Electronic Trading-Partner Agreement for E-Commerce

(tpaML)

Pre-submission Draft, Version: 1.0.6

Technical Contacts:

Martin Sachs
IBM T. J. Watson Research Center
POB 704
Yorktown Hts., NY 10598
USA
email: mwsachs@us.ibm.com

John Ibbotson
MQSeries Technical Strategy & Planning,
IBM UK Ltd,
Hursley Park,
Winchester, SO21 2JN
United Kingdom
email: john_ibbotson@uk.ibm.com

Preface

The Internet stretches traditional strict transaction processing concepts in several directions. First, transactions spanning multiple independent organizations may need to address enforcement of pairwise trading-partner agreements (TPAs). Second, a new transaction processing paradigm is required that supports different views of a unit of business for all participants, i.e., service providers as well as end consumers. This paradigm provides loose coupling of autonomous organizations rather than global data consistency. Global data consistency is not feasible in this environment, both because of long network delays and because it is not appropriate for autonomous organizations to lock each other's resources. There may be a unit of business consisting of several related interactions between any two interacting parties dispersed in time, creating a ***long-running conversation****. Hence, persistent records of business actions need to be kept. Additionally, since each organization's internal processes are independent of those of the other organizations, rollback of the effects of these interactions cannot be done. Instead the results of an interaction may be cancelable (however, this may not mean that all effects are undone, e.g., nonrefundable payments). The greater variability in response time for network computing creates a need for asynchronous and event-driven processing, in which correct handling of reissued and canceled requests is critical. The rules of interaction specified explicitly in the TPA permit interactions across autonomous organizations. This document presents a specification of electronic TPAs.*

Table of Contents

1 Objective

This specification defines the language for creating electronic trading-partner agreements (TPAs) between two business partners (parties to the TPA). Like the trading-partner agreements used in Electronic Data Interchange (EDI), these TPAs define the "information technology terms and conditions" that enable business documents to be electronically interchanged between partners. However, these TPAs are not paper documents. Rather, they are electronic documents, written in XML, which can be processed by computers at the partners' sites in order to set up and then execute the desired business information exchanges.

In general, the parties to a TPA can have both client and server characteristics. A client requests services and a server provides services to the party requesting services. In some applications, one party only requests services and one party only provides services. These applications have some resemblance to traditional client-server applications. In other applications, each party may request services of the other. As will be explained later, tags under the `<ServiceInterface>` tag define the client and server roles of the different parties.

A TPA describes all the valid visible, and hence enforceable, interactions between the parties and is independent of the internal business processes of each party. Each party builds its own internal business process to satisfy these external TPAs and interface them to the rest of its business processes. However, the internal business processes are in general not visible to other parties (unless desired by the service providers themselves). The intent is to provide a high-level specification that can be easily comprehended by humans and yet is precise enough for enforcement by computers.

> NOTE: It is possible to provide a graphic TPA-authoring tool that understands both the semantics of the TPA definition and the XML syntax. Equally important, the definitions in this specification make it feasible to automatically generate, at each party's site, the code needed to execute the TPA, enforce its rules, and invoke the application-specific programs.

A typical TPA consists of various kinds of information that are listed below. This list is not intended to show the structure of the TPA; the structure will be described in 2,"TPA Definition".

> NOTE: Many fields in the TPA are the same in most TPAs; the authoring tools and TPA templates can supply these fields. Where appropriate, the tool can supply default values for many fields; these only need to be stated when other than the default values are desired.

Information in the TPA includes:

1. *Identification,* to identify uniquely the TPA document, and the parties.
2. *Communication,* to specify the transport protocol(s), electronic addresses of the parties.
3. *Security*, to define the certificates used for authentication, nonrepudiation, and digital envelope, and other security parameters.

4. *Invocation-Independent Properties,* to specify overall properties of the TPA, e.g., the valid duration of the TPA.
5. *Data Definition,* for describing formats of the data being passed around.
6. *Role Definition,* that describes each of the roles specified in the TPA that can be filled by specific parties.
7. *Action list,* to describe the requests each party can issue to the other. These actions are the independent units of work. The action definitions define the associated message flows between the invoker and the service provider, responsiveness, failure handling, and other attributes.
8. *Sequencing rules,* to describe valid action invocation sequences in each party.
9. *Global properties,* to describe default properties of various fields in the TPA, e.g., responsiveness.
10. *Comments,* to describe handling of disputes, termination of the TPA as a whole, and other exceptional conditions.

> NOTE: Conceptually, the TPA may be considered to be implemented by a business to business (B2B) server at each party's site. The B2B server provides the code for the services needed to support the TPA including the middleware which supports communication with the other party, execution of the functions specified in the TPA, interfacing to each party's back-end processes, and logging the interactions between the parties for purposes such as audit and recovery. The middleware might support the concept of a long-running conversation as the embodiment of a single unit of business between the parties. This specification uses the term "registration" to denote the process of setting up the TPA for use at each party's site, i.e. recording the static information in a local database and generating the necessary code to support the TPA.

1.1 Normative References

In general, references listed below are those for which specific XML definitions are provided in the TPA. Other specifications are also referred to in this specification in the sense that they are represented by keywords for which the parties to the TPA may obtain plug-ins or write custom support software but do not require specific XML tag sets in the TPA.

In a few cases, the only available specification for a function is a proprietary specification. These are indicated by notes within the citations below.

The following standards contain provisions that, through reference in this standard, constitute provisions of this standard.

1. Extensible Markup Language (XML), World Wide Web Consortium.
2. Key Words for use in RFCs to Indicate Requirement Levels, Internet Engineering Task Force RFC 2119.
3. ebXML Message Header Specification: The precise title and document identification will be added when known. This specification is normative for new applications but may be disregarded for legacy applications.
4. Hypertext Transfer Protocol, Internet Engineering Task Force RFC2616.
5. Standard for the Format of ARPA Internet Text Messages, Internet Engineering Task Force

RFC 822.
6. S/MIME Version 3 Message Specification, Internet Engineering Task Force RFC 2633.
7. MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies. Internet Engineering Task Force RFC 1521.
8. Simple Mail Transfer Protocol, Internet Engineering Task Force RFC 821.
9. File Transfer Protocol (FTP), Internet Engineering Task Force RFC 959.
10. Secure Sockets Layer, Netscape Communications Corp. http://developer.netscape.com. NOTE: At this time, it appears that the Netscape specification is the only available specification of SSL. Work is in progress in IETF on "Transport Layer Security", which is apparently intended as a replacement for SSL.
11. Digital Envelope, RSA Laboratories, http://www.rsasecurity.com/rsalabs/. NOTE: At this time, the only available specification for digital envelope appears to be the RSA Laboratories specification.

NOTE: Other standards referenced by this standard will be added to this list.

1.2 Terminology

In this specification, the term "framework" denotes the software infrastructure that provides services which support the TPA. The framework may also call one or more methods of an application which implement a particular business process.

The term "application" or "business application process" denotes the software at one party to the TPA which performs the business application and communicates with the corresponding application at the other party. An action is an entry point to the application. It may represent the whole application or a single method of the application.

This specification conforms to RFC 2119 (see ref. 2 in 1.1, "Normative References"), which defines key words to indicate requirement levels, except for use of the word "optional".

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED" and "MAY" in this document are to be interpreted as described in RFC 2119.

In this specification, the word "optional" in the description of an XML tag means that the DTD or XML Schema defines the tag as occurring zero or one, or zero or more, times. Optional tags shall be supported by all vendors except where it is stated to the contrary in the description of an individual tag. Vendors shall provide support for the optional tags as needed in all TPA tools and in the framework.

In this specification, indented paragraphs beginning with "NOTE:" provide informative explanations or suggestions which are not required by the standard.

1.3 Document organization

This specification is organized as follows. Section 2, contains the definition of the TPA, identifying the structure and all necessary fields. The appendices include the complete XML TPA definition, a DTD, an XML Schema document equivalent to the DTD, a sample application, and various samples of TPAs.

2 TPA Definition

2.1 Overview

This section describes in detail each of the TPA sections, and the syntax for specification of these sections.

The TPA may exist at several different levels of abstraction:

- An abstract description, as in this standards specification, from which any TPA among any parties for any purpose could be derived.
- A prototype TPA (template) between two or more parties for a specific purpose. This template can be tailored as needed, resulting in a particular TPA. In the template, one or more parties can be represented by role parameters, such as *name*, to be replaced by specific parties in a particular TPA.
- An actual TPA among a specific set of parties.
- Instantiation of the TPA among a specific set of parties for a particular long-running conversation.

  NOTE: Following is an abstract description of how support of the TPA might be implemented. This description is not part of the TPA specification. Although object-oriented terminology is used here, there is no implication that TPA support must be in this fashion.

  The TPA is instantiated as a conversation by the framework, triggered by application code outside the scope of the TPA. Instantiation of the TPA as a conversation results in the creation of objects at each party that represent the TPA instance. They are known as server TPA objects and client TPA objects.

The TPA is formulated as an XML document (see ref. 1 in 1.1, "Normative References"). The tags are formally defined in a DTD file which is shown in appendix "2 DTD Corresponding to Complete TPA Definition". The corresponding XML Schema document is shown in appendix "3 XML Schema Document Corresponding to Complete TPA Definition".

2.2 Formats of Character Strings

In XML, the lowest level tags in any subtree generally have values which are character strings (#PCDATA in XML terminology). Tag names, attribute names, and attribute values are also character strings. This section defines format rules for these strings. Exceptions to these rules, and other details, are stated in the description of each tag.

2.2.1 Tag Names and Attribute Names

Tag names and attribute names are case sensitive (must match DTD and XML Schema document).

2.2.2 Attribute Values

If the DTD or XML Schema document specifies the permissible values for a particular attribute, the attribute stated in the TPA must match the value of one of the attributes defined in the DTD or XML Schema document including case. In this specification, case sensitivity is implied whenever a specific list of values is shown for an attribute.

Attribute values for which the DTD or XML Schema document does not specify a set of choices are also case sensitive.

Several tags, such as `<Action>`, have attributes (e.g. `ActionId`) which define identifiers that must be unique throughout the TPA.

> NOTE: An authoring tool could ensure uniqueness of these identifiers.

2.2.3 Protocol and Version Tags

Values of `<Protocol>`, `<Version>`, and similar tags are flexible. In general, any protocol and version for which the support software is available to both parties may be selected as long as the choice does not require changes to the tpaML DTD or schema and therefore a change to the tpaML specification.

> NOTE: A possible implementation may be based on the use of plug-ins or exits to support the values of these tags.

2.2.4 Party Names and Reference Attributes

Party names are referred to in various places in the TPA. In order to ensure that party names are used consistently throughout the TPA, the party name is defined in the `<PartyName>` tag under `<Participants>` as an ID-type attribute named `Partyname`. The value of `<PartyName>` is the name of the party. It is case sensitive. For example:

```
<PartyName Partyname="ComputerCo">Computer Co</PartyName>
```

The `<OrgName>` tag is ued in those places that have to refer to the party name. It contains an IDREF-type attribute named `Partyname`. The value of this attribute must match (case sensitive) the value of the attribute in the matching `<PartyName>` tag. For example:

```
<OrgName Partyname="ComputerCo"/>
```

It is recommended, for readability, that the value of the `Partyname` attribute be the name of the party, preceded by an underscore, and with any spaces omitted. For example, for Pens Are We, use "PensAreWe". The attribute value must be written the same way wherever it is used.

> NOTE: The reason for the leading underscore is to accommodate party names that begin with numeric characters, such as 3COM and 3M. The underscore is needed to make the attribute value conform to the XML rules for valid ID-type attribute values.

For a TPA written in terms of roles (see 2.4.3, "Role Definitions"), it is recommended, for

readability, that the value of the `Partyname` attribute be the role name as given in `<RoleName>`, preceded by an underscore. In role substitution, the value of `<PartyName>` is replaced by the actual party name but the `Partyname` attribute value is not changed. For example:

```
<PartyName Partyname="obiseller">obiseller</PartyName>
<OrgName Partyname="obiseller"/>
```

2.2.5 Alphanumeric Strings

Alphanumeric strings not further defined in this section follow these rules unless otherwise stated in the description of an individual tag:

- Values of tags in the TPA are case insensitive unless otherwise stated.
- Strings that are copied into a database at TPA registration time for use by other programs or people and are not otherwise processed by the framework have no format requirements from a TPA viewpoint. Other programs that process this information may have requirements. In some cases, this specification recommends particular formats.
- Strings which represent file or directory names are case sensitive to ensure that they are acceptable to both UNIX and Windows systems.

  NOTE: Tag values in the TPA which may map to method names, such as the value of <RequestName>, are not case-sensitive in the TPA. A TPA tool which maps these names into quantities, such as method names, which may be case sensitive is responsible for dealing with that case sensitivity.

2.2.6 Numeric Strings

A numeric string is a signed or unsigned decimal integer in the range imposed by a 32-bit binary number, i.e. -2,147,483,648 to +2,417,483,647. Negative numbers may or may not be permitted in particular tags; for example, a time interval is always positive.

2.2.7 Addresses (geographic, postal)

Values of tags which comprise a geographic or postal address (e.g. `<City>`, `<State>`, `<Zip>`) are in USA format.

> NOTE: An authoring tool can accept input of other national forms of address and convert them to US format. Local mailing software can convert and format the values of these tags as appropriate (e.g. to the national destination of a letter). The value of `<Zip>` should allow enough digits to contain the largest known postal code.

2.2.8 Telephone Numbers

A telephone number or fax number consists of a numeric string containing the area code followed by the phone number with no punctuation marks.

2.2.9 Time and Date Quantities

2.2.9.1 Time Intervals

A time interval is expressed as a numeric character string whose value is a positive integer number of seconds.

> NOTE: An authoring tool could allow the author to enter time intervals in any convenient units, perhaps selected from a pull-down list, and convert the input to seconds.

2.2.9.2 Dates

A date is expressed in United States format, mm/dd/yyyy, i.e. a three numeric integer strings separated by slashes, a 2-digit month, a 2-digit day, and a 4-digit year.

> NOTE: An authoring tool could accept other date formats and convert them to the US format.

2.2.9.3 Time of Day

A time of day is expressed in the form hh:mm:ss (hours, minutes, seconds) using a 24-hour clock, i.e. the value of hh is in the range 00-23. Times are expressed as Coordinated Universal Time.

2.2.10 URLs

A URL conforms to the standard URL format. Fields to the left of the first slash (i.e. protocol and domain name) are case insensitive. Fields to the right of the first slash are case sensitive since most such fields are directory or file names, which must be case sensitive to conform to UNIX rules.

2.3 TPA Structure

In this section we describe the overall structure of the TPA. Unless otherwise noted, TPA fields must be in the order shown here. Subsequent sections describe each of the functions provided by the TPA.

Following is the overall structure of the TPA, expressed in XML.

```
<TPA>
<TPAInfo>  <!-- TPA preamble -->
     ...  <!--TPA name, participants, etc.-->
     </TPAInfo>
<DeliveryChannelSet>
     <!--paired transport and doc exchange definitions-->
     </DeliveryChannelSet>
<Transport>
     ... <!--communication information-->
     </Transport>
<DocExchange>
     ... <!--document exchange information-->
```

```
     </DocExchange>
<BusinessProtocol>
     ... <!--Business protocol information-->
     </BusinessProtocol>
<Comment>
     <!--any reference text-->
     </Comment>
</TPA>
```

The <BusinessProtocol>, <DocExchange>, and <Transport> sections describe the processing of a unit of business (conversation). These sections form a layered structure somewhat analogous to a layered communication model. The following descriptions include both what is defined in the TPA and the corresponding run-time processing.

Business-Protocol layer - The Business Protocol layer defines the heart of the business agreement between the trading partners: the services (actions) which parties to the TPA can request of each other and sequencing rules that determine the order of requests. The Business-Protocol layer is the interface between the TPA-defined actions and the business-application functions that actually perform the actions.

Delivery Channels - A delivery channel consists of one document-exchange definition and one transport definition. Several delivery channels can be defined in one TPA.

Document-Exchange layer - The Document Exchange layer accepts a business document from the Business Protocol layer, optionally encrypts it, optionally adds a digital signature for nonrepudiation, and passes it to the transport layer for transmission to the other party. The options selected for the Document Exchange layer depend on those selected for the Transport layer. For example, if message security is desired and the selected transport protocol does not provide message encryption then it must be specified at the Document-Exchange layer.

Transport layer - The transport layer is responsible for message delivery using the selected transport protocol. The selected protocol affects the choices selected for the Document-Exchange layer. For example, some transport layer protocols may provide encryption and authentication while others have no such facility. Optional functions selected for the Document-Exchange layer must take into account the nature of the transport layer.

It should be understood that the functional layers encompassed by the TPA have no understanding of the contents of the payload of the business documents.

The TPA preamble (<TPAInfo>) contains those definitions that are independent of instantiation, such as TPA name, role definitions, party names, and TPA duration. The optional comment section (<Comment>) may contain any reference text that must be included in the TPA.

2.4 TPA Preamble

The preamble identifies the TPA and its participants and defines some invocation-independent properties such as the valid duration. The preamble is defined by the information under the <TPAInfo> tag.

2.4.1 Identification

The TPA is given a unique identification string, such as US14442869, using the <TPAName> tag.

2.4.2 TPA Type

The <TPAType> tag provides information about the general type of TPA where the TPA is based on a defined standard business-level protocol such as OBI™. It is optional. It includes the following information: business protocol, protocol version, and protocol type. The syntax is

```
<TPAType>
     <Protocol>protocol</Protocol>
     <Version>version</Version>
     <Type>type</Type>
     </TPAType>
```

The <Protocol> tag identifies the business-level protocol. The <Version> tag identifies the version of the protocol being used.

The <Type> tag provides additional information about the business protocol. Specific values depend on the particular protocol and its optional features.

> NOTE: An example of a protocol is OBI. For OBI, an example of the value of <Version> is 1.0, and the values of <Type> are SS (server to server), and SBS (server-browser-server), which define whether or not one or more of the message flows is via the client's browser.

The values of all tags under <TPAType> are case-insensitive alphanumeric strings.

2.4.3 Role Definitions

A TPA may be between specific parties or between some combination of specific parties and arbitrary parties (roles). The <Role> tag defines the relationships between roles and actual parties. It is optional and should be omitted if roles are not used. A TPA containing one or more roles is referred to as a prototype TPA or a reusable TPA. The resulting TPA, with the roles replaced by specific parties, is called a fully qualified TPA.

The syntax of the role definition section is:

```
<Role>
     <RoleDefn>     <!--one or more-->
          <RoleName>role_name</RoleName>
               <!--example:  hotel, airline -->
```

```
        <RolePlayer>player_name</RolePlayer>
             <!--example:  Hotelco, Airlineco-->
        </RoleDefn>
    </Role>
```

In a prototype TPA, the value of <RolePlayer> may be null, blank, or an arbitrary character string. This must be replaced by an actual party name when a party is assigned to a role.

An arbitrary party is referred to as a role and is denoted by a role parameter. The <RoleName> tag defines role parameters. Any alphanumeric string may be used as a role parameter. The actual party name is given in the associated <RolePlayer> tag.

> NOTE: An authoring tool or other tool which replaces role parameters by role names should treat the role parameter as case-insensitive in both <RoleName> and <Party> when comparing these strings for equality. In other words, the matching names should compare equal regardless of case.

A specific party must be assigned to play each unfilled role during the instantiation of a TPA from a prototype TPA. To identify a specific party to play a role, the role parameter must be replaced by the actual party's organization name.

Role parameters are used in the <PartyName> and <Arbitrator> tags. For example, where the role parameter is airline:

```
<PartyName Partyname="airline">airline</PartyName>
```

The leading character of the value of the Partyname attribute must not begin with a numeric character in order to satisfy the XML requirement that ID attributes must not begin with numeric characters.

> NOTE: If the party name begins with a numeric character (e.g. 3Com, 3M), either the digit should be spelled out (e.g. ThreeCom) or the digit should be prefixed with a non-numeric character such as "_".

The <OrgName> tag identifies the parties in other places in the TPA. The identification is via the Partyname attribute. See 2.2.4, "Party Names and Reference Attributes".

In addition, party-dependent tags must be given the appropriate values. In general, these tags are in a subtree below the tag of which <OrgName> is a subelement. For example, when the value of <PartyName> is converted from a role parameter to an actual party name, most tags under <Member> must be given party-specific values. The <Member> tag also has party-dependent attributes (IdCodeType and MemberId) which must be filled in when replacing a role by a specific party. An example is the following fragment of a reusable TPA. The complete reusable TPA is in Appendix 5.1 Reusable TPA.

```
<!--*******************************************************-->
<!--Role parameters                                        -->
<!--*******************************************************-->
```

```
    <Role>
     <RoleDefn>
       <RoleName>OBIBuyer</RoleName>
       <RolePlayer> </RolePlayer>
      </RoleDefn>
     <RoleDefn>
       <RoleName>OBISeller</RoleName>
       <RolePlayer> </RolePlayer>
      </RoleDefn)
     </Role>
    <Participants>
<!--*****************************************************-->
<!-- Specification of Buyer                              -->
<!--*****************************************************-->
      <Member IdCodeType=" " MemberId=" ">
        <PartyName Partyname="obibuyer">OBIBuyer</PartyName>
        <CompanyTelephone> </CompanyTelephone>
       <Address>
          <AddressType>location</AddressType>
          <AddressLine> </AddressLine>
          <AddressLine> </AddressLine>
          <AddressLine> </AddressLine>
          <City> </City>
          <State> </State>
          <Zip> </Zip>
          <Country </Country>
        </Address>
```

When the value of the `<RolePlayer>` tag directly below the first `<RoleName>` tag is filled with the name `LargeCo`, the following must be done:

- Set the value of `IdCodeType` to, for example, `01` for DUNS.
- In `<PartyName>`, set the value of the `MemberId` attribute to Large Co's DUNS number.
- Replace the value of `<PartyName>` to `LargeCo`.
- Fill in the values of all the tags following `<PartyName>` with values of the telephone number and various address components by values specific to Large Co.

