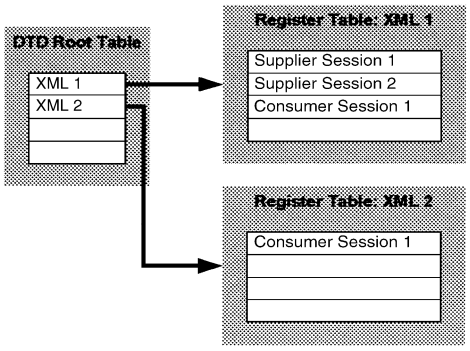

Fig. 2. Structures in the data repository.

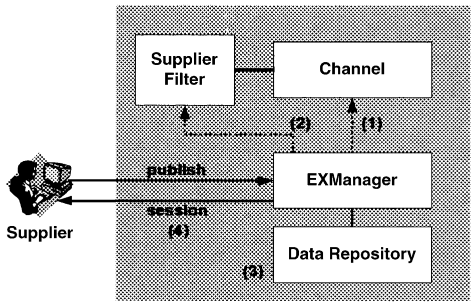

Fig. 3. Data registration and publication process.

2.4. Data repository

There are two kinds of information kept in the Data Repository: *identity information* and *document definition* (including related data supplier and data consumer). Identity information consists of login name, password, and priority. The Data Repository component offers services for EXManager to create/delete identity, change password and manage priority; it stores the document definition. Fig. 2 shows the structure of the document definition registration table. The left side of the figure is a document definition registration table. The right side of the figure is supplier/consumer registration table.

2.5. EXManager

The exchange manager (EXManager) in CODEX dynamically constructs and destroys all operation components, including event channel, supplier filter, and consumer filter. EXManager is a CORBA object, whose interface can be provided to applications that need data exchange service. EXManager manages identity information and manipulates objects in the Data Repository, and offers following the functionalities:

- *Identity login*. When an identity requests to enter the CODEX system, EXManager will inquire for identity information stored in the Data Repository, and determines identity authorization.
- *Data registration and publication process*. When a data supplier wants to publish data, it needs to register the information of publishable data. It includes document definition (DTD), consumer/supplier identification and data classification. After

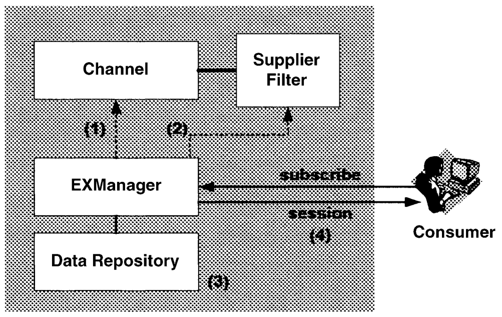

Fig. 4. Data subscription and receiving process.

the data supplier is successfully registered to the EXManager, it gets a session. The data supplier can then send data by way of this session. Fig. 3 shows the data publishing flow.

- *Data subscription and receiving process*. When a data consumer wants to subscribe data, it needs to register the information of document definition (DTD), consumer/supplier identification, data classification, XSLT translation code and the reference of callback function. After the data consumer successfully registers the information of subscription interests, it will get a session from EXManager. The data consumer can then receive data from this session. Fig. 4 shows the data subscribing follow.
- *Set source restriction rule*. The data consumer can set the source restriction rule (SRR) to the EXManager. SRR is used by the consumer filter to screen out undesired data sources.
- *Register XSLT translation*. The data consumer can register the XSLT code to EXManager in order to perform translation before the consumer receives the data.
- *Un-registration*. If the data exchange function is no longer required by the supplier or the consumer, it can issue the un-registration command to the EX-Manager.

3. CODEX design policy

The major objective of the CODEX system is to provide a content-oriented message exchange environment that equips data type checking, message delivery, data filtering and translation functions. The design of CODEX system chooses XML as the fundamental modeling language and adopts Data Type Definition (XML DTD), Document Object Model (DOM) [7,8] and eXtendible Style sheet Language (XSL) to further extend the functionalities of the system. This section reveals critical design issues of the CODEX system.