The resulting fully-qualified TPA is shown in Appendix 5.2 Fully Qualified TPA.

> NOTE: A prototype TPA written in terms of roles might contain some party-dependent tags that have values. It is suggested that an authoring tool present these tags to the TPA author and ask that the author either indicate acceptance of these values or supply replacement values.

Each place where party-specific tags exist will be pointed out below in the discussion these areas.

Note that each party to a TPA has some unique properties (at least unique contact name and address and communication addresses). Therefore, a TPA using roles will always have either two distinct role parameters or (if one party is explicitly specified) one role parameter.

When a role parameter is used for a party name, the associated party-dependent tags throughout the TPA should be included. Their values in the prototype TPA can be null strings, blanks, or arbitrary strings. These strings must be replaced by actual values when the role parameter is replaced by an actual party name. Any party-dependent attributes may be given values of null strings ("") or blank strings (" ") to improve readability.

See the appendix "5 Examples of Reusable and Fully Qualified TPAs" for an example of a reusable TPA and the resulting fully qualified TPA.

It should be noted that in principle, the whole TPA might have to be negotiated when specific parties are assigned to roles. Roles will be most useful when it can be expected that most parties who fill the roles can be expected to agree to most or all details of the TPA.

> NOTE: Using this method of associating role names with role players permits use of a tool for instantiating an actual TPA from a TPA template by using the associations under `<Role>` to automatically replace each role name by the player name. In general, such a tool would be an interactive tool which would request values for each of the party-specific tags associated with a role name everywhere in the TPA. An authoring tool could provide this capability.
>
> NOTE: In addition to having a TPA author manually fill in the party-specific information, this process could be automated by providing a database containing the necessary information about all parties likely to participate in copies of a TPA generated from a particular template. An authoring tool could use the database to automatically fill in the party-specific information throughout the TPA once the author has identified the parties that replace the role parameters. In principle, the party-dependent tags need not be provided in the template at all since the authoring tool could provide them when producing a TPA for a specific set of parties from the TPA. However, including the party-specific tags (even though empty) aids readability of the XML document and also enables the XML document to be used to generate future specific TPAs.

2.4.4 Participants

The `<Participants>` section of the TPA describes all of the parameters needed for person-to-person communication between the persons at each party to the TPA who are responsible for the TPA (e.g. mailing address, email address). It should be noted that the email address is only for person to person communications. Specific forms of address are defined in the communication section for each of the access protocols. There is a `<Member>` tag for each party to the TPA.

> NOTE: Typically, the values of all tags under `<Participants>` might be copied into a database for use by the application, other programs, and people and are not further processed by the framework. There are thus no format requirements imposed by the TPA except as noted below.

The complete syntax is in 2.4.4.1, "Syntax of Participants Section". Most of the tag names are self-explanatory and will not be discussed here. If no specific rules are given for the value of a tag, any alphanumeric string is acceptable from an XML viewpoint.

The <Member> tag provides the details about each party. It has two attributes, which are required. The value of the MemberId attribute is an alphanumeric string that provides an identifier for the member of the type defined by the IdCodeType attribute.

The value of the IdCodeType attribute defines the kind of member identifier given by the MemberId attribute. IdCodeType="01" denotes a Dun and Bradstreet D-U-N-S number. IdCodeType="ZZ" denotes any mutually agreed identifier. In other words, for IdCodeType="ZZ", the identifiers are defined by the parties for the purpose of the TPA but do not come from any industry-standard registry.

NOTE: Additional ID code types will be defined as needed for future applications.

The <PartyName> tag under the <Member> tag identifies each party to the TPA by its usual organization name. For example, in an application in which a travel agent provides air reservations for customers, the <PartyName> tag identifies the travel agent and airline. In other places in the TPA which refer to parties, the <OrgName> tag is used. It contains an ID reference attribute pointing to the appropriate <PartyName> tag. See 2.2.4, "Party Names and Reference Attributes".

The value of the <PartyName> tag in a prototype TPA may be a role parameter. When the role parameter is replaced by an actual name, all of associated tags (in the tree below <Member>) must be given actual values. In addition, the attributes of <Member> must be given appropriate values.

One or more <Address> tags may be provided under <Member>. For documentation convenience, the type of address may be denoted by <AddressType>. For example, a value of billing denotes a billing address; a value of shipping denotes a shipping address.

NOTE: The values of <AddressType> are not defined by the standard.

The <Country> tag is required.

NOTE: It is the responsibility of each party's mail addressing software to determine whether to include the country in the address in a given piece of mail.

NOTE: <Contact> should be used to define the people who are responsible for administering the TPA, not necessarily people who are involved with specific transactions such as particular purchase orders. The latter names could change from transaction to transaction and are matters for the business application. Those names will generally be exchanged in the messages for each transaction.

NOTE: It should be understood that information in <Participants> could change

within the lifetime of the TPA. Such changes would not necessarily be propagated to the original TPA.

The attribute `Type` is used with `<Contact>`, `<ContactTelephone>`, and `<EMail>`. `Type="primary"` denotes the primary contact or primary telephone or email address for this contact. `Type="secondary"` denotes the alternate contact or the secondary telephone or email address for this contact. If `Type` is omitted, `"primary"` is assumed.

NOTE: If a party considers any information under `<Participants>` as sensitive, both parties should provide adequate protection for the TPA document.

2.4.4.1 Syntax of Participants Section

The syntax is:

```
<Participants>
     <Member MemberId=identifer IdCodeType=code> <!--1 or more-->
          <PartyName Partyname=name>name
               </PartyName>  <!--name of organization-->
               <!--The value of the Partyname attribute is the ID
                    by which the OrgName tag references this
                    party.-->
          <CompanyTelephone>telephone_number</CompanyTelephone>
          <Address>
               <AddressType>type</AddressType>
                    <!--AddressType is shipping, billing, or
                         location-->
               <AddressLine>address_line1</AddressLine>
               <AddressLine>address_line2</AddressLine>
                    <!--as many address lines as needed-->
               <City>city_name</City>
               <State>state_name</State>
                    <!--or province or similar division-->
               <Zip>zipcode</Zip>
                    <!--or similar postal code-->
               <Country>country_name</Country>
               </Address>
          <Contact Type=type>  <!--one or more-->
               <!--Type is "primary" or "secondary",
                    default is "primary"-->
               <LastName>name</LastName>
               <FirstName>name</FirstName>
               <MiddleName>name</MiddleName>
               <Title>title</Title>
               <ContactTelephone Type=type> <!--1 or more-->
                    phone_no</ContactTelephone>
                    <!--Type values same as for <Contact>-->
               <EMail Type=type>email_address</EMail>
                    <!--1 or more-->
                    <!--Type values same as for <Contact>-->
               <Fax>fax_number</Fax>   <!--optional-->
```

```
            </Contact>
        </Member>
        <!--Additional Member definitions-->
    </Participants
```

2.4.5 Arbitration of TPA Disputes

The TPA provides for arbitration of TPA disputes by an outside party. Examples of issues that might require arbitration are timeouts, action sequence errors, and rejection of an action invocation for reasons other than normal application errors. An "unable to comply" response might require arbitration whereas a "no room at the hotel" response does not require arbitration.

When a condition that requires arbitration arises, the parties communicate with the arbitrator and provide information that the arbitrator can use to determine fault and assess penalties if appropriate.

> NOTE: The specific interactions between each party and the arbitrator are outside the scope of the TPA between the parties. Those interactions could be formalized by a separate TPA between each party and the arbitrator. In such a TPA, the party that performs the arbitration function is defined by <Member>, not by <Arbitrator>.
>
> NOTE: Among the information that might be provided to the arbitrator is copies of each party's logs for the conversations involved in the dispute.

Each arbitrator is responsible for being able to communicate with each party and for its own availability regarding both arbitrator server failures and communication failures.

Designation of an arbitrator is optional. In order to use the services of an arbitrator, the arbitrator definition must be included in the TPA. If an arbitrator is specified, both parties must agree on a single arbitrator. This arbitrator is named by the <Arbitrator> tag.

The value of <Arbitrator> in a prototype TPA may be a role parameter.

The arbitrator syntax is:

```
<Arbitrator MemberId=identifer IdCodeType=code>
    arbitrator_name</Arbitrator>
```

2.4.6 TPA Lifetime

The <Duration> tag under <TPAInfo> specifies the valid duration of the TPA. <Duration> consists of <Start>, to specify the starting date and time of the TPA, and <End>, to specify the last date and time on which the TPA is valid. If <Start> is omitted, the TPA is valid immediately. If <End> is omitted, there is no limit to the valid duration. <Duration> can be omitted if the defaults for both <Start> and <End> are acceptable. <Start> and <End> each contain <Date> and <Time> to specify the date and the time on that day. See 2.2.9.2, "Dates" and 2.2.9.3, "Time of Day".

When the end date and time are reached, any actions that are still in progress shall be allowed to complete and no new action requests shall be accepted. When all in-progress actions on each conversation are completed, the conversation shall be terminated whether or not it was completed.

> NOTE: It should be understood that if an application recognizes a "unit of business" consisting of multiple action requests, such units of business may be terminated with no error indication when the end date and time are reached. If the framework also recognizes this unit of business as a conversation, it could provide an error indication to the application. If the framework does not recognize this unit of business as a conversation, it is the responsibility of the application to recognize the situation and recover.
>
> NOTE: It should be understood that it may not be feasible to wait for outstanding conversations to terminate before ending the TPA since there is no limit on how long a conversation may last.

Following is the syntax of `<Duration>.`

```
<Duration>
     <Start>  <!--Optional-->
          <Date>date</Date>
          <Time>time</Time>
          </Start>
     <End>
          <Date>date</Date>
          <Time>time</Time>
          </End>
     </Duration>
```

2.4.7 Invocation Limit

The `<InvocationLimit>` tag specifies the total number of times the TPA can be instantiated (conversations created) before having to renegotiate it. If `<InvocationLimit>` is not stated, there is no limit to the number of invocations. The value of this tag is a positive numeric string.

2.4.8 Concurrent Conversations

The `<ConcurrentConversations>` tag defines the maximum number of conversations under this TPA that may be open at any time. If this tag is omitted, there is no defined limit. The value of this tag is a positive numeric string.

> NOTE: a given server may have an implementation-based limit on the number of concurrent conversations and may reject a new conversation if this limit would be exceeded.

2.4.9 Conversation Instantiation, Closure, and Lifetime

A conversation represents a unit of business defined by the business application. Therefore, the business application determines when the conversation is instantiated and closed. From the viewpoint of a TPA between party1 and party2, instantiation takes place at party1 when party1 sends the first action request to party2. At party2, the conversation is instantiated when it receives the first request of the unit of business from party1.

Closure of a conversation takes place when the parties have completed the unit of business. The application may use the service-interface tag `<TerminateConversation>` to indicate one or more actions whose completion terminates the conversation. See 2.9.6, "Conversation Termination".

> NOTE: An implementation should provide means for the application to signal that a conversation is ended in addition to supporting `<TerminateConversation>`. This signal could be a user exit or explicit API call.
>
> NOTE: It is permissible to define an indefinite-length conversation that encompasses multiple units of business as understood by the business application. Doing so prevents the framework from providing services to help manage the units of business. In general, this technique should be limited to support of legacy applications that cannot indicate the beginning and end of a unit of business to the framework.
>
> NOTE: The framework may provide an interface by which the business application requests instantiation and closure of conversations.

To avoid tying up system resources indefinitely, a conversation should have a maximum lifetime defined in the TPA. The maximum lifetime of a conversation is defined by the `<ConversationLife>` tag under `<TPAInfo>`. The purpose of this tag is to detect "hung" conversations and perform error processing. Normally a conversation would end well before this time expires. The framework should detect expiration of this time and inform the application. The error processing must be performed by the application, applying suitable business rules depending on the state of the conversation. For example, money may have already changed hands. If the `<ConversationLife>` tag is omitted, there is no defined maximum lifetime. The value of `<ConversationLife>` is an integer number of seconds.

Note that this section deals specifically with a single instantiation of the TPA (i.e. a single long-running conversation). Premature termination of the whole TPA is a legal matter.

2.4.10 Preamble Syntax

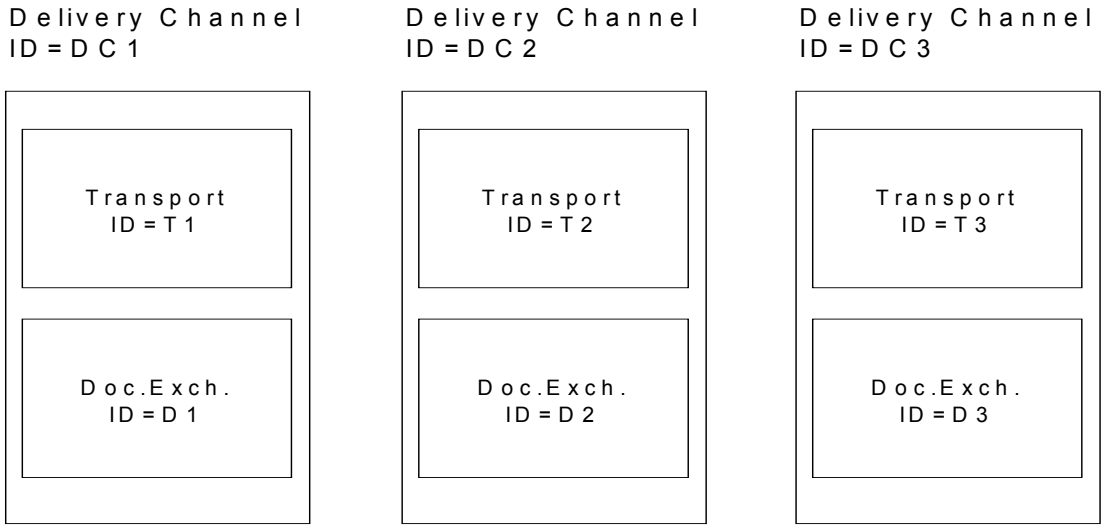
```
<TPAInfo>
     <TPAName>TPA name</TPAName>
          <!--name of TPA-->
     <TPAType>
          ...
          </TPAType>
     <Role>
```

```
     ...
     </Role>
<Participants>
     ...
     </Participants>
<Arbitrator>  <!--optional-->
     ...
     </Arbitrator>
<Duration>     <!--valid duration for TPA-->
     ...
     </Duration>
<InvocationLimit>number</InvocationLimit>
     <!--number of instantiations permitted before
          renegotiation is required-->
<ConcurrentConversations>number</ConcurrentConversations>
     <!--Maximum number of concurrent conversations
          permitted-->
<ConversationLife>time</ConversationLife>
     <!--maximum lifetime of a single conversation-->
</TPAInfo>
```

2.5 Delivery Channels

A delivery channel is a combination of a transport layer definition and a document-exchange definition that specifies how messages from one party's business-protocol layer are moved to the other party's business-protocol layer. The `<DeliveryChannelSet>` tag contains one or more delivery-channel definitions.

A TPA includes one or more delivery-channel definitions, one or more `<Transport>` sections, and one or more `<DocExchange>` sections. The following figure illustrates three delivery channels.



The delivery channels are named DC1, DC2, and DC3. Each delivery channel contains one

transport definition and one document-exchange definition. Each transport definition and each document-exchange definition also has a name as shown.

A specific delivery channel may be associated with each action definition. For an action at one party that is used as the response to an action request at the other party, either the same or different delivery channels may be specified for the two actions.

Multiple delivery channels may include transport sections that use the same or different transport and security protocols. They may differ in, for example, just the communication addresses or in more substantial ways. Similarly, the document-exchange sections of different delivery channels may differ as to any characteristics.

Following is the delivery channel syntax.

```
<DeliveryChannelSet>
     <!--one or more DeliveryChannel tags-->
     <!-- see text for explanation of Channel attributes.-->
     <DeliveryChannel
          ChannelId=name
          TransportId=name
          DocExchangeId=name>
          <Version>version</Version>  <!--Optional-->
          </DeliveryChannel>
     </DeliveryChannelSet>
```

Each `<DeliveryChannel>` tag identifies one `<Transport>` tag and one `<DocExchange>` tag that make up a single delivery channel definition. The same `<Transport>` or `<DocExchange>` tag may be used in more than one delivery channel. Each `<DeliveryChannel>` tag has a name (`ChannelID` ID attribute) by which it can be referenced from an action definition. Attributes are also used to identify the `<Transport>` and `<DocExchange>` elements used by the delivery channel.
The optional `<Version>` tag identifies the specific version of this delivery channel, i.e. the particular combination of `<Transport>` and `<DocExchange>` and the definitions within them. The value of `<Version>` is an alphanumeric character string.

Each action may be statically bound to one delivery channel. The optional `<Channel>` tag may be included in the action definition to define the binding. Within `<Channel>`, the `ChannelId` attribute identifies the desired channel. The value of `ChannelId` must be the same as the value of the `ChannelId` attribute in the desired `<DeliveryChannel>` tag. If an action definition does not contain `<Channel>` and more than one delivery channel is defined in the TPA, the delivery channel for that action is selected dynamically by the framework.

The parties should ensure that only channels compatible with their business application processes are used. For example, if the business application process assumes guaranteed delivery, only channels with guaranteed delivery should be included in the TPA.

> NOTE: The intention for dynamic channel selection is that the delivery components of the framework select a channel for each message based on current conditions such as the

channel being available, not congested, etc.

The values of the attributes of <DeliveryChannel> are arbitrary. Each must be unique within the TPA.

- The value of ChannelId is the name by which the desired channel definition is referred to by the <Channel> tag in the action definition. In the <Channel> tag, the value of the ChannelId attribute must be the same as the value of ChannelId in the desired channel definition.

- The value of TransportId identifies the <Transport> tag to be included in the channel. This value must be the same as the value of the TransportId attribute in the desired <Transport> tag.

- The value of DocExchangeId identifies the <DocExchange> tag to be included in the channel. This value must be the same as the value of the DocExchangeId attribute in the desired <DocExchange> tag.

2.6 Transport

The transport section of the TPA defines communication protocol, encoding, and transport security information. Specific forms of address are defined for each of the access protocols.

If multiple delivery channels are defined in a TPA, one or more transport sections may be defined. The TransportId attribute of <Transport> defines a unique identifier for each transport section, which may be referred to in <DeliveryChannel> tags. See 2.5, "Delivery Channels".

The overall structure of the transport section is:

```
<Transport TransportId = name>  <!--one or more-->
     <Communication>
          …
          </Communication>
     <TransportSecurity>
          …
          </TransportSecurity>
     </Transport>
```

2.6.1 Communication Protocol and Version

Permissible communication protocols are HTTP, Queued Messages, SMTP, VAN-EDI, and FTP. Only one protocol can be specified in a given TPA. The optional <Version> tag identifies a specific version of the protocol. The value of this tag is a character string.

2.6.2 Communication Addressing

The <...Node> tag identifies each party and its communication addressing information. The name of the node tag is specific to each protocol, e.g. <HTTPNode>. There is one <...Node> tag for each party. It identifies the party name by means of the <OrgName> tag, which contains an ID reference attribute pointing to the corresponding <PartyName> tag under <Participants>. All parties must be represented in <Communication>. Under the <...Node> tag is also the unique electronic address of the party (typically an Internet address or URL). This address is contacted for creating an information-delivery channel. The form of address can be different for each communication protocol and is shown in the communication syntax below.

As shown in the syntax, there is a unique communication address tag for each protocol.

For each party which is represented by a role parameter in a prototype TPA, the communication addressing information must be supplied when the role parameter is replaced by an actual party name. This information is the values of the associated tags (under the same <...Node> tag under which the corresponding <OrgName> tag is located).

2.6.3 Transport Encoding

<TransportEncoding> specifies how the transport layer encodes messages for transmission. Possible values of this tag are MIME (see ref. 7 in 1.1, "Normative References"), BASE64 (see ref. 7 in 1.1, "Normative References"), and HTMLEncoding.

The syntax is:

```
<TransportEncoding>encoding</TransportEncoding>
```

The default value for encoding depends on the transport protocol. If the transport protocol performs encoding with no options, <TransportEncoding> may be omitted. Otherwise, the default value is no encoding. If the value of <TransportEncoding> is not consistent with the transport protocol, it shall be ignored.

> NOTE: For proper encoding behavior in the transport level, the characteristics of the data which determine the encoding requirements should be passed from the business application process to the document-exchange level. The document-exchange level should pass this information to the transport level along with an indication of whether the document-exchange level has performed encoding.
>
> NOTE: <TransportEncoding> is under <Communication> since the encoding choices may be transport-protocol-dependent.

2.6.4 Transport Timeout

<TransportTimeout> defines the maximum waiting time for a transport-level acknowledgment of a message and the permitted number of retries. <Timeout> defines the

waiting time. Its value is a numeric integer character string specifying a number of seconds. <Retries> specifies the number of retries allowed before considering that the network as failed. Its value is a numeric integer character string specifying the number of retries. <RetryInterval> specifies the minimum number of seconds from the expiration of the timeout until the retry is initiated.