3.1. Centralized management of document definition

Each XML documents has to be accommodate by a corresponding DTD. These DTD files are registered and stored in a central repository. The XML parser in the supplier filter and consumer filter refer to the registered DTD to validate the correctness of structure of the incoming message. The functionality of the CODEX repository is the same as the Event Type Repository of CORBA Notification Service.

3.2. Data type checking

It will be seriously harmful if applications process data with the wrong type. It is necessary, in an EDI environment, to check the data type of incoming documents in order to reduce the possibility of missed operations. For example, in CODEX, applications can instrument data type for certain data fields like this:

```
<price CODEX_Type="int">20</price>
```

The *price* is defined as *int* type. The original DTD has to be modified as well.

```
The original DTD element declaration is
   <!ELEMENT Price (#PCDATA)>

Instrument type declaration like
   <!ATTLIST price CODEX_Type (int|float|char|string) #IMPLIED>
Where IMPLIED shows the attribute is not necessary. (int|float|char|string)
indicates the type of CODEX_Type may be only integer, float, character, or
string type.
```

If the system wants to have much more primitive types, then add new types directly into the declaration. When a new document definition is registered, CODEX will update the Data Repository. The data supplier can send data that conforms to the type declared in the tag. The supplier filter will also check the XML document by validating the attribute values in the tag.

3.3. Data classification

CODEX's data classification is designed for data exchange environment with a flexible and powerful data security mechanism. The classification shows the priority of content. This is to avoid lower-priority consumers from receiving higher-priority content. The granularity of data classification can be a single tag. It is possible that there are many classification levels in a single document. For example, the statement below means that the price element has to conform to the classification level "3".

```
<price CODEX_Classification="3">20</price>
```

Similarly, this functionality requires modifying the DTD in order to allow a new attribute in a tag: CODEX_Classification. The attribute is also *IMPLIED*. The value in the tag exhibits the set classification of document. The attribute definition is shown as the following:

```
<!ATTLIST price CODEX_Classification CDATA #IMPLIED>
```

When a new document definition is registered, CODEX will update the definition of the document in the Data Repository. The data supplier can add the class of data to a certain tag before sending data out. CODEX will check the consumer priority to make a decision whether the data may be sent to the consumer or not according to the data classification.

3.4. Identification and filtering

In Event Service, it is known that each data in a given channel will be sent to all the consumers, and each consumer will receive all of the data sent to the channel. It is desired that a supplier can assign the destinations of message to, and a consumer can select the sources of message from. And the filtering process should be done in the system. Therefore, consumers need to register their system identification and related information, such as the source restriction rule (SRR) into the system. Data suppliers will also register their identification into the system. The destination restriction rule (DRR) will be appended into the XML document to support system-checking and filtering.

When an identity registers the document definition, the system will append the following declaration into the document definition:

```
<!ELEMENT CODEX_root ( CODEX_DestRestrictRule? , root )>
<!ELEMENT CODEX_DestRestrictRule EMPTY>
<!ATTLIST CODEX_DestRestrictRule CODEX_YesNo ( yes | no ) "no">
<!ATTLIST CODEX_DestRestrictRule CODEX_Yes CDATA #IMPLIED>
<!ATTLIST CODEX_DestRestrictRule CODEX_No CDATA #IMPLIED>
```

The root indicates the root element of the original document. The modified document will have a new element: *CODEX_root*. The root element has two child elements: *CODEX_DestRestrictRule* and *root*. The element following the root is the original XML document that may be appended the type attribute and classification attribute. The *CODEX_DestRestrictRule* tag is an empty tag and it involves the DRR. The tag has three attributes:

- *CODEX_YesNo*: there are two setting modes for DRR. One setting is an acceptance list, the other setting is a rejection list. Here, the attribute *CODEX_YesNo* is a flag. Setting as 'yes' indicates that it accepts some specific identities while setting as 'no' indicates it reject some of identities. The default is set as 'no'.

- *CODEX_Yes*: the attribute is set to the acceptance list. If the setting of CODEX_YesNo is 'no', then the attribute is invalid. The content in the attribute is a string. For example, CODEX_Yes = "Host1; Host3; Host5".
- *CODEX_No*: the attribute is set to the rejection list. If the setting of CODEX_YesNo is 'yes', then the attribute is invalid. The value in the attribute is similar with CODEX_Yes attribute.