> NOTE: The purpose of <RetryInterval> is to improve the likelihood of a successful retry by delaying the retry until any temporary condition that might have caused the timeout is likely to have been corrected.
>
> NOTE: Since the appropriate timeout depends on the transport protocol in use, <TransportTimeout> is under <Communication>.

<TransportTimeout> is optional. If it is not specified, the timeout, number of retries, and retry interval are local matters at each party.

The syntax is:

```
<TransportTimeout>  <!--optional-->
     <Timeout>time</Timeout>
     <Retries>number</Retries>
     <RetryInterval>time</RetryInterval>
     </TransportTimeout>
```

2.6.5 Network Delay

The <NetworkDelay> tag defines the expected worst-case round trip network delay for any message and its response. This delay excludes service time at the recipient of a message prior to its sending a response. Other tags in other parts of the TPA define the service time. The value of the <NetworkDelay> tag must be agreed to by both parties. As explained elsewhere, timeouts for receipt of action responses must include both the network delay and the appropriate service time value. If <NetworkDelay> is omitted, there is no defined network delay and the choice of network delay is a local option.

> NOTE: The appropriate value of <NetworkDelay> may depend on both the chosen communication protocol and the network topology.
>
> NOTE: The worst-case value of <NetworkDelay> includes all the transport-related time involved in sending a request and receiving a response. Setting this value should take into account transmission times for the request and response messages, the time from the transmission of each message to the receipt of the transport-layer acknowledgment, and the maximum number of transport-layer retries. The transmission time for a single message is the time from when the message is passed by the sender's document-exchange function to the transport function until it is received by the recipient's document-exchange function.

2.6.6 Guaranteed Delivery

NOTE: It is recommended that each received message be hardened before returning the transport-layer acknowledgment to the sender of the message. This assures guaranteed delivery and is essential for failure recovery.

2.6.7 Communication Syntax

Following is the syntax for the communication section of the TPA along with discussion specific to each protocol.

2.6.7.1 HTTP

HTTP is Hypertext Transfer Protocol (see ref. 4 in 1.1, "Normative References"). For HTTP, the address is a URL. Depending on the application, there may be one or more URLs, whose use is determined by the application. Each URL tag provides one URL.

Following are the tags that specify URLs. Each provides one URL. Either zero or one of each of these tags may be specified.

`<LogOnURL>` identifies the URL used for the initial connection between client and server.
`<RequestURL>` identifies the URL used for receiving requests.
`<ResponseURL>` identifies the URL used for receiving asynchronous response messages.

The URLs provided by`<RequestURL>` and `<ResponseURL>` may be dynamically overridden for a particular request or asynchronous response by application-specified URLs such as URLs in business documents exchanged under the TPA.

For a synchronous response, `<ResponseURL>` is ignored if present. A synchronous response is always returned on the existing connection, i.e. to the URL which is identified as the source of the connection.

For example:

```
<HTTPAddress>
     <LogOnURL>https://www.a.xxx.com</LogonURL>
     <RequestURL>https://www.b.xxx.com</RequestURL>
     <ResponseURL>https://www.c.xxx.com</ResponseURL>
     </HTTPAddress>
```

NOTE: A response to an action is an action request to the party that sent the request message. However, it is a response and its destination is given by `<ResponseURL>`. If the same action request can be issued as either a new request or a response to a prior action, the URL given by `<RequestURL>` is used for a new request and the URL given by `<ResponseURL>` is used for a response to a prior request.

NOTE: If more than one URL of either of these URL types is needed by the same party to the same TPA, additional delivery channels can be defined for the additional URLs.

Following is the syntax for HTTP.

```
<Communication>
   <HTTP>
     <Version>version</Version>
     <HTTPNode> <!--One for each party-->
          <OrgName Partyname=name/>
          <HTTPAddress> <!--each xxxURL tag is optional-->
               <LogonURL>http://www/a.xxx.com</LogonURL>
               <RequestURL>http://www.b.xxx.com</RequestURL>
               <ResponseURL>http://www.c.xxx.com</ResponseURL>
               </HTTPAddress>
          </HTTPNode>
     <TransportEncoding>encode</TransportEncoding>
     <TransportTimeout>  <!--optional-->
          <Timeout>time</Timeout>
          <Retries>number</Retries>
          <RetryInterval>time</RetryInterval>
          </TransportTimeout>
     <NetworkDelay>time</NetworkDelay>  <!--Optional-->
     </HTTP>
   </Communication>
```

2.6.7.2 Queued Messages

TBD

2.6.7.3 SMTP

SMTP is Simple Mail Transfer Protocol (see ref. 8 in 1.1, "Normative References"). For use with this standard, Multipurpose Internet Mail Extensions (MIME) must be supported. The MIME media type used by the SMTP transport layer is `Application` with a sub-type of `octet-stream`.

For SMTP the communication address (`<SMTPAddress>`) is the fully qualified mail address of the destination party as defined by RFC 822. See ref. 5 in 1.1, "Normative References".

SMTP with MIME automatically encodes or decodes the document as required, on a link by link basis, and presents the decoded document to the destination document-exchange function. If the application design is such that the choices in `<DocumentExchange>` and `<BusinessProtocol>` are intended to be independent of the choice of transport protocol, it is permissible to specify `<MessageEncoding>`.

> NOTE: The SMTP mail transfer agent encodes binary (i.e. data which are not7-bit ASCII) data unless it is aware that the upper level (mail user agent) has already encoded the data. If the data are encoded in the document-exchange level (`<MessageEncoding>`), the information that the data are already encoded should be passed to the mail user agent.

Since SMTP is multi-hop and asynchronous, the value of <NetworkDelay> may need to be longer than it would be for other transport protocols over the same network.

NOTE: SMTP by itself (without any authentication or encryption) is subject to denial of service and masquerading by unknown parties. It is strongly suggested that those partners who choose SMTP as their transport layer also choose a suitable means of encryption and authentication either in the document-exchange layer or in the transport layer (S/MIME).

NOTE: SMTP is an asynchronous protocol that does not guarantee a particular quality of service. A transport-layer acknowledgment (i.e. an SMTP acknowledgment) to the receipt of a mail message constitutes an assertion on the part of the SMTP server that it knows how to deliver the mail message and will attempt to do so at some point in the future. However, the message is not hardened and may never be delivered to the recipient. Furthermore, the sender will see a transport-layer acknowledgment only from the nearest node. If the message passes through intermediate nodes, SMTP does not provide an end to end acknowledgment. Therefore receipt of an SMTP acknowledgement does not guarantee that the message will be delivered to the application and failure to receive an SMTP acknowledgment is not evidence that the message was not delivered.

NOTE: In order to ensure that service timeouts as defined in <ServerServiceTime> and <ResponseServiceTime> can be distinguished from delivery failures by SMTP (which should be retried by the framework), it is strongly recommended that the framework implement an end-to-end acknowledgment. One possible implementation is to provide an end to end email message from the destination mail handler to the source mail handler which carries an acknowledgment of a receipt of a specific message and is sent after the message has been hardened. Failure to receive the acknowledgment message within the expected time would result in a retry of the original message. Fields in the email header can be used to carry the necessary message identifiers. One field in the email header of the original message can carry an email address associated with the sending mail handler itself. This would be the address to which the acknowledgment message would be directed.

Following is the syntax of SMTP.

```
<Communication>
   <SMTP>
     <Version>version</Version>
     <SMTPNode> <!--One for each party-->
          <OrgName Partyname=name/>
          <SMTPAddress>address</SMTPAddress>
               <!--party's email address-->
          </SMTPNode>
     <TransportEncoding>encode</TransportEncoding>
     <TransportTimeout>  <!--optional-->
```

```
          <Timeout>time</Timeout>
          <Retries>number</Retries>
          <RetryInterval>time</RetryInterval>
          </TransportTimeout>
      <NetworkDelay>time</NetworkDelay>
      </SMTP>
   </Communication>
```

2.6.7.4 VAN-EDI

The name VAN-EDI refers to EDI as transported by a value-added network.

For VAN-EDI, the communication address (<VANEDIAddress>) defines the names of two mailboxes, one, <InBox>, for incoming and one, <OutBox>, for outgoing messages. The values of these two tags are strings whose contents represent the paths to the data in whatever form is appropriate for each party. The names are case sensitive since they may represent file names.

Following is the syntax for VAN-EDI.

```
<Communication>
   <VANEDI>
     <Version>version</Version>
     <VANEDINode> <!--One for each party-->
          <OrgName Partyname=name/>
          <VANEDIAddress>
               <InBox>name</InBox> <!--mailbox name-->
               <OutBox>name</OutBox> <!--mailbox name-->
               </VANEDIAddress>
          </VANEDINode>
     <TransportEncoding>encode</TransportEncoding>
     <TransportTimeout>  <!--optional-->
          <Timeout>time</Timeout>
          <Retries>number</Retries>
          <RetryInterval>time</RetryInterval>
          </TransportTimeout>
     <NetworkDelay>time</NetworkDelay>  <!--Optional-->
     </VANEDI>
   </Communication>
```

2.6.7.5 FTP

FTP is File Transfer Protocol (see ref. 9 in 1.1, "Normative References").

The following FTP properties must be specified for each party:

- Input directory path (for PUTs to this party)
- Output directory path (for GETs to this party)

  The input and output directory paths must be expressed as FTP URLs. These URLs also include the domain name or IP address of the FTP server.

- Transfer type (binary or ASCII).

  The specified transfer type may be overridden by information about the characteristics of the document that is passed to the transport layer.

- Passive mode (yes or no)
- Control port number (needed for passive mode)

Following is the syntax for FTP:

```
<Communication>
     <FTP>
          <Version>version</Version>
          <FTPNode>  <!--one for each party-->
               <OrgName Partyname=name/>
               <FTPAddress>
                    <InDirectory>path</Indirectory>
                         <!--for puts-->
                    </OutDirectory>path</OutDirectory>
                         <!--for gets-->
                    </FTPAddress>
               <TransferType>type</TransferType>
                    <!--type is ascii or binary -->
               <PassiveMode>choice</PassiveMode-->
                    <!--choice is yes or no-->
               <ControlPort>number <ControlPort>
                    <!--For use with passive mode-->
               </FTPNode>
          <TransportEncoding>encode</TransportEncoding>
          <TransportTimeout>  <!--optional-->
               <Timeout>time</Timeout>
               <Retries>number</Retries>
               <RetryInterval>time</RetryInterval>
               </TransportTimeout>
          <NetworkDelay>time</NetworkDelay>
          </FTP>
     </Communication>
```

2.6.8 Transport Security

The `<TransportSecurity>` tag provides the security specifications for the transport layer of the TPA. It may be omitted if transport security will not be used for this TPA. Unless otherwise specified below, transport security applies to messages in both directions.

For each party which is represented by a role parameter in a prototype TPA, the corresponding tags under `<Party>` and `<LogonParty>` must be given values when the role parameter is replaced by an actual name.

Transport security may be either encryption or authentication. Both encryption and authentication may be used in the same TPA, especially if authentication is by userid and

password. Authentication verifies the sender of the message. Encryption prevents the message from being read by unauthorized parties but by itself performs no authentication or nonrepudiation

For encryption, the protocol and protocol version (optional) must be supplied by the appropriate tags. Examples of encryption protocols are `RC2, RC5, RC6, and S/MIME`. Only certificate-based encryption is defined.

For authentication, the authentication type must be specified with the `<PasswordAuthentication>` or `<CertificateAuthentication>` tag. Password and certificate authentication may be used together. The protocol, protocol version (optional), and certificate parameters or optional starting userid and password must be specified. Examples of protocols are SSL and S/MIME.A public key certificate must be supplied for each party (except as specified below).

Authentication is bi-directional if each party has to authenticate to the other during a given message exchange. The alternative is that the client authenticates to the server but not *vice-versa.* In most cases, the choice is determined by other factors. If SSL Ver. 3 is specified for encryption, authentication is always bi-directional. If the authentication type is `PASSWORD`, the client authenticates to the server. At the same time, the server may use a certificate to authenticate to the client. If the authentication type is `CERTIFICATE`, the authentication could be bi-directional or not. For the purpose of this TPA, it is defined as bi-directional unless otherwise specified below.

It should be understood that in a TPA in which each party can act as either a server or a client for different actions, the security definitions must enable each party to authenticate to the other, though not necessarily in the same message exchange.

The `<Certificate>` tag provides the parameters needed to define the certificates. The `<CertType>` tag identifies the type of certificate. Examples are `X.509V1, X.509V2, and X.509V3`. The `<KeyLength>` tag gives the key length in bits (e.g. `512`). Both parties must agree on the certificate type and key length. However different type and length may be specified for authentication, nonrepudiation, and digital envelope.

To support certificate authentication, the TPA must reference the public-key certificate for each party. The `<Party>` tag defines the party-specific parameters. There must be one `<Party>` subtree for each party. `<OrgName>` contains an ID reference attribute that points to the corresponding `<PartyName>` tag.

For certificate authentication, the `<IssuerOrgName>` tag under `<Party>` identifies the certificate issuer, a recognized certificate authority. The value of `<IssuerOrgName>` must exactly match what is on the certificate, including case. The `<IssuerCertSource>` tag (under the `<Party>` tag) provides a URL from which the certificate issuer's certificate can be obtained during initialization of communication between parties. This URL is associated with the organization named in the `<IssuerOrgName>` tag. There is no requirement that all parties use the same certification authority.

NOTE: Because the public-key infrastructure is still evolving, support for <IssuerOrgName> and <IssuerCertSource> is to some extent still to be determined. Parties to a TPA may have their own ways to obtain each other's certificates that are outside the scope of the TPA. A party may in any case use <IssuerOrgName> to validate the issuer of a certificate.

Password authentication includes the option of specifying a userid and password for use the first time each party invokes an action against the other. The <LogonInfo> tag is used for this purpose. The initial values specified for each party are those that the other party uses to log on.

NOTE: Specifying initial values for userid and password in a TPA introduces a degree of security risk. If initial values are specified, the parties should take precautions. Both parties should keep the TPA document under security controls. Each party should change the password at the initial contact with the other party. Each TPA should specify a unique set of initial values (i.e. not reuse previously-used values). These precautions keep to a minimum the ability of an unauthorized party to make use of these values.

The security mode used for the communication that initializes a conversation depends on which security modes are defined in the <TransportSecurity> section. If authentication is defined, authentication is used. Otherwise, encryption is used if defined. If neither authentication nor encryption is defined, no security is applied to the initial communication.

2.6.8.1 Specifics for HTTP

For encryption with HTTP, the protocol is SSL (Secure Socket Layer) Version 2.0 or 3.0, which uses public-key encryption (see ref. 10 in 1.1, "Normative References"). The encryption tags include the certificate parameters.

Authentication of the client to the server may be either by password or by certificate. Certificate authentication is SSL version 3.0. If each party may act as a server at times and a client at other times, the TPA must specify certificates for both parties.

The password algorithm is HTTPAuthentication. The server presents a certificate to the client that authenticates the server. The client uses that certificate to encrypt the password. If each party may act as a client at times and as a server at other times, the TPA must specify password authentication and certificate authentication for both parties.

2.6.8.2 Transport Security Syntax

```
<TransportSecurity>
     <Encryption>  <!--Optional-->
          <Protocol>protocol</Protocol>
          <Version>version</Version>
               <!--string denoting version of protocol-->
          <Certificate>
               <CertType>type</CertType>
                         <!--Example:  X.509V1, X.509V2-->
                    <KeyLength>length_in_bits</KeyLength>
                    <Party>
```

```
                    <!--one for each party that can act
                         as a server-->
                    <OrgName Partyname=name/>
                         <!--name of party-->
                    <IssuerOrgName>name</IssuerOrgName>
                         <!--Example:  VeriSign, Inc.-->
                    <IssuerCertSource>url</IssuerCertSource>
                         <!--URL of certificate-->
                    </Party>
               </Certificate>
          </Encryption>
     <Authentication>
          <PasswordAuthentication>
               <Protocol>protocol</Protocol>
                    <!--Example: HTTPAuthentication-->
               <Version>version</Version> <!--optional-->
                    <!--version of protocol-->
               <LogonInfo>  <!--optional-->
                    <LogonParty>
                         <!--one for each party that can act as a
                            server-->
                         <OrgNamePartyname=name/>
                         <UserId>value</UserId>
                         <Password>value</Password>
                              </OrgName>
                         </LogonParty>
                    </LogonInfo>
               </PasswordAuthentication>
          <CertificateAuthentication>  <!--optional-->
               <Protocol>protocol</Protocol>
                    <!--example:  SSL-->
               <Version>version</Version>
               <Certificate>
                    <CertType>type</CertType>
                         <!--Example:  X.509V1, X.509V2-->
                    <KeyLength>length_in_bits</KeyLength>
                    <Party>   <!--one for each party>
                         <OrgName Partyname=name/>
                              <!--name of party-->
                         <IssuerOrgName>name</IssuerOrgName>
                              <!--Example: VeriSign, Inc.-->
                         <IssuerCertSource>url</IssuerCertSource>
                              <!--URL of certificate-->
                         </Party>
                    </Certificate>
               </CertificateAuthentication>
          </Authentication>
     </TransportSecurity>
```

2.7 Document Exchange

The document-exchange section provides information that the parties must agree on regarding exchange of documents between them. This information includes the message security definition.

If multiple delivery channels are defined, one or more document-exchange sections may be defined. The `DocExchangeId` attribute of `<DocExchange>` defines a unique identifier for each document-exchange section, which may be referred to in `<DeliveryChannel>` tags. See 2.5, "Delivery Channels ".

The syntax is:

```
<DocExchange DocExchangeID=name> <!--one or more-->
     <MessageEncoding>encoding</MessageEncoding>
          <!--currently must be BASE64 or omitted-->
     <MessageIdempotency>choice</MessageIdempotency>
          <!--choice is yes or no-->
     <MessageRetries>
          …
          </MessageRetries>
     <MessageSecurity>
          …
          </MessageSecurity>
     </DocExchange>
```

2.7.1 Message Encoding

`<MessageEncoding>` specifies how the messages are encoded by the document-exchange layer for transmission. Currently, BASE64 is the only choice. If the `<MessageEncoding>` tag is omitted, the default is no document-exchange encoding.

Other definitions concerning message formats are provided within the action definition.

2.7.2 Idempotency Test

`<MessageIdempotency>` specifies whether all messages sent under the TPA are subject to an idempotency test (detection and discard of duplicate messages) in the document exchange layer. If the value of the tag is `yes`, all messages are subject to the test. If the value is `no`, messages are not subject to an idempotency test in the document-exchange layer. Testing for duplicates is based on the message identifier that is carried in the message header. See ref. 1 in 1.1, "Normative References".

> NOTE: Additional testing for duplicates may take place in the business application based on application information in the messages (e.g. purchase order number).

The idempotency test checks whether a message duplicates a prior message between the same client and server. If the idempotency test is requested, a duplicate message is passed to the recipient with a "duplicate" indication. A "duplicate" indication is also returned to the sender.

One of the main purposes of this test is to aid in retry following timeouts and in recovery following node failures. In these cases, a request is reissued when it is not known whether the original request was received. If the original request was received, the duplicate is discarded by the server and the server returns the original results to the requester.

If a communication protocol always checks for duplicate messages, the check in the communication protocol overrides any idempotency specifications in the TPA.

2.7.3 Message Retries

<MessageRetries> specifies the permitted number of retries, and interval between retries, of a request following a business-level error condition. This tag is optional. If <MessageRetries> is omitted, the default is no retries. An error condition is either a server-level timeout (<ResponseServiceTime> or <ServerServiceTime>) or is indicated in a response (usually exception response) message. Exhaustion of retries of a timeout may indicate either a failure condition or a possible TPA violation. It is the responsibility of the application to determine how to proceed following exhaustion of the permitted number of retries.

> NOTE: Errors indicated in a response message generally indicate some kind of document content or format error that must be corrected by application-program action or by correction of a software problem before a retry is worthwhile. Such an error could also be caused by a data entry error and this might be correctable and retriable if the retry interval (see below) is long enough.

When a server-level timeout or <ExceptionResponse> message occurs, the action is terminated. A retry is a new instance of the action request. If the action defines more than one response message and the error condition occurs after one or more response messages have been issued, the actions is nonetheless considered failed and a retry is still a new instance of the action request.

The syntax of <MessageRetries> is:

```
<MessageRetries>
      <Retries>number</Retries>
      <RetryInterval>time</RetryInterval>
      </MessageRetries>
```

The value of <Retries> is the maximum number of retries permitted before the action is considered failed. The value of <RetryInterval> is the minimum number of seconds between retries of actions. The purpose of <RetryInterval> is to improve the likelihood of success on retry be deferring the retry until temporary conditions that caused the error might correct themselves.

> NOTE: In general, <RetryInterval> is intended to limit the retry frequency for conditions indicated by <ExceptionResponse> messages. While it is required for retries of all conditions, in general, server-level timeouts are much longer than typical values of <RetryInterval>, so that these timeouts can be retried immediately.

The sole purpose of the <MessageRetries> tag is to record agreement between the two parties on the number of permitted retries and minimum retry interval. The actual retry methodology is an application matter and not defined in the TPA. It may, however, be necessary for the framework to be aware of retries. Some informative notes on this follow.

> NOTE: Once a new instance of retry begins, any messages left in the system from prior instances should not generate error conditions. Messages may be left in the system if it cannot be guaranteed that messages are always received in the order sent. Any remaining timeouts from prior instances should be cancelled.
>
> NOTE: It is strongly recommended that the request message that retries a request have a new and unique message identifier. This prevents the idempotency test from considering the retry message as a duplicate.
>
> NOTE: For applications that do use the previous message identifier for the retry, it is strongly recommended that the current retry count be included in the message header. Frameworks should recognize such protocols based on <TPAType>, <MessageSet>, or content-type information in the message header and perform the idempotency test on the combination of message identifier and current retry count. This is particularly important if the implementation or application protocol is such that messages from prior instances of the action (initial or earlier retries) cannot be purged from the system before the new retry is initiated.