If suppliers have not set the *CODEX_DestRestrictRule* tag, the default value of *CODEX_YesNo* is "no". Then *CODEX_No* = "" indicates data sent to all the consumers connected to the channel. Similarly, the consumer can also use interfaces provided by CODEX to register the SRR, which is the same as DRR. If consumers have not set the SRR, the default value of *CODEX_SourRestrictRule* is "no". Then *CODEX_No* = "" indicates received data from all the suppliers connected to the channel.

3.5. Performance consideration

Performance is a major concern for a practical data exchange system. Obviously, the major overhead in the CODEX is using a XML parser to parse a XML document. The time is increasing while the size of the parsed XML document is growing. In general, the size of the XML document is unpredictable. The only way to reduce overhead is to reduce the parsing time.

There are three chances to parse XML documents in the CODEX. The first is data type checking before the message enters the event channel. The second is to create a source tree for the document before XSLT translation. The final is the parsing of the XSLT translation code. To reduce the requirement of XML parsing, we keep the data structure after data type checking and send it to the destination via event channel. The XML document has already been a source tree before arriving at the XSLT translation step. It is not necessary to parse it again. In addition, the XSLT translation code has been parsed in registration time and has resided in the memory. It does not need to parse translation code again and, hence, runtime overhead is reduced.

4. Programming interface

This section will briefly introduce the programming interface in the CODEX. We show the CORBA IDL–CODEX.idl, which includes supplier filter *PushS*, consumer filter *PushC*, *CallbackObject*, and *EXManager*.

The supplier filter plays the role of supplier of event channel and directly connects to the event channel. Hence, *PushS* needs to inherit from the *CosEventComm*::*PushSupplier* interface. *PushS* interface offers a method *pushXML()* for supplier to publishing data.

```
interface PushS : CosEventComm::PushSupplier{
    string pushXML(in CharArray XMLFile);
};
```

Consumer filter plays the role of consumer of event channel and also directly connects to the event channel. Hence, *PushC* needs to inherit from the *CosEventComm*::*PushConsumer* interface. *PushC* offers two methods for setting XSLT translation code and SRR. These two methods are all necessary giving identification UserKey to check authorization.

```
interface PushC : CosEventComm::PushConsumer{
long setXSLT(in string UserKey, in Doc XSLTFile);
    long setSourceRestrictRule(in string UserKey, in string Flag,
                               in string List);
};
```

It is asked to add a callback function when data consumer registers data subscribing. The callback function is invoked by consumer filter if a consumer desires the data received.

```
        Interface CallbackObject{
            void callback(in CharArray XMLFile);
        };
```

EXManager offers six methods. The *login()* checks authorization, and needs to pass three parameters: identity name, password, and site name etc. The *publish()* provides function for registering supplier publishable data. The method needs to pass three parameters: DTD name, user key, and DTD file. The *subscribe()* provides function for registering consumer subscribing data and the method needs to pass five parameters: DTD name, user key, DTD file, XSLT file, and callback function. The callback function is a stringed object reference. The *setXSLT()* provides the function for setting XSLT translation code. The parameters of this method are user key, session ID, and XSLT file. The *setSourceRestrictRule()* provides function to the consumer for setting SRR. The parameters of this method are user key, session ID, and a list of host IDs. The *unregisterDTD()* is to release the registered information.

```
        Interface EXManager{
            string login(in string UserName, in string Password,
                    in string SiteName);
            string publish(in string DTDName, in string UserKey,
                       in CharArray DTDFile);
            string subscribe(in string DTDName, in string UserKey,
                         in CharArray DTDFile, in CharArray XSLFile,
                         in string CallbackObjectReference);
            long setXSLT(in string UserKey, in string Session,
                     in CharArray XSLTFile);
            long setSourceRestrictRule(in string UserKey, in string Session,
                                  in string Flag, in string List);
            long unregisterDTD(in string DTDName, in string Session);
        };
```