2.7.4 Message Security

The <MessageSecurity> tag provides the security specifications for the document exchange function. It may be omitted if message security will not be used for this TPA. Message security may be either nonrepudiation or digital envelope or both.

Message security applies to all messages in both directions for actions for which message security is enabled. See the discussion of <ServiceSecurity> and <ActionSecurity> below.

For each party which is represented by a role parameter in a prototype TPA, the corresponding tags under <Party> must be given values when the role parameter is replaced by an actual name. This must be done under <NonRepudiation> and <DigitalEnvelope>.

Nonrepudiation both proves who sent a message and prevents later repudiation of the contents of the message. Digital envelope (see ref. 11 in 1.1, "Normative References"). is an encryption procedure in which the message is encrypted by symmetric encryption(shared secret key) and the secret key is sent to the message recipient encrypted with the recipient's public key.

The <NonRepudiation> and <DigitalEnvelope> tags are optional. Each must be supplied only if the particular mode will be used.

For nonrepudiation, the protocol (e.g. DigitalSignature), protocol version (optional), hash function (e.g. SHA1, MD5) encryption algorithm (see above), signature algorithm (e.g. DSA), and certificate type (see above) must be specified. A public key certificate must be supplied for each party (see below).

The <Certificate> tag provides the parameters needed to define the certificates. The <CertType> tag identifies the type of certificate. Examples are X.509V1, X.509V2, and X.509V3. The <KeyLength> tag gives the key length in bits (e.g. 512). All parties must agree on the certificate type and key length. However different type and length may be specified for nonrepudiation and digital envelope.

To support nonrepudiation and digital envelope, the TPA must reference the public-key certificate for each party. Separate certificates can be defined for nonrepudiation and digital envelope. The <Party> tag defines the party-specific parameters. There must be one <Party> subtree for each party. <OrgName> contains an ID reference attribute that points to the corresponding <PartyName> tag. <OrgCertSource> is optional. If present, it provides the URL of a certificate issued by the party itself. <OrgCertSource> can be used only for digital envelope. The <IssuerOrgName> tag under <Party> identifies the certificate issuer, a recognized certificate authority. The value of <IssuerOrgName> must exactly match what is on the certificate, including case. The <IssuerCertSource> tag (under the <Party> tag) provides a URL from which the certificate issuer's certificate can be obtained during initialization of communication between parties. This URL is associated with the organization named in the <IssuerOrgName> tag. There is no requirement that all parties use the same certification authority. The corresponding private key for each party, of course, is private to each party and does not appear in the TPA.

Because the public-key infrastructure is still evolving, support for <OrgCertSource>, <IssuerOrgName>, and <IssuerCertSource> is to some extent still to be determined. Parties to a TPA may have their own ways to obtain each other's certificates that are outside the scope of the TPA. A party may in any case use <IssuerOrgName> to validate the issuer of a certificate.

The use of message security (i.e. nonrepudiation or digital envelope) for exchanges on a conversation is controlled by the <ServiceSecurity> tag in the <ServiceInterface> section and the <ActionSecurity> tag in the action definition. These tags control whether message security is applied to all, some, or no messages. The value of the <ServiceSecurity> and <ActionSecurity> tags is either yes or no. If the value of the <ServiceSecurity> or <ActionSecurity> tag is yes in any service interface or action definition, either or both of the message security protocols must be defined in the <MessageSecurity> section.

If the value of a <ServiceSecurity> tag is yes, then message security is applied to all messages to and from this server unless overridden by an <ActionSecurity> tag in an action definition with a value of no in the same service interface. If the <ServiceSecurity> tag has a value of no or is omitted, message security is not to applied to any messages unless specified in an <ActionSecurity> tag in an action definition in the same service interface.

In an action definition, an <ActionSecurity> tag with a value of yes means that message security will be applied to all messages for this action. A value of no means that message security will not be applied to messages for this action. If the <ActionSecurity> tag is not present in an action definition, the value of the <ServiceSecurity> tag in the same <ServiceInterface> section determines whether or not message security is applied to this action. If message security is not applicable to an action, encryption is used if defined in the <TransportSecurity> section. Otherwise no security is applied.

<DigitalEnvelope> consists of <Protocol>, <Version>, <EncryptionAlgorithm>, and <Certificate>, as previously described.

2.7.4.1 Message Security Syntax

```
<MessageSecurity>
     <NonRepudiation>
          <Protocol>protocol</Protocol>
               <!--example: DigitalSignature-->
          <Version>version</Version>
               <!--version of protocol-->
          <HashFunction>function</HashFunction>
               <!--example:  MD5, SHA1-->
          <EncryptionAlgorithm>algorithm
               </EncryptionAlgorithm>
               <!--Example:  RC2 -->
          <SignatureAlgorithm>algorithm</SignatureAlgorithm>
               <!--Example:  DSA-->
          <Certificate>
               <CertType>type</CertType>
                    <!--Example:  X.509V1, X.509V2-->
               <KeyLength>length_in_bits</KeyLength>
               <Party>   <!--one for each party>
                    <OrgName Partyname=name/>
                         <!--name of party-->
                    <IssuerOrgName>name</IssuerOrgName>
                         <!--Example:  VeriSign, Inc.-->
                    <IssuerCertSource>url</IssuerCertSource>
                         <!--URL of  certificate-->
                    </Party>
               </Certificate>
          </NonRepudiation>
     <DigitalEnvelope>
          <Protocol>DigitalEnvelope</Protocol>
          <Version>version</Version>
          <EncryptionAlgorithm>algor</EncryptionAlgorithm>
               <!--example:  RC4-->
          <Certificate>
               <CertType>type</CertType>
                    <!--Example:  X.509V3-->
               <KeyLength>length_in_bits</KeyLength>
                    <!example:  512-->
               <Party>   <!--one for each party>
```

```
                    <OrgName Partyname=name/>
                         <!--name of party-->
                    <OrgCertSource>url</OrgCertSource>
                         <!--URL of certificate-->
                    <IssuerOrgName>name</IssuerOrgName>
                         <!--Example:  VeriSign, Inc.-->
                    <IssuerCertSource>url</IssuerCertSource>
                         <!--URL of  certificate-->
                    </Party>
               </Certificate>
          </DigitalEnvelope>
     </MessageSecurity>
```

2.8 Business Protocol

The `<BusinessProtocol>` tag defines the section of the TPA that contains all the business-protocol definitions that support the business application. Under `<BusinessProtocol>` is the service interface definition for each party that can act as a server. The syntax is

```
<BusinessProtocol>
     <MessageSet>name</MessageSet>
     <BusinessInterface>
          <ServiceInterface>  <!--one or two-->
               ...
               </ServiceInterface>
          </BusinessInterface>
     </BusinessProtocol>
```

2.8.1 Message Set

The `<MessageSet>` tag identifies the collection of messages exchanged by action requests and responses under this TPA. Its value is a character string that identifies the messages. The value may be either a keyword whose meaning is understood by both parties, e.g. `OBIMessages`, or a URL whose target is accessible to both parties.

> NOTE: Some implementations may use this tag to identify the correct parser or mapper to process the messages or to identify the plug-ins that are needed to support the protocol.

2.8.2 Business Interface

The `<BusinessInterface>` tag brackets the one or two service interfaces (one for each party) which define the business process.

2.9 Service Interface

The `<ServiceInterface>` tag defines the services provided by a party that acts as a server. The services include the list of actions provided by that server, their characteristics, and the

actions permitted initially. The TPA contains a separate service interface definition for each party that acts as a server.

> NOTE: For some applications, each party may have both server and client characteristics, i.e. each party may issue requests to the other party.

The <ServiceInterface> tag has one attribute, InterfaceId. The value of this attribute is an alphanumeric string that contains an identifier for this service interface. Each service interface must have an ID that is unique within the TPA. An example is InterfaceId="SERVER01".

2.9.1 Service Provider

The <OrgName> tag identifies the party whose service interface is defined. This provides an ID reference attribute that points to the corresponding <PartyName> tag.

2.9.2 Task Name

The <TaskName> tag provides a label to identify each service interface. The value is a case-insensitive alphanumeric string. <TaskName> is optional.

2.9.3 Actions

The action menu is discussed following the other elements of the service interface.

2.9.4 Server Service Time

The <ServerServiceTime> tag provides information about the server's expected time (as seen at the server) to complete an action request and send the final reply. This tag gives a default value to be used with action definitions for this server, which do not include the <ResponseServiceTime> tag.

The <ServiceTime> tag gives the expected service time. The value is a character string containing a number of seconds.

<ServerServiceTime> is optional. If it is not present, there are no defined defaults for any of the quantities under <ServerServiceTime>. If this tag is not present, for any action definition that does not include <ResponseServiceTime>, the client's choice of a timeout value is outside the scope of the TPA.

See 2.9.20, "Timing of Action Results" for additional discussion. See 2.9.21, "Presumed Result Following Timeout" for a discussion of the <Presume> tag.

Following is the complete syntax of <ServerServiceTime>.

```
<ServerServiceTime>
     <ServiceTime>time</ServiceTime>
     <Presume>result</Presume>
```

```
        <!--Result is success or fail-->
    </ServerServiceTime>
```

2.9.5 Sequencing Rules

In general, within a single conversation, actions on a given service interface are performed sequentially. One action must not be invoked until the prior action is completed. An exception is actions with `Type="concurrent".` See 2.9.13.1, "Action Type Concurrent".

When the framework supports concurrent conversations, each conversation is independent of the others and the sequencing rules apply separately to each conversation.

Sequencing rules define which actions are allowed and which are not allowed by a given server after a given action on a given conversation. At any point in the conversation, some actions are allowed and others are not allowed. For example, a reservation cannot be canceled before it is requested. In the service interface definition for each provider, but outside the scope of the action menu, the <StartEnabled> tag identifies which actions are permitted as the initial action on that server. If <StartEnabled> is omitted, all actions defined in that service interface are permitted as the initial action.

In each action definition, sequencing rules cause actions to be added to or deleted from the current list of allowed actions. The <Sequencing> tag identifies which actions are allowed and which are not allowed following that action. The <Disable> tag identifies which actions are not permitted after the defined action. For example, a specific TPA may require that a new reservation request cannot be made after a successful reservation request. The <Enable> tag identifies which actions are permitted after the defined action. For example, a reservation can be confirmed, modified, or canceled after a successful reservation request. If the action definition does not include a <Sequencing> tag, the list of permitted actions is not changed. If the <Enable> tag is omitted, no new actions are enabled. If the <Disable> tag is omitted, no additional actions are disabled.

Under <StartEnabled>, <Enable>, and <Disable>, the <RequestName> tag identifies each action which is enabled or disabled.

> NOTE: TPA tools which process these tags should treat the value of <RequestName> as case-insensitive in any comparisons with the values of <RequestName> in action definitions.

The rules in <Sequencing> are effective only if the action succeeded. If the action did not succeed, the lists of actions allowed and not allowed are unchanged.

An application might define a unit of business consisting of multiple actions and perform multiple sequential units of business in the same conversation. If so, the sequencing rules for the final action of a unit of business should restore the permitted and disallowed action lists to the state appropriate for the beginning of a new unit of business.

See 2.9.13, "Action Type" and its subsections for any special considerations for the extended actions.

2.9.6 Conversation Termination

<TerminateConversation> may be used to indicate which action or actions cause the current conversation to be terminated. Termination shall take place at the conclusion of the first such action, when the party that issued the action request receives the final reply message.

Under <TerminateConversation>, the <RequestName> tag identifies each action which can terminate the conversation.

> NOTE: TPA tools which process these tags should treat the value of <RequestName> as case-insensitive in any comparisons with the values of <RequestName> in action definitions. NOTE: If this tag is not used, the application can signal conversation termination to the framework using an exit or API call provided by the framework.

The syntax is

```
<TerminateConversation>  <!--optional>
     <RequestName>action_name</RequestName>
          <!--one for each action defined as ending a
               conversation-->
     </TerminateConversation>
```

2.9.7 Service Security

See 2.7.4, "Message Security" for a discussion of the <ServiceSecurity> tag.

2.9.8 Service Interface Syntax

```
<ServiceInterface InterfaceId=id>
     <!--Example:  InterfaceId="SERVER01">
     <OrgName Partyname=name/>
          <!--The party that acts as a server-->
     <TaskName>name</TaskName>  <!--optional-->
     <ActionMenu>   <!--see "Action Menu"-->
          <Action>  <!--one for each action-->
               ...
               </Action>
          </ActionMenu>
     <ServerServiceTime>  <!--see "Server Service Time"-->
          ...
          </ServerServiceTime>
     <StartEnabled>
          <RequestName>action_name</RequestName>
               <!--one for each action permitted as the initial
                   action-->
          </StartEnabled>
     <TerminateConversation>  <!--optional>
          <RequestName>action_name</RequestName>
               <!--one for each action defined as ending a
```

```
                conversation-->
        </TerminateConversation>
    <ServiceSecurity>option</ServiceSecurity>
        <!-- option is yes or no-->
    </ServiceInterface>
```

2.9.9 Action Menu

The following subsections discuss the elements of the action menu. These discussions are followed by the complete syntax of the action definition.

2.9.10 Action Definition

This section gives a general description of the action. Subsequent sections provide the details of the action definition.

An action is a unit of work defined in the TPA. It consists of a request for service and associated information to be discussed below. An action may be associated with multiple message flows. The maximum expected response time may be specified for each action (see 2.9.20, "Timing of Action Results"). Some actions can be revoked by corresponding cancel actions defined in the TPA. Note that the implementation of each cancel function does not necessarily imply complete undo or rollback; rather the intended use is to compensate for the effects of the associated, previously performed action. An example is canceling a hotel reservation.

Each action is invoked via a request message. During the long execution period, the service provider can send one or more informative messages, followed by a final reply message. A completed action can be canceled if allowed by the specification. An action description also specifies handling of failures.

> NOTE: In reality, an action defined in the TPA is a placeholder for what really goes on at the service provider. The TPA states an action name or equivalent but at the TPA level, it is not known what the action really does. For example, cancel might cancel the reservation, do nothing, or even make a new reservation. Of course there must be some agreement between the parties as to the nature of each action but the details are hidden in the local implementation.

There is a separate set of action definitions within the service interface definition for each party.

2.9.11 Attributes of the Action Tag

The `<Action>` tag has the following attributes. These are discussed in the following subsections.

`ActionId` - The value is a string that is unique for each action in the TPA.

`Type` - Permissible values are:

```
"basic"
"concurrent"
```

The default value is "basic"

Invocation - Permissible values are:
"syncOnly"
"asyncOnly"

The default is "asyncOnly".

2.9.12 Action ID

The ActionId attribute is required. Its value is an alphanumeric string that is unique among all actions defined in the TPA. For example, ActionId="action03".

2.9.13 Action Type

The Type attribute identifies the action as either Type="basic" (performs the function specified for it) or one of several kinds of extended actions in which the framework supplies function in addition to the basic action. The extended action is Type="concurrent".

> NOTE: Additional action types may be defined once the request patterns for a range of applications are understood. One possibility is an action type in which the service provider does not commit the results of a successful action until the client confirms.

If the Type attribute is omitted, the default is "basic", which means that no extended service is defined for this action.

Following are the services that may be applied through extended actions:

2.9.13.1 Action Type Concurrent

Type="concurrent" means that this action may be invoked concurrently with other actions in the same conversation, provided that it is enabled by the sequencing rules.

> NOTE: This attribute concerns only actions in the same conversation. In addition, the framework may support multiple concurrent conversations.

2.9.14 Invocation Attribute

The synchronous/asynchronous properties of an action as seen by the client are specified using the Invocation attribute of the <Action> tag. The following values can be specified:

> Invocation="syncOnly" - The action can only be invoked as synchronous.
> Invocation="asyncOnly" - The action can only be invoked as asynchronous.

If the Invocation attribute is not provided, the default is "asyncOnly".

2.9.15 Synchronous Actions

A synchronous action is an action for which the response is returned on the same transport (e.g. TCP) connection as the request. A synchronous action can be used only with a transport that supports returning the response on the same connection as the request. For purposes of this subsection, such delivery channels are called synchronous delivery channels.

> NOTE: Of the transports supported by tpaML, only HTTP meets this requirement.

If a TPA that includes synchronous actions has only one delivery channel, that delivery channel must be synchronous. If a TPA includes more than one delivery channel and a specific delivery channel is bound to a synchronous action, that delivery channel must be synchronous. If a TPA includes more than one delivery channel but a synchronous action does not have a delivery channel bound to it, the framework shall dynamically select only synchronous delivery channels for the synchronous action.

> NOTE: An authoring tool could enforce the above rules.

> NOTE: An application may choose to emulate synchronous operation by blocking until a response is received. This is a private matter for each party and is not part of the TPA definition.

2.9.16 Asynchronous Actions

An asynchronous action is one for which the request is sent on one transport connection and the response is returned on a new transport connection. Using a new transport connection permits much longer times between receiving a request and sending the response than typical response times for a synchronous transport protocol. Any supported transport protocol may be used with an asynchronous action. When an asynchronous action is used with a synchronous transport protocol such as HTTP, a transport-level acknowledgment (e.g. HTTP 200 OK) terminates the first connection. The response is later transferred on a new connection as an HTTP request, which has its own transport-level acknowledgment that terminates the new connection.

2.9.17 Delivery Channel

The optional `<Channel>` tag under `<Request>` specifies the binding of an action to a specific delivery channel. The `<Channel>` tag has one attribute, `ChannelId`, whose value must be equal to the value of the `ChannelId` attribute in the `<DeliveryChannel>` tag for the desired delivery channel. See 2.5, "Delivery Channels".

2.9.18 Request

The `<Request>` tag within the action definition defines the name of an action, the input information and the result information.

2.9.18.1 Request Name

`<RequestName>` defines the name of the action. Each action under a given service interface must have a name which is unique within that service interface. The value of the tag is a character string.

> NOTE: An implementation may place a restriction on the length of the string.

2.9.18.2 Input Information to Request

The `<RequestMessage>` tag defines the input information for the request. Examples are `EDI840` (EDI message 840) and `OBIPOR` (OBI Purchase Order Request).

The value of `<RequestMessage>` is an alphanumeric string that references information that defines the format of the message. An example of information that might be referenced is a DTD or XML Schema document. The value may be either a keyword whose meaning is understood by both parties or a URL whose destination is accessible to both the server and the client.

> NOTE: It is essential that both parties agree on the format of the message and record this agreement in a way that ensures that both will be using the same information. One way to ensure this is to use the URL of a DTD or other schema description as the value of `<RequestMessage>`. This ensures that both parties are using exactly the same document format description. The same thing can be done for document formats other than XML as long as there is a document that defines the format and can be referred to by `<RequestMessage>.`
>
> NOTE: One possible implementation is that the value of the tag is a keyword that is an entry in a dictionary local to each party. The dictionary entry contains a local file path or URL from which the format definition or the name of a suitable document generator or parser (for a response message) is obtained. The keyword is resolved to the file path or URL when the TPA is registered at each party's site. If the document format is defined by local information at each party, the two parties must reach agreement on the document format and ensure that both maintain the same local information.
>
> NOTE: If the tag value is a reference to a non-local repository, it is suggested that the item be obtained from the repository at TPA registration time or conversation startup time and be stored locally for subsequent use. Whether it is obtained at registration or conversation startup depends on how frequently it is expected to change.

2.9.19 Response to Request

The response information discussed here is returned for both the final result of an action and any intermediate responses. The final result and any intermediate response information are transferred by means of response messages. It should be understood that, depending on the function performed by an action, it may or may not return any result information.

The normal (error-free) response to a request is sent by means of one or more response messages defined under <Response>. <Response> must be omitted if no response information is returned by the action.

Responses for exceptions, such as errors and other unusual conditions, are defined by the optional <ExceptionResponse> tag.

> NOTE: Some application protocols may not distinguish between normal and exception responses except by fields in the response message. In such cases, it is up to the business application process to recognize and deal with the exceptions. An example is the cXML "Order Response" message.

A response message is an action request directed to the party that sent the original request message. The corresponding action must be defined in the action menu of the party that sent the original request. See 2.9.19.1, "Response".

> NOTE: Although responses are defined as separate actions at the requester, an implementation design may have a single message path to the application, with the <RequestName> tag selecting the application function that will process each message. This is true for both synchronous and asynchronous actions.

When an action is asynchronous, all responses to a request are asynchronous messages. When an action is synchronous, the first or only response message is synchronous to the original request message. If a synchronous action has more than one <ResponseName> tag, all responses except the first are asynchronous messages.

> NOTE: If <Response> is omitted from an action definition, it is recommended that both the requester and the recipient of the request notify the framework when processing of the request is complete. Notification could be by software means defined by the framework. Notification enables the framework to maintain any state tracking and management of the sequencing rules.

It is not required that a client or server support asynchronous reply messages. The means used to convey results is a matter of negotiation when the contract is written.

See 2.9.20, "Timing of Action Results" for a discussion of specifying the service time for each of these responses. See 2.9.22, "Error Handling" for more information.

2.9.19.1 Response

<Response> contains one or more <ResponseName> tags. The value of <ResponseName> is the same as the value of <RequestName> in the definition of the action (at the client) that is to process the response message. The value of <ResponseName> is an alphanumeric string.

> NOTE: A TPA tool that matches the value of <ResponseName> to the value of the corresponding <RequestName> tag should treat these values as case-insensitive, i.e.

the matching names should compare equal regardless of case.

NOTE: An implementation might place a restriction on the length of the string.

NOTE: A response message should carry an indicator that it is a response message and the ID of the past request to which it is a response. The response indicator enables the framework to detect that the response was received within the maximum allowed time. An implementation could use the presence of the past request ID as the response indicator.

NOTE: When an action can be performed both as a new action and as a response to a prior action, the state of the response indicator allows the framework and the business application process to distinguish between the two cases.

NOTE: When an action is performed as a response to a prior action, the framework should be made aware when the response action is complete.

`<ResponseName>` has one attribute, `Required`. The value of `Required` is `"yes"` or `"no"`. The default value is `"yes"`. If `Required="yes"`, this particular response message is always issued. If `Required="no"`, the message may or may not be issued, depending on conditions at the time the request is processed.

NOTE: If the last or only response message is defined with `Request="no"`, the application should inform the framework when the action is complete.

For an action that results in multiple result sets, a separate `<ResponseName>` tag must be provided for each expected response. The tags must be in the order in which the responses will be sent. The intermediate result sets do not affect the sequencing rules. The final result set is transmitted when the action is complete and causes the sequencing rules defined in the action definition to take effect.

NOTE: Although responses must be sent by the application in the same order as the `<ResponseName>` tags, they may be received out of order. This can happen, for example, if different delivery channels are used for different responses or if an implementation sends a message without waiting for the transport-level acknowledgment of the previous message.

When the response to a request indicates a business-level error, the recipient of the request sends no further response messages even if additional responses are defined in the action definition. The action is terminated. Such errors include both application-specific errors and error conditions detected in the document-exchange functions, which usually indicate document format errors. The action may be retried if the allowed number of retries is not exhausted. However, the retry is a new instance of the action. See 2.7.3, "Message Retries".

NOTE: Following termination of a request because of an error indicated in a response message under `<Response>`, the application should notify the framework that the

action is complete. This can be done by means of an action-complete indicator in a message header, if available, or by software means defined by the framework.

The syntax of <Response> is:

```
<Response>  <!--optional-->
     <ResponseName Required=option>name</ResponseName>
          <!-- one or more-->
          <!-- option is "yes" or "no"-->
     </Response>
```

2.9.19.2 Application Result Information

The details of the application-level result information are determined and interpreted by the application. These details are included in the payload of the response message.

NOTE: The framework-level information associated with the message indicates only success or failure. In addition, a field in the standard message header, if present, which is processed by the framework, may indicate success or failure.

2.9.19.3 Exception Response

<ExceptionResponse> defines the response messages for errors and other exception conditions. It contains one or more <ExceptionResponseName> tags. The value of <ExceptionResponseName> must be the same as the value of <RequestName> in the action which is to process the exception response message. The value of <ExceptionResponseName> is an alphanumeric string.

NOTE: A TPA tool that matches the value of <ExceptionResponseName> to the value of the corresponding <RequestName> tag should treat these values as case-insensitive, i.e. the matching names should compare equal regardless of case.

NOTE: An implementation might place a restriction on the length of the string.

NOTE: An exception response message should carry an indicator that it is a response message and the ID of the past request to which it is a response. The response indicator enables the framework to detect that the response was received within the maximum allowed time. An implementation could use the presence of the past request ID as the response indicator.

When an exception response to a request is issued, the recipient of the request sends no further response messages and the action is terminated. The action may be retried if the allowed number of retries is not exhausted. However, the retry is a new instance of the action. See 2.7.3, "Message Retries".

Multiple <ExceptionResponseName> tags may be included under <ExceptionResponse> for different conditions. Any of these exception response messages may be issued. However the first exception response message terminates the action.

The syntax of <ExceptionResponse> is:

```
<ExceptionResponse>
     <ExceptionResponseName>name</ExceptionResponseName>
     <!-- one or more-->
     </ExceptionResponse>
```

2.9.20 Timing of Action Results

For an action invocation, the time to receive results may be considerably longer than the time to acknowledge the request since the time to the results may include processing at the server, actions invoked by the server on subcontractors, and similar functions, in addition to the network delays. The time to perform a given action may be quite long in some cases, possibly on the order of hours or longer.

The <ServerServiceTime> tag (in the service-interface definition) and the <ResponseServiceTime> tag (in the action definition) define the maximum service time until the server completes processing the action request. The <ServiceTime> tag gives the maximum service time. It is a character string containing an integer number of seconds. The specified service time is the maximum time from sending the request message until the last response message is received.

It must be understood that the server service time (<ServerServiceTime> and <ResponseServiceTime>) and the network delay (<NetworkDelay>) are separately stated in the TPA and must be added together to determine the timeout value to be used in waiting for the response. If the recovery from the timeout fails, the timeout must be logged as a permanent error condition.

The <ResponseServiceTime> tag contains the value of the service time for a particular action. It is optional in the action definition. If omitted, the value of <ServerServiceTime> for the same service interface is used. If stated in the action definition, it overrides the <ServerServiceTime> value. If neither tag is present, the client must assume a value outside the scope of the TPA.

For a particular request, the service time stated in <ServerServiceTime> and <ResponseServiceTime> may be overridden by an application-specified service time which might, for example, be stated in a business document exchanged under the TPA.

When <Response> and <ExceptionResponse> are both omitted from the action definition, <ResponseServiceTime> and <ServerServiceTime> are not applicable to this request.

> NOTE: It should be understood that recoverable errors, such as short-duration network failures, are detected by the framework on time scales much shorter than the TPA-defined timeout for the results. An example is failure to establish a TCP/IP connection to transmit the request message. Since failure to receive an acknowledgment to a message is usually caused by a network error or failure, the lack of an acknowledgment should not

be considered a TPA violation. The message should be retried. In general, the framework should retry these error conditions. Once an acknowledgment is received, a results timeout indicates a serious problem at the server that prevented it from completing the requested action, or a TPA violation by the server.

2.9.21 Presumed Result Following Timeout

If a timeout persists after the maximum number of retries is exhausted, it may be possible to presume that the action nonetheless succeeded. Whether or not this is possible depends on the nature of the action and the kind of information that would be returned in the reply.

One example of an action in which success might be presumed is an action that merely confirms the result of a prior action. If the result of the confirmation request is not received, it can be retried at the application level without logging an error.

The <Presume> tag under <ResponseServiceTime> indicates whether the action is presumed to have succeeded or failed after a timeout. The value of <Presume> is success or fail. If the <Presume> tag is provided under <ServerServiceTime>, it states the presumed result for all actions under that service interface unless overridden by the <Presume> tag in a specific action definition. If the <Presume> tag is not provided, the presumption is fail.

It should be understood that if the action invocation reaches the point where the service time timeout condition is relevant, it can be assumed that the server received and acknowledged the invocation message and has assumed responsibility for completed the requested action. See 2.9.20, "Timing of Action Results".

The syntax of <ResponseServiceTime> is:

```
<ResponseServiceTime>
     <ServiceTime>time</ServiceTime>
     <Presume>result</Presume>
          <!--result is success or fail-->
     </ResponseServiceTime>
```

2.9.22 Error Handling

Errors of concern to the TPA are grouped in the following categories:

- Connectivity errors
- Response errors
- Sequencing errors
- Application-specific errors

Connectivity errors are detected as exhaustion of the permitted number of retries for failure to receive the transport-level acknowledgment to the transmission of a message, such as the message that sends an action request to a server.

Response errors are detected as exhaustion of the permitted number of retries for failure to receive the results of an action within the time given by <ResponseServiceTime> or <ServerServiceTime>. This subject is discussed in 2.9.20, "Timing of Action Results" and in 2.7.3, "Message Retries".

Sequencing errors are violations of the order of actions specified in the sequencing rules. See 2.9.5, "Sequencing Rules". A sequencing error could be committed by a client (action invoker) or could result from a state error in the server. Therefore a sequencing error may require arbitration by the arbitrator specified in <Arbitrator>. See 2.4.5, "Arbitration of TPA Disputes".

> NOTE: Sequencing errors can be handled by the framework and error-handling rules do not appear in the TPA. As part of processing sequencing errors, the framework can obtain the recent history of the conversation and send it to the arbitrator.

Application-specific errors are indicated in the payload of the <ExceptionResponse> message.

2.9.23 Sequencing

See 2.9.5, "Sequencing Rules".

2.9.24 Action Security

See 2.7.4, "Message Security" for a discussion of the <ActionSecurity> tag.

2.9.25 Complete Action Definition

Following is the syntax of the action definition.

```
<Action ActionId=identifier
          Type=type
          Invocation=invoc>
               <!--see 2.9.11, "Attributes of the Action Tag" for
               discussion of purpose of each attribute and
               permissible values to right of equal signs-->
          <!--attribute value must be enclosed in quotes-->
     <Request>
          <RequestName>name_of_action</RequestName>
               <!--example:  rsrvAir-->
          <RequestMessage>message</RequestMessage>
          <Channel ChannelId=name />  <!--optional-->
          </Request>
     <Response> <!-- optional -->
          <ResponseName Required=option>name</ResponseName>
               <!-- option is "yes" or "no"-->
               <!-- one or more -->
          </Response>
     <ExceptionResponse>
```

```
          <ExceptionResponseName>name</ExceptionResponseName>
          <!-- one or more-->
          </ExceptionResponse>
     <ResponseServiceTime>
          <ServiceTime>time_value</ServiceTime>
          <Presume>result</Presume>
               <!--result is success or fail-->
          </ResponseServiceTime>
     <Sequencing>   <!--optional-->
          <Enable>   <!--actions permitted after this one-->
               <RequestName>name_of_action</RequestName>
                    ...
               </Enable>
          <Disable>  <!--actions not permitted after this one-->
               <RequestName>name_of_action</RequestName>
               <RequestName>name_of_action</RequestName>
                    ...
               </Disable>
          </Sequencing>
     <ActionSecurity>option</ActionSecurity>
          <!-- option is yes or no-->
     </Action>
```

2.10 Comment Text

In general, a TPA between businesses can be expected to be accompanied by a traditional legal contract document. In addition, it may be desirable to include within the TPA textual statements explaining various points or referring to the traditional legal contract. The <Comment> tag is used for this purpose. The value of <Comment> is a text string that can contain alphanumeric characters and punctuation but no tags or angle brackets (XML restriction). <Comment> is optional. Multiple <Comment> tags are permitted to provide a very simple formatting capability.

Examples of such text are shown below.

```
<Comment>
     If it can be proved via the following third parties, that we
     are not reachable via the network, we will honor late
     cancellation.
     </Comment>
<Comment>
     If Hotelco is not reachable for more than three consecutive
     days, Hotelco will offer the customer free two nights stay.
     </Comment>
```

Another use of <Comment> might be to identify a traditional paper contract that accompanies the TPA.

NOTE: Information that is copied from other sources, e.g. an accompanying legal contract,

generally should not be included in `<Comment>`. Such information could change during the lifetime of the electronic TPA and the changes would not necessarily be reflected back into the original TPA.

Appendix 1 Complete TPA Definition

Following is the full XML TPA definition. Note that this is really a TPA template since it does not contain specific quantities for most of the tag values.

```
<?xml version="1.0"?>
<!--(C) Copyright IBM Corporation 2000; All rights reserved-->
<!DOCTYPE TPA SYSTEM "TPA.dtd">
<!--The DOCTYPE line above is for use only with a DTD parser.
     It must be removed for use with an XML Schema parser.-->
<TPA xmlns="tpa.xsd">
<!--The xmlns attribute is for use with an XML Schema parser.
     The definition in the DTD causes it to be ignored by
     a DTD parser.-->
   <TPAInfo>  <!-- TPA preamble -->
     <TPAName>TPA name</TPAName>
          <!--name of TPA-->
     <TPAType>
          <Protocol>protocol</Protocol>
          <Version>version</Version>
          <Type>type</Type>
          </TPAType>
     <Role>
          <!--defines the role parameters representing parties to
               the TPA-->
          <RoleDefn>     <!--one or more-->
               <RoleName>rolename</RoleName>
                    <!--example:  hotel, airline -->
               <RolePlayer>playername</RolePlayer>
                    <!--example:  Hotelco, Airlineco-->
               </RoleDefn>
          </Role>
     <Participants>
          <Member MemberId="identifer" IdCodeType="code">
          <PartyName Partyname="playername">rolename
                    </PartyName>
                    <!--name of organization-->
               <CompanyTelephone>
                    telephone_number</CompanyTelephone>
               <Address>
                    <AddressType>type</AddressType>
                         <!--AddressType is shipping, billing, or
                              location-->
                    <AddressLine>address_line1</AddressLine>
                    <AddressLine>address_line2</AddressLine>
                         <!--as many address lines as needed-->
                    <City>city_name</City>
                    <State>state_name</State>
                         <!--or province or similar division-->
                    <Zip>zipcode</Zip>
                         <!--or similar postal code-->
                    <Country>country_name</Country>
```

```
                    </Address>
               <Contact Type="primary">  <!--one or more-->
                    <!--Type is "primary" or "secondary",
                         default is "primary"-->
                    <LastName>name</LastName>
                    <FirstName>name</FirstName>
                    <MiddleName>name</MiddleName>
                    <Title>title</Title>
                    <ContactTelephone Type="primary">
                         phone_no</ContactTelephone>
                         <!--1 or more-->
                         <!--Type values same as for Contact-->
                    <EMail Type="primary">
                         email_address</EMail>
                         <!--1 or more-->
                         <!--Type values same as for Contact-->
                    <Fax>fax_number</Fax>   <!--optional-->
                    </Contact>
               </Member>
               <!--Additional Member definitions-->
          </Participants>
     <Arbitrator MemberId="identifer" IdCodeType="code">
           arbitrator_name</Arbitrator>
     <Duration>     <!--valid duration for TPA-->
          <Start>  <!--Optional-->
               <Date>date</Date>
               <Time>time</Time>
               </Start>
          <End>     <!--Optional-->
               <Date>date</Date>
               <Time>time</Time>
               </End>
          </Duration>
     <InvocationLimit>number</InvocationLimit>
          <!--number of instantiations permitted before
               renegotiation is required-->
     <ConcurrentConversations>number</ConcurrentConversations>
          <!--Maximum number of concurrent conversations
               permitted-->
     <ConversationLife>time</ConversationLife>
          <!--maximum lifetime of a single conversation-->
     </TPAInfo>
  <DeliveryChannelSet> <!--one or more-->
     <!--one or more DeliveryChannel tags-->
     <DeliveryChannel
          ChannelId="channel01"
          TransportId="transport01"
          DocExchangeId="docexchange01" >
          <Version>version</Version>
          </DeliveryChannel>
     </DeliveryChannelSet>
  <Transport TransportId="transport01">  <!--one or more-->
  <!--*************************************************-->
  <!--Following are separate Communication tag sets for the
```

```
          different protocols.-->
  <!--*****************************************************-->
   <Communication>
     <HTTP>
          <Version>version</Version>
          <HTTPNode> <!--One for each party-->
               <OrgName Partyname="playername"/>
               <HTTPAddress> <!--each xxxURL tag is optional-->
                  <LogonURL>http://www/a.xxx.com</LogonURL>
                  <RequestURL>http://www.b.xxx.com</RequestURL>
                  <ResponseURL>http://www.c.xxx.com</ResponseURL>
                  </HTTPAddress>
               </HTTPNode>
          <TransportEncoding>encode</TransportEncoding>
          <TransportTimeout>  <!--optional-->
               <Timeout>time</Timeout>
               <Retries>number</Retries>
               <RetryInterval>time</RetryInterval>
               </TransportTimeout>
          <NetworkDelay>time</NetworkDelay>  <!--Optional-->
          </HTTP>
     </Communication>
<!--*****************************************************-->

     <!--Queued Messages transport goes here-->

<!--*****************************************************-->
 <Communication>
   <SMTP>
     <Version>version</Version>
     <SMTPNode> <!--One for each party-->
          <OrgName Partyname="playername"/>
          <SMTPAddress>address</SMTPAddress>
               <!--party's email address-->
          </SMTPNode>
     <TransportEncoding>encode</TransportEncoding>
     <TransportTimeout>  <!--optional-->
          <Timeout>time</Timeout>
          <Retries>number</Retries>
          <RetryInterval>time</RetryInterval></TransportTimeout>
     <NetworkDelay>time</NetworkDelay>
     </SMTP>
  </Communication>
<!--*****************************************************-->
 <Communication>
     <VANEDI>
       <Version>version</Version>
       <VANEDINode>
          <!--One for each party-->
          <OrgName Partyname="playername"/>
          <VANEDIAddress>
               <InBox>name</InBox> <!--mailbox name-->
```

```
                <OutBox>name</OutBox> <!--mailbox name-->
                </VANEDIAddress>
           </VANEDINode>
        <TransportEncoding>encode</TransportEncoding>
        <TransportTimeout> <!--optional-->
           <Timeout>time</Timeout>
           <Retries>number</Retries>
           <RetryInterval>time</RetryInterval>
           </TransportTimeout>
        <NetworkDelay>time</NetworkDelay>  <!--optional-->
        </VANEDI>
     </Communication>
<!--*************************************************-->
 <Communication>
     <FTP>
          <Version>version</Version>
          <FTPNode>  <!--one for each party-->
               <OrgName Partyname="playername"/>
               <FTPAddress>
                    <InDirectory>path</InDirectory>
                         <!--for puts-->
                    <OutDirectory>path</OutDirectory>
                         <!--for gets-->
                    </FTPAddress>
               <TransferType>type</TransferType>
                    <!--type is ascii or binary -->
               <PassiveMode>choice</PassiveMode>
                    <!--choice is yes or no-->
               <ControlPort>number </ControlPort>
                    <!--For use with passive mode-->
               </FTPNode>
          <TransportEncoding>encode</TransportEncoding>
          <TransportTimeout>  <!--optional-->
               <Timeout>time</Timeout>
               <Retries>number</Retries>
               <RetryInterval>time</RetryInterval>
               </TransportTimeout>
          <NetworkDelay>time</NetworkDelay>
          </FTP>
     </Communication>
<!--*************************************************-->
   <TransportSecurity>
     <Encryption>  <!--Optional-->
          <Protocol>protocol</Protocol>
          <Version>version</Version>
               <!--string denoting version of protocol-->
          <Certificate>
               <CertType>type</CertType>
                         <!--Example:  X.509V1, X.509V2-->
                    <KeyLength>length_in_bits</KeyLength>
                    <Party>
                         <!--one for each party that can act
                              as a server-->
                         <OrgName Partyname="playername"/>
```

```
                              <!--name of party-->
                         <IssuerOrgName>name</IssuerOrgName>
                              <!--Example:  VeriSign, Inc.-->
                         <IssuerCertSource>url</IssuerCertSource>
                              <!--URL of certificate-->
                         </Party>
                    </Certificate>
               </Encryption>
          <Authentication>
               <PasswordAuthentication>
                    <Protocol>protocol</Protocol>
                         <!--Example: SSL-->
                    <Version>version</Version> <!--optional-->
                         <!--version of protocol-->
                    <LogonInfo>  <!--optional-->
                         <LogonParty>
                              <!--one for each party that can act as a
                                 server-->
                              <OrgName Partyname="playername"/>
                              <UserId>value</UserId>
                              <Password>value</Password>
                              </LogonParty>
                         </LogonInfo>
                    </PasswordAuthentication>
               <CertificateAuthentication>  <!--optional-->
                    <Protocol>protocol</Protocol>
                    <Version>version</Version>
                    <Certificate>
                         <CertType>type</CertType>
                              <!--Example:  X.509V1, X.509V2-->
                         <KeyLength>length_in_bits</KeyLength>
                         <Party>   <!--one for each party-->
                              <OrgName Partyname="playername"/>
                                   <!--name of party-->
                              <IssuerOrgName>name</IssuerOrgName>
                                   <!--Example: VeriSign, Inc.-->
                              <IssuerCertSource>url</IssuerCertSource>
                                   <!--URL of certificate-->
                              </Party>
                         </Certificate>
                    </CertificateAuthentication>
               </Authentication>
            </TransportSecurity>
          </Transport>
<DocExchange DocExchangeId="docexchange01">  <!--one or more-->
          <MessageEncoding>encoding</MessageEncoding>
               <!--currently must be BASE64 or omitted-->
          <MessageIdempotency>choice</MessageIdempotency>
               <!--choice is yes or no-->
          <MessageRetries>
               <Retries>number</Retries>
               <RetryInterval>time</RetryInterval>
               </MessageRetries>
          <MessageSecurity>
```

```
          <NonRepudiation>
               <Protocol>protocol</Protocol>
                    <!--example: DigitalSignature-->
               <Version>version</Version>
                    <!--version of protocol-->
               <HashFunction>function</HashFunction>
                    <!--example:  MD5, SHA1-->
               <EncryptionAlgorithm>algorithm
                    </EncryptionAlgorithm>
                    <!--Example:   RC2 -->
               <SignatureAlgorithm>algorithm</SignatureAlgorithm>
                    <!--Example:  DSA-->
               <Certificate>
                    <CertType>type</CertType>
                         <!--Example:  X.509V1, X.509V2-->
                    <KeyLength>length_in_bits</KeyLength>
                    <Party>   <!--one for each party-->
                         <OrgName Partyname="playername"/>
                              <!--name of party-->
                         <IssuerOrgName>name</IssuerOrgName>
                              <!--Example:  VeriSign, Inc.-->
                         <IssuerCertSource>url</IssuerCertSource>
                              <!--URL of  certificate-->
                         </Party>
                    </Certificate>
               </NonRepudiation>
          <DigitalEnvelope>
               <Protocol>DigitalEnvelope</Protocol>
               <Version>version</Version>
               <EncryptionAlgorithm>algor</EncryptionAlgorithm>
                    <!--example:  RC4-->
               <Certificate>
                    <CertType>type</CertType>
                         <!--Example:  X.509V3-->
                    <KeyLength>length_in_bits</KeyLength>
                         <!--example:  512-->
                    <Party>   <!--one for each party-->
                         <OrgName Partyname="playername"/>
                              <!--name of party-->
                         <OrgCertSource>url</OrgCertSource>
                              <!--URL of certificate-->
                         <IssuerOrgName>name</IssuerOrgName>
                              <!--Example:  VeriSign, Inc.-->
                         <IssuerCertSource>url</IssuerCertSource>
                              <!--URL of  certificate-->
                         </Party>
                    </Certificate>
               </DigitalEnvelope>
          </MessageSecurity>
     </DocExchange>
    <BusinessProtocol>
     <MessageSet>name</MessageSet>
     <BusinessInterface>
     <ServiceInterface InterfaceId="id">
```

```
<!-- one for each service provider-->
<!--Example:  InterfaceId="SERVER01"-->
<OrgName Partyname="playername"/>
     <!--the party which acts as a server-->
<TaskName>name</TaskName>  <!--optional-->
<ActionMenu>
 <!--one Action tag for each action-->
 <Action
     ActionId="identifier"
     Type="basic"
     Invocation="asyncOnly">
          <!--attribute value must be enclosed in quotes-->
     <Request>
          <RequestName>name_of_action</RequestName>
               <!--example:  rsrvAir-->
          <RequestMessage>message</RequestMessage>
          <Channel ChannelId="channel01"/> <!--optional-->
          </Request>
     <Response>
          <ResponseName Required="yes">name</ResponseName>
               <!--one or more-->
               <!--"option" = "yes" or "no"-->
          </Response>
     <ExceptionResponse>
          <ExceptionResponseName>name
               </ExceptionResponseName>
               <!-- one or more-->
          </ExceptionResponse>
     <ResponseServiceTime>
          <ServiceTime>time_value</ServiceTime>
          <Presume>result</Presume>
               <!--result is success or fail-->
          </ResponseServiceTime>
     <Sequencing>   <!--optional-->
          <Enable>   <!--actions permitted after this one-->
               <RequestName>name_of_action</RequestName>
                    <!--one or more-->
               </Enable>
          <Disable>
               <!--actions not permitted after this one-->
               <RequestName>name_of_action</RequestName>
               <RequestName>name_of_action</RequestName>
                    <!--one or more-->
               </Disable>
          </Sequencing>
     <ActionSecurity>option</ActionSecurity>
          <!-- option is yes or no-->
     </Action>
    </ActionMenu>
<ServerServiceTime>
     <ServiceTime>time</ServiceTime>
     <Presume>result</Presume>
          <!--result is success or fail-->
     </ServerServiceTime>
```

```
  <StartEnabled>
       <RequestName>action_name</RequestName>
            <!--one for each action permitted as the initial
            action-->
       </StartEnabled>
  <TerminateConversation>  <!--optional-->
       <RequestName>action_name</RequestName>
            <!--one for each action defined as ending a
                 conversation-->
       </TerminateConversation>
  <ServiceSecurity>option</ServiceSecurity>
       <!-- option is yes or no-->
  </ServiceInterface>
  </BusinessInterface>
 </BusinessProtocol>
<Comment>   <!--One or more-->
  <!--Explanatory text-->
  </Comment>
</TPA>
```