5. Example and operation scenario

In this section, we present an example to demonstrate the scenario and operation follow of CODEX. A XML document definition DTD is shown as follows. The DTD is commonly recognized by the supplier and consumer to structure the exchanged data.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!ELEMENT man ( name , phone* , address+ , sex )>
<!ELEMENT name (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT address ( city , road? )>
<!ELEMENT sex EMPTY)>
<!ATTLIST sex type ( male | female ) #REQUIRED>
<!ELEMENT city (#PCDATA)>
<!ELEMENT road (#PCDATA)>
```

5.1. Data supplier registers publishable data

As mentioned above, registered document will be slightly changed, which adds new attributes into proper tag, such as *CODEX_Type* and *CODEX_Classification* attribute. In addition, data supplier also adds a lot of DRR attributes, such as the *CODEX_DestRestrictRule* attribute. Simultaneously, *EXManager* will allocate an event channel and a supplier filter to serve data exchange. And add the information of supplier into the entry of Data Repository. The modified document definition is shown as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT man ( name , phone* , address+ , sex)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT address ( city , road? )>
<!ELEMENT sex EMPTY>
<!ATTLIST sex type ( male | female ) #REQUIRED>
<!ELEMENT city (#PCDATA)>
<!ELEMENT road (#PCDATA)>
<!ATTLIST man CODEX_Classification CDATA #IMPLIED>
<!ATTLIST name CODEX_Classification CDATA #IMPLIED>
<!ATTLIST name CODEX_Type (int|float|char|string) #IMPLIED>
<!ATTLIST phone CODEX_Classification CDATA #IMPLIED>
<!ATTLIST phone CODEX_Type (int|float|char|string) #IMPLIED>
<!ATTLIST address CODEX_Classification CDATA #IMPLIED>
<!ATTLIST city CODEX_Classification CDATA #IMPLIED>
<!ATTLIST city CODEX_Type (int|float|char|string) #IMPLIED>
<!ATTLIST road CODEX_Classification CDATA #IMPLIED>
<!ATTLIST road CODEX_Type (int|float|char|string) #IMPLIED>
<!ELEMENT CODEX_man (CODEX_DestRestrictRule?, man)>
<!ELEMENT CODEX_DestRestrictRule EMPTY>
<!ATTLIST CODEX_DestRestrictRule CODEX_YesNo ( yes | no ) "no">
<!ATTLIST CODEX_DestRestrictRule CODEX_Yes CDATA #IMPLIED>
<!ATTLIST CODEX_DestRestrictRule CODEX_No CDATA #IMPLIED>
```

5.2. Data consumer registers subscribing data

When a data consumer wants to register the subscribing data, the document definition modification is the same as the data supplier described above. The activities which need to be done are information registration into Data Repository, event channel and consumer filter creation. Because the DTD is commonly recognized by supplier and consumer, the modified document definition will be equivalent regardless of which registration comes first, and there is just one copy stored in the Data Repository.

5.3. Data supplier

Data supplier can send XML data out after registration of publishable data. The following is an example of an XML document that will be exchanged. The XML file refers to an external DTD—http://Magic.dorm6.nctu.edu.tw/Dic/CODEX_man.dtd. This also shows the CODEX exchange server is running on the host: magic.dorm6.nctu.edu.tw. And the document definition is /Dic/CODEX_man.dtd file.