Appendix 2 DTD Corresponding to Complete TPA Definition

Following is the DTD that defines the contents of the TPA.

```
<!--*********************************************************************-->
<!-- (C)Copyright IBM Corporation 2000; all rights reserved              -->
<!--                TPA DTD                                              -->
<!-- File name: TPA.dtd                                                  -->
<!--*********************************************************************-->
<!--*********************************************************************-->
<!ELEMENT TPA (TPAInfo, DeliveryChannelSet, Transport+,
  DocExchange+, BusinessProtocol, Comment*)>
<!ATTLIST TPA xmlns CDATA #IMPLIED>
<!--The xmlns attribute identifies the XML Schema document when a TPA is  -->
<!--parsed by an XML Schema parser. It is ignored by a DTD parser.        -->
<!--*********************************************************************-->
<!-- General information                                                 -->
<!--*********************************************************************-->
<!ELEMENT TPAInfo (TPAName, TPAType?, Role?, Participants, Arbitrator?,
  Duration?, InvocationLimit?, ConcurrentConversations?, ConversationLife?)>
<!ELEMENT TPAName (#PCDATA)>
<!ELEMENT TPAType (Protocol, Version, Type)>
<!ELEMENT Protocol (#PCDATA)>
<!ELEMENT Version (#PCDATA)>
<!ELEMENT Type (#PCDATA)>
<!--*********************************************************************-->
<!-- Specification of Roles and Participants                             -->
<!--*********************************************************************-->
<!ELEMENT Role (RoleDefn+)>
<!ELEMENT RoleDefn (RoleName, RolePlayer)>
<!ELEMENT RoleName (#PCDATA)>
<!ELEMENT RolePlayer (#PCDATA)>
<!ELEMENT Participants (Member+)>
<!ELEMENT Member (PartyName, CompanyTelephone, Address+, Contact+)>
<!ATTLIST Member
  MemberId CDATA #REQUIRED
  IdCodeType CDATA #REQUIRED >
<!ELEMENT PartyName (#PCDATA)>
<!ATTLIST PartyName Partyname ID #REQUIRED>
<!ELEMENT CompanyTelephone (#PCDATA)>
<!ELEMENT Address (AddressType, AddressLine+, City, State, Zip, Country)>
<!ELEMENT AddressType (#PCDATA)>
<!ELEMENT AddressLine (#PCDATA)>
<!ELEMENT City (#PCDATA)>
<!ELEMENT State (#PCDATA)>
<!ELEMENT Zip (#PCDATA)>
<!ELEMENT Country (#PCDATA)>
<!ELEMENT Contact (LastName, FirstName, MiddleName, Title, ContactTelephone+,
 EMail+, Fax?)>
<!ATTLIST Contact
  Type (primary|secondary) "primary" >
<!ELEMENT LastName (#PCDATA)>
<!ELEMENT FirstName (#PCDATA)>
<!ELEMENT MiddleName (#PCDATA)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT ContactTelephone (#PCDATA)>
<!ATTLIST ContactTelephone
  Type (primary|secondary) "primary" >
<!ELEMENT EMail (#PCDATA)>
<!ATTLIST EMail
  Type (primary|secondary) "primary" >
```

```
<!ELEMENT Fax (#PCDATA)>
<!ELEMENT Arbitrator (#PCDATA)>
<!ATTLIST Arbitrator
  MemberId CDATA #REQUIRED
  IdCodeType CDATA #REQUIRED >
<!ELEMENT Duration  (Start?, End?)>
<!ELEMENT Start (Date, Time)>
<!ELEMENT End (Date, Time)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT Time (#PCDATA)>
<!ELEMENT InvocationLimit (#PCDATA)>
<!ELEMENT ConcurrentConversations (#PCDATA)>
<!ELEMENT ConversationLife (#PCDATA)>
<!--************************************************************************-->
<!-- Specification of Delivery Channels                                     -->
<!--************************************************************************-->
<!ELEMENT DeliveryChannelSet (DeliveryChannel+) >
<!ELEMENT DeliveryChannel (Version?) >
<!ATTLIST DeliveryChannel
  ChannelId ID #REQUIRED
  TransportId IDREF #REQUIRED
  DocExchangeId IDREF #REQUIRED>
<!--************************************************************************-->
<!-- Specification of Transport Protocol                                    -->
<!--************************************************************************-->
<!ELEMENT Transport (Communication,  TransportSecurity?)>
<!ATTLIST Transport
  TransportId ID #REQUIRED>
<!ELEMENT Communication (HTTP|SMTP|VANEDI|FTP)>
<!ELEMENT HTTP (Version?, HTTPNode+, TransportEncoding?, TransportTimeout?,
  NetworkDelay?)>
<!ELEMENT SMTP (Version?, SMTPNode+, TransportEncoding?, TransportTimeout?,
  NetworkDelay?)>
<!ELEMENT VANEDI (Version?, VANEDINode+, TransportEncoding?,
  TransportTimeout?, NetworkDelay?)>
<!ELEMENT FTP (Version?, FTPNode+, TransportEncoding?, TransportTimeout?,
  NetworkDelay?)>
<!ELEMENT HTTPNode (OrgName, HTTPAddress)>
<!ELEMENT SMTPNode (OrgName, SMTPAddress)>
<!ELEMENT VANEDINode (OrgName, VANEDIAddress)>
<!ELEMENT FTPNode (OrgName, FTPAddress,
  TransferType, PassiveMode, ControlPort)>
<!ELEMENT OrgName EMPTY>
<!ATTLIST OrgName Partyname IDREF #REQUIRED>
<!ELEMENT HTTPAddress (LogonURL?, RequestURL?, ResponseURL?)>
<!ELEMENT LogonURL (#PCDATA)>
<!ELEMENT RequestURL (#PCDATA)>
<!ELEMENT ResponseURL (#PCDATA)>
<!ELEMENT VANEDIAddress (InBox, OutBox)>
<!ELEMENT InBox (#PCDATA)>
<!ELEMENT OutBox (#PCDATA)>
<!ELEMENT SMTPAddress (#PCDATA)>
<!ELEMENT FTPAddress (InDirectory, OutDirectory)>
<!ELEMENT InDirectory (#PCDATA)>
<!ELEMENT OutDirectory (#PCDATA)>
<!ELEMENT TransferType (#PCDATA)>
<!ELEMENT PassiveMode (#PCDATA)>
<!ELEMENT ControlPort (#PCDATA)>
<!ELEMENT TransportEncoding (#PCDATA)>
<!ELEMENT TransportTimeout (Timeout, Retries, RetryInterval)>
<!ELEMENT Timeout (#PCDATA)>
<!ELEMENT Retries (#PCDATA)>
<!ELEMENT RetryInterval (#PCDATA)>
<!ELEMENT NetworkDelay (#PCDATA)>
```

```
<!--************************************************************************-->
<!-- Specification of Transport Security Protocol                           -->
<!--************************************************************************-->
<!ELEMENT TransportSecurity (Encryption?, Authentication?)>
<!ELEMENT Encryption (Protocol, Version?, Certificate)>
<!ELEMENT Authentication (PasswordAuthentication?,
  CertificateAuthentication?)>
<!ELEMENT PasswordAuthentication (Protocol, Version?, LogonInfo?)>
<!ELEMENT LogonInfo (LogonParty+)>
<!ELEMENT LogonParty (OrgName, UserId, Password)>
<!ELEMENT UserId (#PCDATA)>
<!ELEMENT Password (#PCDATA)>
<!ELEMENT CertificateAuthentication (Protocol, Version?, Certificate)>
<!ELEMENT Certificate (CertType, KeyLength, Party+)>
<!ELEMENT CertType (#PCDATA)>
<!ELEMENT KeyLength (#PCDATA)>
<!ELEMENT Party (OrgName, OrgCertSource?, IssuerOrgName, IssuerCertSource)>
<!ELEMENT OrgCertSource (#PCDATA)>
<!ELEMENT IssuerOrgName (#PCDATA)>
<!ELEMENT IssuerCertSource (#PCDATA)>
<!--************************************************************************-->
<!-- Specification of DocExchange Protocol                                  -->
<!--************************************************************************-->
<!ELEMENT DocExchange (MessageEncoding?,
   MessageIdempotency, MessageRetries?,  MessageSecurity?)>
<!ATTLIST DocExchange
  DocExchangeId ID #REQUIRED>
<!ELEMENT MessageEncoding (#PCDATA)>
<!ELEMENT MessageIdempotency (#PCDATA)>
<!ELEMENT MessageRetries (Retries, RetryInterval)>
<!--************************************************************************-->
<!-- Specification of Message Security Protocol                             -->
<!--************************************************************************-->
<!ELEMENT MessageSecurity (NonRepudiation?, DigitalEnvelope?)>
<!ELEMENT NonRepudiation (Protocol, Version?, HashFunction,
  EncryptionAlgorithm,
  SignatureAlgorithm, Certificate)>
<!ELEMENT HashFunction (#PCDATA)>
<!ELEMENT EncryptionAlgorithm (#PCDATA)>
<!ELEMENT SignatureAlgorithm (#PCDATA)>
<!ELEMENT DigitalEnvelope (Protocol?, Version?, EncryptionAlgorithm,
  Certificate?)>
<!--************************************************************************-->
<!-- Specification of Business Protocol                                     -->
<!--************************************************************************-->
<!ELEMENT BusinessProtocol (MessageSet, BusinessInterface)>
<!ELEMENT MessageSet (#PCDATA)>
<!ELEMENT BusinessInterface (ServiceInterface+)>
<!ELEMENT ServiceInterface (OrgName, TaskName?, ActionMenu,
  ServerServiceTime?,
  StartEnabled?, TerminateConversation?, ServiceSecurity?)>
<!ATTLIST ServiceInterface
  InterfaceId CDATA #REQUIRED >
<!ELEMENT TaskName (#PCDATA)>
<!ELEMENT ActionMenu (Action+)>
<!ELEMENT Action (Request, Response?,ExceptionResponse?,
  ResponseServiceTime?,
  Sequencing?, ActionSecurity?)>
<!ATTLIST Action
  ActionId CDATA #REQUIRED
  Type (basic|concurrent) "basic"
  Invocation (syncOnly | asyncOnly  ) "asyncOnly" >
<!--************************************************************************-->
<!-- Each Request is identified by a request name.                          -->
```

```
<!--*********************************************************************-->
<!ELEMENT Request (RequestName, RequestMessage, Channel?)>
<!ELEMENT RequestName (#PCDATA)>
<!ELEMENT RequestMessage (#PCDATA)>
<!--*********************************************************************-->
<!-- Specify optional binding to delivery channel                      -->
<!--*********************************************************************-->
<!ELEMENT Channel EMPTY>
<!ATTLIST Channel
  ChannelId IDREF #REQUIRED>
<!--*********************************************************************-->
<!-- Response is a reply from the server following a prior              -->
<!-- request invocation. There can be zero, one or multiple responses for -->
<!-- a given request, depending on the application.                     -->
<!--*********************************************************************-->
<!ELEMENT Response (ResponseName+)>
<!ELEMENT ResponseName (#PCDATA)>
<!ATTLIST ResponseName
  Required (yes|no) "yes">
<!ELEMENT ExceptionResponse (ExceptionResponseName+)>
<!ELEMENT ExceptionResponseName (#PCDATA)>
<!ELEMENT ResponseServiceTime (ServiceTime, Presume?)>
<!ELEMENT ServiceTime (#PCDATA)>
<!ELEMENT Presume (#PCDATA)>
<!--*********************************************************************-->
<!-- Specify none or more sequencing rules to be applied to this action   -->
<!--*********************************************************************-->
<!ELEMENT Sequencing (Enable?, Disable?)>
<!ELEMENT Enable (RequestName+)>
<!ELEMENT Disable (RequestName+)>
<!--*********************************************************************-->
<!-- Specify ActionSecurity yes for this action to overide the message    -->
<!-- security default defined in ServiceSecurity.                       -->
<!-- No Need to specify this tag if default is no                       -->
<!--*********************************************************************-->
<!ELEMENT ActionSecurity (#PCDATA)>
<!--*********************************************************************-->
<!-- Service time-out if specified is the default time-out for actions    -->
<!-- that do not have time-out defined.                                 -->
<!--*********************************************************************-->
<!ELEMENT ServerServiceTime (ServiceTime, Presume?)>
<!--*********************************************************************-->
<!ELEMENT StartEnabled (RequestName+)>
<!ELEMENT TerminateConversation (RequestName+)>
<!--*********************************************************************-->
<!-- Specify default message security for all actions in the server.      -->
<!-- Value is yes. No Need to specify this tag if message security not    -->
<!-- wanted since default is no.                                        -->
<!--*********************************************************************-->
<!ELEMENT ServiceSecurity (#PCDATA)>
<!--*********************************************************************-->
<!ELEMENT Comment (#PCDATA)>
```