In the following example, the XML document indicates that the type of first phone number must be an integer. The second phone number only can be received by the consumer with priority larger than 2. And this document cannot be received by the consumer at the "Host5" host that is in the rejection list.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE
CODEX_man PUBLIC "Magic/CODEX_man.dtd"
 "http://Magic.dorm6.nctu.edu.tw/Dic/CODEX_man.dtd">
<CODEX_man>
<CODEX_DestRestrictRule CODEX_No="Host5"/>
<man>
<name>Magic</name>
<phone CODEX_Type="int">76121</phone>
<phone CODEX_Classification="2">0922876543</phone>
<address><city>HsinChu</city><road>Success</road></address>
<sex type="male"/>
</man>
</CODEX_man>
```

5.4. Supplier filter

The supplier filter will do a lot of checking when a XML document is sent to it, and discard illegitimate documents after type and format validation. In addition, all *CODEX_Type* tag attributes are removed from the XML document while the document is validated by the supplier filter, and the source identification will be appended into the document until consumer filter verifies the document. The modified document is shown as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE
  CODEX_man PUBLIC "Magic/CODEX_man.dtd"
  "http://Magic.dorm6.nctu.edu.tw/Dic/CODEX_man.dtd">
<CODEX_man>
<CODEX_Source name="Host2"/>
<CODEX_DestRestrictRule CODEX_No="Host5"/>
<man>
<name>Magic</name>
<phone CODEX_Type="int">76121</phone>
<phone CODEX_Classification="2">0922876543</phone>
<address><city>HsinChu</city><road>Success</road></address>
<sex type="male"/>
</man>
</CODEX_man>
```

Subsequently, Supplier filter parses the XML document using XML parser, constructs a DOM tree structure, encapsulates DOM tree structure into CORBA *Any* object, and sends it out by event channel. Event channel will store, dispatch, and deliver the *Any* object.

5.5. Consumer filter

Here, we assume the data consumer is located on "Host4" host and is prioritized as class 1. And he already set the SRR, the flag of *CODEX_YesNo* is "yes", and the acceptance list is "Host1; Host2; Host3".

Consumer filter will compare the DRR in the XML document with identification and find its own host ID "Host4" is not in rejection list "Host5". Next, it also finds that "Host2" is in the acceptance list "Host1; Host2; Host3" of SRR. Obviously, the document can pass the identification validation and be delivered to the destination host. The filtered XML document with the tag of *CODEX_Source* and *CODEX_DestRestrictRule* will be removed.

In addition, the priority of Host4 is 1, and the assigned class of tag "phone" is 2. The phone element will not be sent to the consumer due to its lower priority. If the priority of data consumer is larger than exchanged data, then only the *CODEX_Classification* tag will be deleted and phone tag will be retained. The following shows the XML document that passes all of checking.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE
  CODEX_man PUBLIC "Magic/CODEX_man.dtd"
  "http://Magic.dorm6.nctu.edu.tw/Dic/CODEX_man.dtd">
<CODEX_man>
<CODEX_Source name="Host2"/>
<CODEX_DestRestrictRule CODEX_No="Host5"/>
<man>
<name>Magic</name>
<phone>76121</phone>
<phone CODEX_Classification="2">0922876543</phone>
<address><city>HsinChu</city><road>Success</road></address>
<sex type="male"/>
</man>
</CODEX_man>
```

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
   mlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output encoding="UTF-8"/>
<xsl:template match="/">
  <html>
  <head><title><xsl:value-of select="*/*/name"/></title></head>
  <body bgcolor="#eeeeff">
  <h2>Personal data of <xsl:value-of select="*/*/name"/></h2>
  <xsl:apply-templates select="*/man"/>
  </body>
  </html>
</xsl:template>
<xsl:template match="man">
Sex : <xsl:value-of select="sex/@type"/>
Phone :
<ul><xsl:for-each select="phone">
  <li><xsl:value-of select="."/></li>
  </xsl:for-each></ul>
```

If the consumer does not register the XSLT translation code, data consumer will finally receive the above XML document file. It is well-known that XSLT is a powerful language for translating XML as various predictable formats. The following shows consumer register XSLT translation code and get a HTML formatted file. For XSLT specification, please see Ref. [6].

After translation, a familiar HTML file is given as follows:

```
<html>
<head><title>Magic</title></head>
<body bgcolor="#eeeeff">
<h2>Personal data of Magic</h2>
Sex : male
Phone :
<ul>
<li>76121</li>
<li>0922888888</li>
</ul>
Address :
  HsinChu - Success road.
</body>
</html>
```

As mentioned above, during registration, the consumer needs to provide a callback function for consumer filter to call back to consumer for sending desired data. Combining XSLT translation code and callback function can have a powerful effect for generating the desired format.

6. Appraisals and discussions

Content-oriented information exchange becomes an important feature in the distributed computing environment. CODEX adopts CORBA's Event Service to perform the message delivery. Other MOM software can also provide the same framework. The reason we chose CORBA standards as the CODEX infrastructure is as follows:

- *Heterogeneous platform and language support*. CORBA can integrate heterogeneous platform and support several programming languages. It can help different applications at different environments to work together, especially for the legacy mainframes.
- *Reusable services*. CORBA has complete software architecture and necessary services. CODEX utilizes Event Service to accomplish message exchange. Based on CORBA, applications can reuse not only Event Service but also other services. Most of MOM software only provides message transactions, naming service. No more horizontal or vertical reusable services for domain applications.

Another important feature of the CODEX system is the content-orientation. CODEX system naturally provides the content-based filtering and routing features for business applications. Other commercial MOM software or standards usually deliver messages as a whole and cannot provide content-based routing, unless software architects further instrument more components upon them.

Based on CORBA architecture, CODEX can be quickly deployed into different environments. We believe the time-to-production is also an important factor for the design and implementation of data exchange system.

7. Conclusions and future works

In this paper, we proposed a Content-Oriented Data EXchange system named "CODEX", which combines XML features and the functionality of CORBA's Event Service. CODEX offers functions like primitive data type checking, data classification and filtering, system identification and filtering. Obviously, the extension can conform with most of EDI's requirements. In addition, this paper presented the design philosophy, policy and consideration of CODEX; and depicted CODEX system architecture, CODEX's components and their interfaces. This paper also presented an example to demonstrate the scenario and operation follow of CODEX.

In the CORBA standard, distributed objects mostly communicate with each other through pre-defined or dynamic interfaces (IDL). In real practice, a large-scale distributed system requires loosely coupled business objects. Asynchronous data exchange between distributed objects, therefore, is desirable. As discussed in this paper, the type checking, content filtering, source/destination screening and message classification are emergent features. It suggests that CORBA standard is possible to offer more function-rich services based on Event Service/Notification Service to facilitate practical business application development.

Acknowledgements

We are grateful for the many excellent comments and suggestions made by the anonymous referees. This work was supported in part by the Nation Science Council of Republic of China under Grant No. NSC90-2213-E-159-005 and the Ministry of Education's Program of Excellence Research under Grant No. 89-E-FA04-1-4.