Appendix 3 XML Schema Document Corresponding to Complete TPA Definition

Following is the XML Schema document that defines the contents of the TPA.

```
<?xml version="1.0"?>
<!-- (C) Copyright IBM Corporation 2000; all rights reserved            -->
<schema xmlns='http://www.w3.org/1999/XMLSchema'>
<annotation>
  <documentation>
   XML Schema (Feb 25, 2000 WD) based on tpastd31.dtd
   Produced by IBM's Data Descriptor By Example (DDbE) available at
           http://www.alphaworks.ibm.com/tech/DDbE
  </documentation>
</annotation>

<element  name="TPA">
<complexType  content="elementOnly">
<sequence>
<element name="TPAInfo"/>
<element name="DeliveryChannelSet"/>
<element name="Transport"  maxOccurs='*'/>
<element name="DocExchange"  maxOccurs='*'/>
<element name="BusinessProtocol"/>
<element name="Comment"  maxOccurs='*' minOccurs="0"/>
</sequence>
      <attribute name="xmlns" type="string"  minOccurs="0" />
</complexType>
</element>
<element  name="TPAInfo">
<complexType  content="elementOnly">
<sequence>
<element name="TPAName"/>
<element name="TPAType"   minOccurs="0" maxOccurs="1"/>
<element name="Role"   minOccurs="0" maxOccurs="1"/>
<element name="Participants"/>
<element name="Arbitrator"   minOccurs="0" maxOccurs="1"/>
<element name="Duration"   minOccurs="0" maxOccurs="1"/>
<element name="InvocationLimit"   minOccurs="0" maxOccurs="1"/>
<element name="ConcurrentConversations"   minOccurs="0" maxOccurs="1"/>
<element name="ConversationLife"   minOccurs="0" maxOccurs="1"/>
</sequence>
</complexType>
</element>
<element  name="TPAName">
<simpleType  name="TPAName" base="string"/>
</element>
<element  name="TPAType">
<complexType  content="elementOnly">
<sequence>
<element name="Protocol"/>
<element name="Version"/>
<element name="Type"/>
</sequence>
```

```
</complexType>
</element>
<element  name="Protocol">
<simpleType  name="Protocol" base="string"/>
</element>
<element  name="Version">
<simpleType  name="Version" base="string"/>
</element>
<element  name="Type">
<simpleType  name="Type" base="string"/>
</element>
<element  name="Role">
<complexType  content="elementOnly">
<element name="RoleDefn"  maxOccurs='*'/>
</complexType>
</element>
<element  name="RoleDefn">
<complexType  content="elementOnly">
<sequence>
<element name="RoleName"/>
<element name="RolePlayer"/>
</sequence>
</complexType>
</element>
<element  name="RoleName">
<simpleType  name="RoleName" base="string"/>
</element>
<element  name="RolePlayer">
<simpleType  name="RolePlayer" base="string"/>
</element>
<element  name="Participants">
<complexType content="elementOnly">
<element name="Member"  maxOccurs='*'/>
</complexType>
</element>
<element  name="Member">
<complexType  content="elementOnly">
<sequence>
<element name="PartyName"/>
<element name="CompanyTelephone"/>
<element name="Address"  maxOccurs='*'/>
<element name="Contact"  maxOccurs='*'/>
</sequence>
      <attribute name="IdCodeType" type="string"  minOccurs="1" />

      <attribute name="MemberId" type="string"  minOccurs="1" />
</complexType>
</element>
<element  name="PartyName">
<simpleType  name="PartyName" base="string"/>
</element>
<element  name="CompanyTelephone">
<simpleType  name="CompanyTelephone" base="string"/>
</element>
<element  name="Address">
<complexType  content="elementOnly">
<sequence>
```

```
<element name="AddressType"/>
<element name="AddressLine"  maxOccurs='*'/>
<element name="City"/>
<element name="State"/>
<element name="Zip"/>
<element name="Country"/>
</sequence>
</complexType>
</element>
<element  name="AddressType">
<simpleType  name="AddressType" base="string"/>
</element>
<element  name="AddressLine">
<simpleType  name="AddressLine" base="string"/>
</element>
<element  name="City">
<simpleType  name="City" base="string"/>
</element>
<element  name="State">
<simpleType  name="State" base="string"/>
</element>
<element  name="Zip">
<simpleType  name="Zip" base="string"/>
</element>
<element  name="Country">
<simpleType  name="Country" base="string"/>
</element>
<element  name="Contact">
<complexType  content="elementOnly">
<sequence>
<element name="LastName"/>
<element name="FirstName"/>
<element name="MiddleName"/>
<element name="Title"/>
<element name="ContactTelephone"  maxOccurs='*'/>
<element name="EMail"  maxOccurs='*'/>
<element name="Fax"   minOccurs="0" maxOccurs="1"/>
</sequence>
      <attribute name="Type" type="NMTOKEN"  minOccurs="0" default="primary"
>
 <simpleType base="NMTOKEN">
 <enumeration value='primary|secondary' />
 </simpleType>
 </attribute>
 </complexType>
 </element>
 <element  name="LastName">
 <simpleType  name="LastName" base="string"/>
 </element>
 <element  name="FirstName">
 <simpleType  name="FirstName" base="string"/>
 </element>
 <element  name="MiddleName">
 <simpleType  name="MiddleName" base="string"/>
 </element>
 <element  name="Title">
 <simpleType  name="Title" base="string"/>
```

```
 </element>
 <element  name="ContactTelephone">
 <simpleType  name="ContactTelephone" base="string"/>
 </element>
 <element  name="EMail">
 <simpleType  name="EMail" base="string"/>
 </element>
 <element  name="Fax">
 <simpleType  name="Fax" base="string"/>
 </element>
 <element  name="Arbitrator">
 <simpleType  name="Arbitrator" base="string"/>
 </element>
 <element  name="Duration">
 <complexType  content="elementOnly">
 <sequence>
 <element name="Start"   minOccurs="0" maxOccurs="1"/>
 <element name="End"   minOccurs="0" maxOccurs="1"/>
 </sequence>
 </complexType>
 </element>
 <element  name="Start">
 <complexType  content="elementOnly">
 <sequence>
 <element name="Date"/>
 <element name="Time"/>
 </sequence>
 </complexType>
 </element>
 <element  name="End">
 <complexType  content="elementOnly">
 <sequence>
 <element name="Date"/>
 <element name="Time"/>
 </sequence>
 </complexType>
 </element>
 <element  name="Date">
 <simpleType  name="Date" base="string"/>
 </element>
 <element  name="Time">
 <simpleType  name="Time" base="string"/>
 </element>
 <element  name="InvocationLimit">
 <simpleType  name="InvocationLimit" base="string"/>
 </element>
<element  name="ConcurrentConversations">
<simpleType  name="ConcurrentConversations" base="string"/>
</element>
<element  name="ConversationLife">
<simpleType  name="ConversationLife" base="string"/>
</element>
<element  name="DeliveryChannelSet">
<complexType  content="elementOnly">
<element name="DeliveryChannel"  maxOccurs='*'/>
</complexType>
</element>
```

```
<element  name="DeliveryChannel">
<complexType  content="elementOnly">
<element name="Version"   minOccurs="0" maxOccurs="1"/>
      <attribute name="DocExchangeId" type="IDREF"  minOccurs="1" />

      <attribute name="TransportId" type="IDREF"  minOccurs="1" />

      <attribute name="ChannelId" type="ID"  minOccurs="1" />
</complexType>
</element>
<element  name="Transport">
<complexType  content="elementOnly">
<sequence>
<element name="Communication"/>
<element name="TransportSecurity"   minOccurs="0" maxOccurs="1"/>
</sequence>
      <attribute name="TransportId" type="ID"  minOccurs="1" />
</complexType>
</element>
<element  name="Communication">
<complexType  content="elementOnly">
<choice>
<element name="HTTP"/>
<element name="SMTP"/>
<element name="VANEDI"/>
<element name="FTP"/>
</choice>
</complexType>
</element>
<element  name="HTTP">
<complexType  content="elementOnly">
<sequence>
<element name="Version"   minOccurs="0" maxOccurs="1"/>
<element name="HTTPNode"  maxOccurs='*'/>
<element name="TransportEncoding"   minOccurs="0" maxOccurs="1"/>
<element name="TransportTimeout"   minOccurs="0" maxOccurs="1"/>
<element name="NetworkDelay"   minOccurs="0" maxOccurs="1"/>
</sequence>
</complexType>
</element>
<element  name="SMTP">
<complexType  content="elementOnly">
<sequence>
<element name="Version"   minOccurs="0" maxOccurs="1"/>
<element name="SMTPNode"  maxOccurs='*'/>
<element name="TransportEncoding"   minOccurs="0" maxOccurs="1"/>
<element name="TransportTimeout"   minOccurs="0" maxOccurs="1"/>
<element name="NetworkDelay"   minOccurs="0" maxOccurs="1"/>
</sequence>
</complexType>
</element>
<element  name="VANEDI">
<complexType  content="elementOnly">
<sequence>
<element name="Version"   minOccurs="0" maxOccurs="1"/>
<element name="VANEDINode"  maxOccurs='*'/>
<element name="TransportEncoding"   minOccurs="0" maxOccurs="1"/>
```

```
<element name="TransportTimeout"   minOccurs="0" maxOccurs="1"/>
<element name="NetworkDelay"   minOccurs="0" maxOccurs="1"/>
</sequence>
</complexType>
</element>
<element  name="FTP">
<complexType  content="elementOnly">
<sequence>
<element name="Version"   minOccurs="0" maxOccurs="1"/>
<element name="FTPNode"  maxOccurs='*'/>
<element name="TransportEncoding"   minOccurs="0" maxOccurs="1"/>
<element name="TransportTimeout"   minOccurs="0" maxOccurs="1"/>
<element name="NetworkDelay"   minOccurs="0" maxOccurs="1"/>
</sequence>
</complexType>
</element>
<element  name="HTTPNode">
<complexType  content="elementOnly">
<sequence>
<element name="OrgName"/>
<element name="HTTPAddress"/>
</sequence>
</complexType>
</element>
<element  name="SMTPNode">
<complexType  content="elementOnly">
<sequence>
<element name="OrgName"/>
<element name="SMTPAddress"/>
</sequence>
</complexType>
</element>
<element  name="VANEDINode">
<complexType  content="elementOnly">
<sequence>
<element name="OrgName"/>
<element name="VANEDIAddress"/>
</sequence>
</complexType>
</element>
<element  name="FTPNode">
<complexType  content="elementOnly">
<sequence>
<element name="OrgName"/>
<element name="FTPAddress"/>
<element name="TransferType"/>
<element name="PassiveMode"/>
<element name="ControlPort"/>
</sequence>
</complexType>
</element>
<element  name="OrgName">
<complexType content="empty">
      <attribute name="Partyname" type="IDREF"  minOccurs="1" />
</complexType>
</element>
<element  name="HTTPAddress">
```

```
<complexType  content="elementOnly">
<sequence>
<element name="LogonURL"   minOccurs="0" maxOccurs="1"/>
<element name="RequestURL"   minOccurs="0" maxOccurs="1"/>
<element name="ResponseURL"   minOccurs="0" maxOccurs="1"/>
</sequence>
</complexType>
</element>
<element  name="LogonURL">
<simpleType  name="LogonURL" base="string"/>
</element>
<element  name="RequestURL">
<simpleType  name="RequestURL" base="string"/>
</element>
<element  name="ResponseURL">
<simpleType  name="ResponseURL" base="string"/>
</element>
<element  name="VANEDIAddress">
<complexType  content="elementOnly">
<sequence>
<element name="InBox"/>
<element name="OutBox"/>
</sequence>
</complexType>
</element>
<element  name="InBox">
<simpleType  name="InBox" base="string"/>
</element>
<element  name="OutBox">
<simpleType  name="OutBox" base="string"/>
</element>
<element  name="SMTPAddress">
<simpleType  name="SMTPAddress" base="string"/>
</element>
<element  name="FTPAddress">
<complexType  content="elementOnly">
<sequence>
<element name="InDirectory"/>
<element name="OutDirectory"/>
</sequence>
</complexType>
</element>
<element  name="InDirectory">
<simpleType  name="InDirectory" base="string"/>
</element>
<element  name="OutDirectory">
<simpleType  name="OutDirectory" base="string"/>
</element>
<element  name="TransferType">
<simpleType  name="TransferType" base="string"/>
</element>
<element  name="PassiveMode">
<simpleType  name="PassiveMode" base="string"/>
</element>
<element  name="ControlPort">
<simpleType  name="ControlPort" base="string"/>
</element>
```

```
<element  name="TransportEncoding">
<simpleType  name="TransportEncoding" base="string"/>
</element>
<element  name="TransportTimeout">
<complexType  content="elementOnly">
<sequence>
<element name="Timeout"/>
<element name="Retries"/>
<element name="RetryInterval"/>
</sequence>
</complexType>
</element>
<element  name="Timeout">
<simpleType  name="Timeout" base="string"/>
</element>
<element  name="Retries">
<simpleType  name="Retries" base="string"/>
</element>
<element  name="RetryInterval">
<simpleType  name="RetryInterval" base="string"/>
</element>
<element  name="NetworkDelay">
<simpleType  name="NetworkDelay" base="string"/>
</element>
<element  name="TransportSecurity">
<complexType  content="elementOnly">
<sequence>
<element name="Encryption"   minOccurs="0" maxOccurs="1"/>
<element name="Authentication"   minOccurs="0" maxOccurs="1"/>
</sequence>
</complexType>
</element>
<element  name="Encryption">
<complexType  content="elementOnly">
<sequence>
<element name="Protocol"/>
<element name="Version"   minOccurs="0" maxOccurs="1"/>
<element name="Certificate"/>
</sequence>
</complexType>
</element>
<element  name="Authentication">
<complexType  content="elementOnly">
<sequence>
<element name="PasswordAuthentication"   minOccurs="0" maxOccurs="1"/>
<element name="CertificateAuthentication"   minOccurs="0" maxOccurs="1"/>
</sequence>
</complexType>
</element>
<element  name="PasswordAuthentication">
<complexType  content="elementOnly">
<sequence>
<element name="Protocol"/>
<element name="Version"   minOccurs="0" maxOccurs="1"/>
<element name="LogonInfo"   minOccurs="0" maxOccurs="1"/>
</sequence>
</complexType>
```

```
</element>
<element name="LogonInfo">
<complexType  content="elementOnly">
  <element name="LogonParty"  maxOccurs='*'/>
  </complexType>
  </element>
  <element  name="LogonParty">
  <complexType  content="elementOnly">
  <sequence>
  <element name="OrgName"/>
  <element name="UserId"/>
  <element name="Password"/>
  </sequence>
  </complexType>
  </element>
  <element  name="UserId">
  <simpleType  name="UserId" base="string"/>
  </element>
  <element  name="Password">
  <simpleType  name="Password" base="string"/>
  </element>
  <element  name="CertificateAuthentication">
  <complexType  content="elementOnly">
  <sequence>
  <element name="Protocol"/>
  <element name="Version"   minOccurs="0" maxOccurs="1"/>
  <element name="Certificate"/>
  </sequence>
  </complexType>
  </element>
  <element  name="Certificate">
  <complexType  content="elementOnly">
  <sequence>
  <element name="CertType"/>
  <element name="KeyLength"/>
  <element name="Party"  maxOccurs='*'/>
  </sequence>
  </complexType>
  </element>
  <element  name="CertType">
  <simpleType  name="CertType" base="string"/>
  </element>
  <element  name="KeyLength">
  <simpleType  name="KeyLength" base="string"/>
  </element>
  <element  name="Party">
  <complexType  content="elementOnly">
  <sequence>
  <element name="OrgName"/>
  <element name="OrgCertSource"   minOccurs="0" maxOccurs="1"/>
  <element name="IssuerOrgName"/>
  <element name="IssuerCertSource"/>
  </sequence>
  </complexType>
  </element>
  <element  name="OrgCertSource">
  <simpleType  name="OrgCertSource" base="string"/>
```

```
  </element>
  <element  name="IssuerOrgName">
  <simpleType  name="IssuerOrgName" base="string"/>
  </element>
  <element  name="IssuerCertSource">
  <simpleType  name="IssuerCertSource" base="string"/>
  </element>
<element  name="DocExchange">
<complexType  content="elementOnly">
<sequence>
<element name="MessageEncoding"   minOccurs="0" maxOccurs="1"/>
<element name="MessageIdempotency"/>
<element name="MessageRetries"   minOccurs="0" maxOccurs="1"/>
<element name="MessageSecurity"   minOccurs="0" maxOccurs="1"/>
</sequence>
      <attribute name="DocExchangeId" type="ID"  minOccurs="1" />
</complexType>
</element>
<element  name="MessageEncoding">
<simpleType  name="MessageEncoding" base="string"/>
</element>
<element  name="MessageIdempotency">
<simpleType  name="MessageIdempotency" base="string"/>
</element>
<element  name="MessageRetries">
<complexType  content="elementOnly">
<sequence>
<element name="Retries"/>
<element name="RetryInterval"/>
</sequence>
</complexType>
</element>
<element  name="MessageSecurity">
<complexType  content="elementOnly">
<sequence>
<element name="NonRepudiation"   minOccurs="0" maxOccurs="1"/>
<element name="DigitalEnvelope"   minOccurs="0" maxOccurs="1"/>
</sequence>
</complexType>
</element>
<element  name="NonRepudiation">
<complexType  content="elementOnly">
<sequence>
<element name="Protocol"/>
<element name="Version"   minOccurs="0" maxOccurs="1"/>
<element name="HashFunction"/>
<element name="EncryptionAlgorithm"/>
<element name="SignatureAlgorithm"/>
<element name="Certificate"/>
</sequence>
</complexType>
</element>
<element  name="HashFunction">
<simpleType  name="HashFunction" base="string"/>
</element>
<element  name="EncryptionAlgorithm">
<simpleType  name="EncryptionAlgorithm" base="string"/>
```

```
</element>
<element  name="SignatureAlgorithm">
<simpleType  name="SignatureAlgorithm" base="string"/>
</element>
<element  name="DigitalEnvelope">
<complexType  content="elementOnly">
<sequence>
<element name="Protocol"   minOccurs="0" maxOccurs="1"/>
<element name="Version"   minOccurs="0" maxOccurs="1"/>
<element name="EncryptionAlgorithm"/>
<element name="Certificate"   minOccurs="0" maxOccurs="1"/>
</sequence>
</complexType>
</element>
<element  name="BusinessProtocol">
<complexType  content="elementOnly">
<sequence>
<element name="MessageSet"/>
<element name="BusinessInterface"/>
</sequence>
</complexType>
</element>
<element  name="MessageSet">
<simpleType  name="MessageSet" base="string"/>
</element>
<element  name="BusinessInterface">
<complexType  content="elementOnly">
<element name="ServiceInterface"  maxOccurs='*'/>
</complexType>
</element>
<element  name="ServiceInterface">
<complexType  content="elementOnly">
<sequence>
<element name="OrgName"/>
<element name="TaskName"   minOccurs="0" maxOccurs="1"/>
<element name="ActionMenu"/>
<element name="ServerServiceTime"   minOccurs="0" maxOccurs="1"/>
<element name="StartEnabled"   minOccurs="0" maxOccurs="1"/>
<element name="TerminateConversation"   minOccurs="0" maxOccurs="1"/>
<element name="ServiceSecurity"   minOccurs="0" maxOccurs="1"/>
</sequence>
      <attribute name="InterfaceId" type="string"  minOccurs="1" />
</complexType>
</element>
<element  name="TaskName">
<simpleType  name="TaskName" base="string"/>
</element>
<element  name="ActionMenu">
<complexType  content="elementOnly">
<element name="Action"  maxOccurs='*'/>
</complexType>
</element>
<element  name="Action">
<complexType  content="elementOnly">
<sequence>
<element name="Request"/>
<element name="Response"   minOccurs="0" maxOccurs="1"/>
```

```
<element name="ExceptionResponse"   minOccurs="0" maxOccurs="1"/>
<element name="ResponseServiceTime"   minOccurs="0" maxOccurs="1"/>
<element name="Sequencing"   minOccurs="0" maxOccurs="1"/>
<element name="ActionSecurity"   minOccurs="0" maxOccurs="1"/>
</sequence>
      <attribute name="Type" type="NMTOKEN"  minOccurs="0" default="basic" >
 <simpleType base="NMTOKEN">
 <enumeration value='basic|concurrent' />
 </simpleType>
 </attribute>
      <attribute name="ActionId" type="string"  minOccurs="1" />

      <attribute name="Invocation" type="NMTOKEN"  minOccurs="0"
default="asyncOnly" >
 <simpleType base="NMTOKEN">
 <enumeration value='syncOnly|asyncOnly' />
 </simpleType>
 </attribute>
 </complexType>
 </element>
 <element  name="Request">
 <complexType  content="elementOnly">
 <sequence>
 <element name="RequestName"/>
 <element name="RequestMessage"/>
 <element name="Channel"   minOccurs="0" maxOccurs="1"/>
 </sequence>
 </complexType>
 </element>
 <element  name="RequestName">
 <simpleType  name="RequestName" base="string"/>
 </element>
 <element  name="RequestMessage">
 <simpleType  name="RequestMessage" base="string"/>
 </element>
 <element  name="Channel">
 <complexType content="empty">
      <attribute name="ChannelId" type="IDREF"  minOccurs="1" />
</complexType>
</element>
<element  name="Response">
<complexType  content="elementOnly">
<element name="ResponseName"  maxOccurs='*'/>
</complexType>
</element>
<element  name="ResponseName">
<simpleType  name="ResponseName" base="string"/>
</element>
<element  name="ExceptionResponse">
<complexType  content="elementOnly">
<element name="ExceptionResponseName"  maxOccurs='*'/>
</complexType>
</element>
<element  name="ExceptionResponseName">
<simpleType  name="ExceptionResponseName" base="string"/>
</element>
<element  name="ResponseServiceTime">
```

```
<complexType  content="elementOnly">
<sequence>
<element name="ServiceTime"/>
<element name="Presume"   minOccurs="0" maxOccurs="1"/>
</sequence>
</complexType>
</element>
<element  name="ServiceTime">
<simpleType  name="ServiceTime" base="string"/>
</element>
<element  name="Presume">
<simpleType  name="Presume" base="string"/>
</element>
<element  name="Sequencing">
<complexType  content="elementOnly">
<sequence>
<element name="Enable"   minOccurs="0" maxOccurs="1"/>
<element name="Disable"   minOccurs="0" maxOccurs="1"/>
</sequence>
</complexType>
</element>
<element  name="Enable">
<complexType  content="elementOnly">
<element name="RequestName"  maxOccurs='*'/>
</complexType>
</element>
<element  name="Disable">
<complexType  content="elementOnly">
<element name="RequestName"  maxOccurs='*'/>
</complexType>
</element>
<element  name="ActionSecurity">
<simpleType  name="ActionSecurity" base="string"/>
</element>
<element  name="ServerServiceTime">
<complexType  content="elementOnly">
<sequence>
<element name="ServiceTime"/>
<element name="Presume"   minOccurs="0" ma xOccurs="1"/>
</sequence>
</complexType>
</element>
<element  name="StartEnabled">
<complexType  content="elementOnly">
<element name="RequestName"  maxOccurs='*'/>
</complexType>
</element>
<element  name="TerminateConversation">
<complexType  content="elementOnly">
<element name="RequestName"  maxOccurs='*'/>
</complexType>
</element>
<element  name="ServiceSecurity">
<simpleType  name="ServiceSecurity" base="string"/>
</element>
<element  name="Comment">
<simpleType  name="Comment" base="string"/>
```

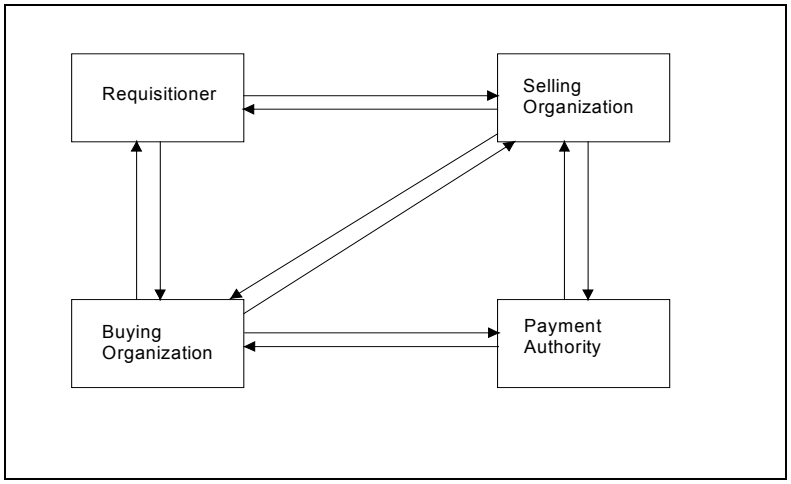
```
</element>
</schema>
```

Appendix 4 Example: OBI

In this section, we describe an OBI scenario. A sample TPA for this example is in Appendix "4.1 TPA for Open Buying on the Internet (OBI)".

Open Buying on the Internet™ (OBI) is a protocol for business-to-business Internet commerce. It was designed by the Internet Purchasing Roundtable and is supported by the OBI consortium (http://www.openbuy.org). OBI defines the procedures for the high-volume, low-dollar purchasing transactions that make up most of an organization's purchasing activity.

The following figure illustrates the participants in an OBI transaction and the basic information flows.



The requisitioner is a member of the buying organization (e.g. an employee of a company) and is permitted to place orders directly with the selling organization's merchant server. The requisitioner can browse a catalog and place an order with the selling organization using a browser. The catalog may be tailored to the requisitioner's organization. When the requisitioner has placed an order, the selling organization's server sends a partial purchase order (purchase order request) to the buying organization's server. The buying organization validates the purchase order request and transforms it into a complete purchase order that it returns to the selling organization. The selling organization then prepares an invoice or otherwise arranges for payment and ships the ordered merchandise. The payment authority is an optional part of the system. Its purpose is to handle electronic payments. Using the browser, the requisitioner can also view and update various information at the buying organization server such as the requisitioner's profile, outstanding requests, etc. The requisitioner can also check the status of an order at the selling organization.

An additional possibility is that the buying organization can send an "unsolicited" purchase order to the selling organization without a prior request and partial purchase order initiated by a requisitioner.

There is a TPA between the buying organization and the selling organization, each of which has a B2B server. The payment authority, if present, is outside the scope of the 2-party TPA between the buying organization and the selling organization. It may interact with the buying organization and the selling organization in a variety of ways. The interaction may be through separate 2-party TPAs between the payment authority and the buyer and seller organizations. It may also be simply through application programs without involving the B2B server.

Appendix 4.1 TPA for Open Buying on the Internet (OBI)

Following is the TPA for the OBI example.