References

[1] Object Management Group (OMG), CORBAservices: Common Object Services Specification, Available at: http://www.omg.org/library/csindx.html (December, 1998).
[2] OMG, The Common Object Request Broker: Architecture and Specification, Available at: http://www.omg.org/library/c2indx.html (October, 1999).
[3] OMG, Notification Service—Joint Revised Submission, Available at: ftp://ftp.omg.org/pub/docs/telecom/99-07-01.pdf (August, 1999).
[4] DSTC, dCon—An OMG CORBA Notification Service Implementation, Available at: http://www.dstc.com/Products/CORBA/Notification_Service/ (April, 2000).
[5] World Wide Web Consortium (W3C), Extensible Markup Language (XML) 1.0 Specification, Available at: http://www.w3.org/TR/REC-xml (February, 1998).
[6] W3C, XSL Transformation (XSLT) 1.0 Specification, Available at: http://www.w3.org/TR/xslt (November, 1999).
[7] W3C, Document Object Model (DOM) Level 1 Specification, Available at: http://www.w3.org/TR/REC-DOM-Level-1/ (October, 1998).
[8] W3C, Document Object Model (DOM) Level 2 Specification, Available at: http://www.w3.org/TR/DOM-Level-2/ (November, 2000).
[9] W3C, XML Schema Part 2: Datatypes, Available at: http://www.w3.org/TR/xmlschema-2/ (May, 2001).
[10] D.C. Schmidt, S. Vinoski, Distributed callbacks and decoupled communication in CORBA, SIGS 8 (9) (October, 1996) 48–56.
[11] D.C. Schmidt, S. Vinoski, The OMG events service, SIGS 9 (2) (February, 1997) 37–46.
[12] D.C. Schmidt, S. Vinoski, Overcoming drawbacks in the OMG events service, SIGS 9 (6) (June, 1997). Also available at http://www.cs.wustl.edu/~schmidt/PDF/C++_report_col15.pdf.
[13] D.C. Schmidt, S. Vinoski, An introduction to CORBA messaging, SIGS 1010 (November/December, 1998). Also available at http://www.cs.wustl.edu/~schmidt/PDF/C+++_report_col10.pdf.
[14] S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, L. Tanca, XML-GL: A Graphical Language for Querying and Restructuring XML Documents. 8th International Conference on World Wide Web, Toronto, Canada, 1999, pp. 1171–1187.
[15] H. Maruyama, K. Tamura, N. Uramoto, XML and Java—Developing Web Applications, Addison Wesley, August, 1999.
[16] T.-Y. Hsiao, W.-T. Lo, S.-M. Yuan, An asynchronous message exchange system on CORBA, Proc. of the Technology of Object-Oriented Languages and Systems (TOOLS 2000 Pacific), 2000, pp. 14–23.
[17] T.-Y. Hsiao, W.-T. Lo, S.-M. Yuan, An asynchronous message exchange system on CORBA, Technology of Object-Oriented Languages and Systems. TOOLS-Pacific 2000, Proceedings 37th International Conference on Sydney, Australia, 19–20 Nov. 2000, pp. 14–23.
[18] TIBCO, TIB/Rendezvous, Available at: http://www.tibco.com/solutions/products/active_enterprise/rv/default.jsp (2002).
[19] G. Cugola, E. Di Nitto, A. Fuggetta, Exploiting an event-based infrastructure to develop complex distributed systems. Proceeding of the 20th International Conference on Software Engineering (IECS98), Kyoto, Japan.
[20] G. Cugola, E. Di Nitto, A. Fuggetta, The JEDI event-based infrastructure and its application to the development of the OPSS WFMS, IEEE Trans. Softw. Eng.
[21] Sun Microsystems, Java Message Service Specification, 2002.
[22] Java™ 2 Platform, Enterprise Edition, http://java.sun.com/j2ee.

Yue-Shan Chang was born on August 4, 1965 in Tainan, Taiwan, Republic of China. He received his BS degree in Electronic Technology from National Taiwan Institute of Technology in 1990, his MS degree in Electrical Engineering from the National Cheng Kung University in 1992, and his PhD degree from Computer and Information Science at National Chiao Tung University in 2001. Dr. Chang joined the Department of Electronic Engineering of Ming Hsing University of Science and Technology (MUST) as a lecturer in August 1992. In August 2001, he became an associate professor. He is now the Director of the Computer Center of MUST. His research interests are in Distributed Systems, Object Oriented Programming, Information Retrieval and Integration, and Internet Technologies.

Ruey-Shyang Wu received his BS and MS degrees in computer and information science from National Chiao Tung University, Taiwan, in 1998 and 2000, respectively. He is now a PhD student in the computer science department of the same school. His current research interests include Web technology, workflow technology and software engineering.

Kai-Chih Liang received his BS and MS degrees in computer and information science from National Chiao Tung University, Taiwan, in 1994 and 1996 respectively. He is now a PhD candidate in computer science of the same school. His current research interests include Web technology, distributed object computing architecture, high confidence middleware, enterprise application integration and software engineering.

Shyan-Ming Yuan was born on July 11, 1959 in Mauli, Taiwan, Republic of China. He received his BSEE degree from National Taiwan University in 1981, his MS degree in Computer Science from University of Maryland, Baltimore County in 1985, and his PhD degree in Computer Science from the University of Maryland College Park in 1989. Dr. Yuan joined the Electronics Research and Service Organization, Industrial Technology Research Institute as a Research Member in October 1989. Since September 1990, he has been an Associate Professor at the Department of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan. He became a Professor in June 1995. His current research interests include Distributed Objects, Internet Technologies, and Software System Integration. Dr. Yuan is a member of ACM and IEEE.

Magic Yang received his BS and MS degrees in computer and information science from National Chiao Tung University, Taiwan, in 1995 and 2000, respectively. He is an active consultant in the W&Jsoft, Taiwan. His current research interests include Web technology, workflow technology and software engineering.