NOTE: To use this example with an XML Schema parser, remove the DOCTYPE line.

```
<?xml version="1.0"?>
<!-- (C)Copyright IBM Corporation 2000; all rights reserved              -->
<!DOCTYPE TPA SYSTEM "TPA.dtd" >
<!--************************************************************************-->
<!--              OBI TPA      between Large Co (buying company)          -->
<!--                     and Pens Are We (selling company)                 -->
<!--************************************************************************-->
<TPA xmlns="tpa.xsd">
<!--************************************************************************-->
<!-- The xmlns attribute is for use with an XML Schema parser.  Its       -->
<!-- definition in the DTD causes it to be ignored by a DTD parser.       -->
<!--************************************************************************-->
<!-- General information                                                  -->
<!--************************************************************************-->
  <TPAInfo>
    <TPAName>OBIStandard</TPAName>
  <TPAType>
    <Protocol>OBI</Protocol>
    <Version>1.0</Version>
    <Type>SS</Type>
   </TPAType>
<!--************************************************************************-->
    <Participants>
<!--************************************************************************-->
<!-- Specification of Buyer                                               -->
<!--************************************************************************-->
      <Member IdCodeType="ZZ" MemberId="777777777777777">
        <PartyName Partyname="LargeCo">Large Co</PartyName>
        <CompanyTelephone>914-945-3000</CompanyTelephone>
        <Address>
          <AddressType>location</AddressType>
          <AddressLine>Large Co</AddressLine>
          <AddressLine>HQ Building</AddressLine>
          <AddressLine>1 Main Street</AddressLine>
          <City>SmallTown</City>
          <State>NY</State>
          <Zip>10000</Zip>
          <Country>USA</Country>
        </Address>
        <Address>
          <AddressType>billing</AddressType>
          <AddressLine>Large Co</AddressLine>
          <AddressLine>Accounting Department</AddressLine>
          <AddressLine>100 Bean Counters Road</AddressLine>
          <City>Any City</City>
          <State>CT</State>
```

```
          <Zip>06000</Zip>
          <Country>USA</Country>
        </Address>
        <Address>
          <AddressType>shipping</AddressType>
          <AddressLine>Large Co</AddressLine>
          <AddressLine>Procurement Department</AddressLine>
          <AddressLine>99 Purchase Road</AddressLine>
          <City>Buy City</City>
          <State>NY</State>
          <Zip>10001</Zip>
          <Country>USA</Country>
        </Address>
        <Contact Type = "primary">
          <LastName>Smith</LastName>
          <FirstName>John</FirstName>
          <MiddleName>L.</MiddleName>
          <Title>Senior Buyer</Title>
          <ContactTelephone Type = "primary">914-111-6789
            </ContactTelephone>
          <ContactTelephone Type = "secondary">914-111-6790
            </ContactTelephone>
          <EMail Type = "primary">jjsmith@largeco.com</EMail>
          <EMail Type = "secondary">
            http://www.largeco.com/procurement/jsmith.html
            </EMail>
          <Fax>914-111-6780</Fax>
        </Contact>
        <Contact Type = "secondary">
          <LastName>Blow</LastName>
          <FirstName>Joe</FirstName>
          <MiddleName>J.</MiddleName>
          <Title>Buyer</Title>
          <ContactTelephone Type = "primary">914-111-6722
            </ContactTelephone>
          <ContactTelephone Type = "secondary">914-111-6725
            </ContactTelephone>
          <EMail Type = "primary">jblow@largeco.com</EMail>
          <Fax>914-111-6780</Fax>
        </Contact>
      </Member>
<!--************************************************************************-->
<!-- Specification of Seller                                                -->
<!--************************************************************************-->
      <Member IdCodeType="ZZ" MemberId="888000009000000">
        <PartyName Partyname="PensAreWe">Pens Are We
            </PartyName>
        <CcompanyTelephone>945-123-1000</CcompanyTelephone>
        <Address>
          <AddressType>location</AddressType>
          <AddressLine>Pens Are We</AddressLine>
          <AddressLine>Building 001</AddressLine>
          <AddressLine>123 High Street</AddressLine>
          <City>EarthQuake City</City>
          <State>CA</State>
          <Zip>94567</Zip>
          <Country>USA</Country>
        </Address>
        <Contact Type = "primary">
          <LastName>Doe</LastName>
          <FirstName>Jane</FirstName>
          <MiddleName>E.</MiddleName>
          <Title>Vice President of Internet Sales</Title>
          <ContactTelephone Type = "primary">945-123-4567
```

```
            </ContactTelephone>
           <ContactTelephone Type = "secondary">945-123-4570
            </ContactTelephone>
           <EMail Type = "primary">janedoe@pensarewe.com</EMail>
           <EMail Type = "secondary">
             http://www.pensarewe.com/sales/jdoe.html
             </EMail>
           <Fax>945-123-9999</Fax>
         </Contact>
       </Member>
      </Participants>
<!--*************************************************************************-->
<!-- Specification of Arbitrator                                             -->
<!--*************************************************************************-->
    <Arbitrator IdCodeType="01" MemberId="888000009000001">
      XYZArbitrator</Arbitrator>
    <Duration>
      <Start>
        <Date>01/01/1999</Date>
        <Time>00:00:00</Time>
      </Start>
      <End>
        <Date>01/01/2001</Date>
        <Time>00:00:00</Time>
      </End>
    </Duration>
    <InvocationLimit>100000</InvocationLimit>
    <ConcurrentConversations>1</ConcurrentConversations>
    <ConversationLife>86400</ConversationLife>
  </TPAInfo>
<!--*************************************************************************-->
<!-- Specification of Delivery Channel Set                                   -->
<!--*************************************************************************-->
<DeliveryChannelSet>
      <DeliveryChannel
            ChannelId="channel01"
            TransportId="transport01"
            DocExchangeId="docexchange01" >
            </DeliveryChannel>
      </DeliveryChannelSet>
<!--*************************************************************************-->
<!-- Specification of Transport Protocol #01                                 -->
<!--*************************************************************************-->
  <Transport TransportId="transport01">
   <Communication>
    <HTTP>
     <HTTPNode>
      <OrgName Partyname="LargeCo"/>
      <HTTPAddress>
        <RequestURL>
            https://www.largeco.com/jackal/servlet/OBIBuy</RequestURL>
      </HTTPAddress>
     </HTTPNode>
     <HTTPNode>
      <OrgName Partyname="PensAreWe"/>
      <HTTPAddress>
        <LogonURL>
            https://www.pensarewe.com/coyote/servlet/OBILogon</LogonURL>
        <RequestURL>
            https://www.pensarewe.com/coyote/servlet/OBIsell</RequestURL>
        <ResponseURL>
            https://www.pensarewe.com/coyote/servlet/OBIsell</ResponseURL>
      </HTTPAddress>
     </HTTPNode>
```

```
    <NetworkDelay>300</NetworkDelay>
    </HTTP>
   </Communication>
<!--*********************************************************************-->
<!-- Specification of Transport Security Protocol                        -->
<!--*********************************************************************-->
   <TransportSecurity>
      <Encryption>
        <Protocol>SSL</Protocol>
        <Version>3.0</Version>
        <Certificate>
          <CertType>X509.V3</CertType>
          <KeyLength>1024</KeyLength>
          <Party>
            <OrgName Partyname="LargeCo"/>
            <IssuerOrgName>VeriSign, Inc.</IssuerOrgName>
            <IssuerCertSource>http://www.verisign.com/certs
                  </IssuerCertSource>
          </Party>
          <Party>
            <OrgName Partyname="PensAreWe"/>
            <IssuerOrgName>GTE, Inc.</IssuerOrgName>
            <IssuerCertSource>http://www.gte.com/certs
                  </IssuerCertSource>
          </Party>
        </Certificate>
      </Encryption>
      <Authentication>
        <CertificateAuthentication>
        <Protocol>SSL</Protocol>
        <Version>3.0</Version>
        <Certificate>
          <CertType>X509.V3</CertType>
          <KeyLength>1024</KeyLength>
          <Party>
            <OrgName Partyname="LargeCo"/>
            <IssuerOrgName>VeriSign, Inc.</IssuerOrgName>
            <IssuerCertSource>http://www.verisign.com/certs
                  </IssuerCertSource>
          </Party>
          <Party>
            <OrgName Partyname="PensAreWe"/>
            <IssuerOrgName>GTE, Inc.</IssuerOrgName>
            <IssuerCertSource>http://www.gte.com/certs
                  </IssuerCertSource>
          </Party>
        </Certificate>
       </CertificateAuthentication>
      </Authentication>
     </TransportSecurity>
    </Transport>
<!--*********************************************************************-->
<!-- Specification of DocExchange Protocol                               -->
<!--*********************************************************************-->
  <DocExchange DocExchangeId="docexchange01">
    <MessageEncoding>BASE64</MessageEncoding>
    <MessageIdempotency>yes</MessageIdempotency>
<!--*********************************************************************-->
<!-- Specification of Message Security                                   -->
<!--*********************************************************************-->
    <MessageSecurity>
      <NonRepudiation>
        <Protocol>DigitalSignature</Protocol>
        <HashFunction>MD5</HashFunction>
```

```
        <EncryptionAlgorithm>RSA</EncryptionAlgorithm>
        <SignatureAlgorithm>DSA</SignatureAlgorithm>
        <Certificate>
          <CertType>X509.V3</CertType>
          <KeyLength>1024</KeyLength>
          <Party>
            <OrgName Partyname="LargeCo"/>
            <IssuerOrgName>Verisign Inc.</IssuerOrgName>
            <IssuerCertSource>http://www.verisign.com/certs
                  </IssuerCertSource>
          </Party>
          <Party>
            <OrgName Partyname="PensAreWe"/>
            <IssuerOrgName>GTE Inc.</IssuerOrgName>
            <IssuerCertSource>http://www.gte.com/certs</IssuerCertSource>
          </Party>
        </Certificate>
      </NonRepudiation>
    </MessageSecurity>
  </DocExchange>
  <BusinessProtocol>
  <MessageSet>OBIMessages</MessageSet>
  <BusinessInterface>
<!--************************************************************************-->
<!-- Specification of Service Interface 01 - Buying organization          -->
<!--************************************************************************-->
   <ServiceInterface InterfaceId="interface01">
    <OrgName Partyname="LargeCo"/>
    <ActionMenu>
      <Action ActionId="action01" Type="basic">
        <Request>
          <RequestName>putOPOR</RequestName>
          <RequestMessage>OBIPOR</RequestMessage>
        <Channel ChannelId="channel01"/>
        </Request>
        <Response>
          <ResponseName>putOPO</ResponseName>
          </Response>
        <ResponseServiceTime>
         <ServiceTime>3600</ServiceTime>
         <Presume>fail</Presume>
         </ResponseServiceTime>
      </Action>
    </ActionMenu>
    <ServerServiceTime>
        <ServiceTime>3660</ServiceTime>
        <Presume>fail</Presume>
    </ServerServiceTime>
    <StartEnabled>
      <RequestName>putOPOR</RequestName>
    </StartEnabled>
   </ServiceInterface>
<!--************************************************************************-->
<!-- Specification of Service Interface 02 - Selling organization         -->
<!-- This interface below is for initiating shopping and for solicited or -->
<!-- UnSolicited OBIPO from                                               -->
<!-- buying organization to selling organization                          -->
<!--************************************************************************-->
  <ServiceInterface InterfaceId="interface02">
    <OrgName Partyname="PensAreWe"/>
    <ActionMenu>
      <Action ActionId="action03" Type="basic">
        <Request>
          <RequestName>shop</RequestName>
```

```
          <!--Initiates shopping at merchant server-->
          <RequestMessage>shopMessage</RequestMessage>
        </Request>
      </Action>
      <Action ActionId="action02" Type="basic">
        <Request>
          <RequestName>putOPO</RequestName>
             <!--receives solicited or unsolicited PO -->
          <RequestMessage>OBIPO</RequestMessage>
        </Request>
      </Action>
    </ActionMenu>
    <ServerServiceTime>
        <ServiceTime>3660</ServiceTime>
        <Presume>fail</Presume>
    </ServerServiceTime>
   </ServiceInterface>
   </BusinessInterface>
  </BusinessProtocol>
</TPA>
```

Appendix 5 Examples of Reusable and Fully Qualified TPAs

Following is an example of a reusable TPA using roles. First is part of the OBI TPA example written with roles. This is followed by the same example in which the roles have been evaluated into specific parties and the party-specific information filled in making a fully qualified TPA.

Appendix 5.1 Reusable TPA

In the reusable TPA, role names appear wherever party names would appear in a completed TPA. The values of the associated party-specific tags are blank.

```
<?xml version="1.0"?>
<!-- (C)Copyright IBM Corporation 2000; all rights reserved               -->
<!DOCTYPE TPA SYSTEM "TPA.dtd" >
<!--************************************************************************-->
<!--              OBI TPA between Large Co (buying company)               -->
<!--                      and Pens Are We (selling company)               -->
<!--************************************************************************-->
<TPA xmlns="tpa.xsd">
<!--************************************************************************-->
<!-- General TPA information                                               -->
<!--************************************************************************-->
  <TPAInfo>
    <TPAName>OBIStandard</TPAName>
    <TPAType>
      <Protocol>OBI</Protocol>
      <Version>1.0</Version>
      <Type>SS</Type>
    </TPAType>
<!--************************************************************************-->
<!--Role parameters                                                         -->
<!--************************************************************************-->
    <Role>
      <RoleDefn>
        <RoleName>OBIBuyer</RoleName>
        <RolePlayer> </RolePlayer>
       </RoleDefn>
      <RoleDefn>
        <RoleName>OBISeller</RoleName>
        <RolePlayer> </RolePlayer>
       </RoleDefn)
      </Role>
    <Participants>
<!--************************************************************************-->
<!-- Specification of Buyer                                                -->
<!--************************************************************************-->
      <Member IdCodeType=" " MemberId=" ">
        <PartyName Partyname="obibuyer">OBIBuyer</PartyName>
        <CompanyTelephone> </CompanyTelephone>
        <Address>
          <AddressType>location</AddressType>
          <AddressLine> </AddressLine>
          <AddressLine> </AddressLine>
          <AddressLine> </AddressLine>
          <City> </City>
          <State> </State>
          <Zip> </Zip>
          <Country </Country>
        </Address>
```

```
        <Address>
          <AddressType>billing</AddressType>
          <AddressLine> </AddressLine>
          <AddressLine> </AddressLine>
          <AddressLine> </AddressLine>
          <City> </City>
          <State> </State>
          <Zip> </Zip>
          <Country> </Country>
        </Address>
        <Address>
          <AddressType>shipping</AddressType>
          <AddressLine> </AddressLine>
          <AddressLine> </AddressLine>
          <AddressLine> </AddressLine>
          <City> </City>
          <State> </State>
          <Zip> </Zip>
          <Country> </Country>
        </Address>
        <Contact Type = "primary">
          <LastName> </LastName>
          <FirstName> < MiddleName>
          <Title> </Title>
          <ContactTelephone Type = "primary"> </ContactTelephone>
          <ContactTelephone Type = "secondary"> </ContactTelephone>
          <EMail Type = "primary"> </EMail>
          <EMail Type = "secondary"> </EMail>
          <Fax> </Fax>
        </Contact>
        <Contact Type = "secondary">
          <LastName> </LastName>
          <FirstName> </FirstName>
          <MiddleName> </MiddleName>
          <Title> </Title>
          <ContactTelephone Type = "primary"> </ContactTelephone>
          <ContactTelephone Type = "secondary"> </ContactTelephone>
          <EMail Type = "primary"> </EMail>
          <Fax> </Fax>
        </Contact>
      </Member>
<!--*************************************************************************-->
<!-- Specification of Seller                                                 -->
<!--*************************************************************************-->
      <Member IdCodeType=" " MemberId=" ">
        <PartyName Partyname="obiseller">OBISeller/PartyName>
        <CompanyTelephone> </CompanyTelephone>
        <Address>
          <AddressType> </AddressType>
          <AddressLine> </AddressLine>
          <AddressLine> </AddressLine>
          <AddressLine> </AddressLine>
          <City> </City>
          <State> </State>
          <Zip> </Zip>
          <Country> </Country>
        </Address>
        <Contact Type = "primary">
          <LastName> </LastName>
          <FirstName> </FirstName>
          <MiddleName> </MiddleName>
          <Title> </Title>
          <ContactTelephone Type = "primary"> </ContactTelephone>
          <ContactTelephone Type = "secondary"> </ContactTelephone>
```

```
          <EMail Type = "primary"> </EMail>
          <EMail Type = "secondary"> </EMail>
          <Fax> </Fax>
        </Contact>
      </Member>
    </Participants>
  </TPAInfo>

...

<!--************************************************************************-->
<!-- Specification of Transport Protocol                                    -->
<!--************************************************************************-->
  <Transport TransportId="transport01">
   <Communication>
    <HTTP>
     <HTTPNode>
      <OrgName Partyname="obibuyer"/>
      <HTTPAddress>
        <RequestURL> </RequestURL>
      </HTTPAddress>
...

     </HTTPNode>
     <HTTPNode>
      <OrgName Partyname="obiseller"/>
      <HTTPAddress>
        <LogonURL> </LogonURL>
        <RequestURL> </RequestURL>
        <ResponseURL> </ResponseURL>
      </HTTPAddress>
...
     </HTTPNode>
    <NetworkDelay>300</NetworkDelay>
    </HTTP>
   </Communication>
<!--************************************************************************-->
<!-- Specification of Transport Security Protocol                           -->
<!--************************************************************************-->
   <TransportSecurity>
      <Encryption>
       ...
        <Certificate>
...

          <Party>
            <OrgName Partyname="obibuyer"/>
            <IssuerOrgName> </IssuerOrgName>
          </Party>
          <Party>
            <OrgName Partyname="obiseller"/>
            <IssuerOrgName> </IssuerOrgName>
          </Party>
        </Certificate>
...
      <Authentication>
...
        <Certificate>
...
          <Party>
            <OrgName Partyname="obibuyer"/>
            <IssuerOrgName> </IssuerOrgName>
          </Party>
          <Party>
```

```
            <OrgName Partyname="obiseller"/>
            <IssuerOrgName> </IssuerOrgName>
          </Party>
        </Certificate>
       </CertificateAuthentication>
      </Authentication>
    </TransportSecurity>
   </Transport>
<!--*********************************************************************-->
<!-- Specification of DocExchange                                        -->
<!--*********************************************************************-->

   <DocExchange DocExchangeID="docexchange01">
...
<!--*********************************************************************-->
<!-- Specification of Message Security                                   -->
<!--*********************************************************************-->

    <MessageSecurity>
      <NonRepudiation>

...

        <Certificate>
          <Party>

...

            <OrgName Partyname="obibuyer"/>
            <IssuerOrgName> </IssuerOrgName>
          </Party>
          <Party>
            <OrgName Partyname="obiseller"/>
            <IssuerOrgName> </IssuerOrgName>
          </Party>
        </Certificate>
      </NonRepudiation>
    </MessageSecurity>
  </DocExchange>>

...

  <BusinessProtocol>
  <MessageSet>OBIMessages</MessageSet>
  <BusinessInterface>
<!--*********************************************************************-->
<!-- Specification of Service Interface 01 - Buying organization         -->
<!--*********************************************************************-->
  <ServiceInterface InterfaceId="interface01">
    <OrgName Partyname="obibuyer"/>
    <ActionMenu>
      <Action ActionId="action01">

...

        <ResponseServiceTime>
          <ServiceTime> </ServiceTime>

...

        <ResponseServiceTime>
      </Action>
    </ActionMenu>
    <ServerServiceTime>
```

```
      <ServiceTime> </ServiceTime>

...

    </ServerServiceTime>

...

  </ServiceInterface>
<!--************************************************************************-->
<!-- Specification of Service Interface 02 - Selling organization         -->
<!-- This interface below is for initiating shopping and for solicited or -->
<!-- UnSolicited OBIPO from                                               -->
<!-- buying organization to selling organization                          -->
<!--************************************************************************-->
  <ServiceInterface InterfaceId="interface02">
    <OrgName Partyname="obiseller"/>
    <ActionMenu>
      <Action ActionId="action02">
...

     </Action>
    </ActionMenu>
    <ServerServiceTime>
      <ServiceTime> </ServiceTime>

...

    </ServerServiceTime>
...

  </ServiceInterface>
  </BusinessInterface>
  </BusinessProtocol>
</TPA>
```

Appendix 5.2 Fully Qualified TPA

Following is a fully qualified TPA, the result of replacing the role names in the reusable TPA in the previous section with specific party names and filling in the party-specific information in various tags. Note that the <Role> tag and its subelements have been left in the fully qualified TPA for reference although they have no further function.

```
<?xml version="1.0"?>
<!-- (C)Copyright IBM Corporation 2000; all rights reserved              -->
<!DOCTYPE TPA SYSTEM "TPA.dtd" >
<!--************************************************************************-->
<!--              OBI TPA between Large Co (buying company)               -->
<!--                      and Pens Are We (selling company)               -->
<!--************************************************************************-->
<TPA xmlns="tpa.xsd">
<!--************************************************************************-->
<!-- General TPA information                                              -->
<!--************************************************************************-->
  <TPAInfo>
    <TPAName>OBIStandard</TPAName>
    <TPAType>
      <Protocol>OBI</Protocol>
      <Version>1.0</Version>?
      <Type>SS</Type>
```

```
    </TPAType>
<!--*************************************************************************-->
<!--Role parameters                                                           -->
<!--*************************************************************************-->
    <Role>
      <RoleDefn>
        <RoleName>OBIBuyer</RoleName>
        <RolePlayer>Large Co</RolePlayer>
       </RoleDefn>
      <RoleDefn>
        <RoleName>OBISeller</RoleName>
        <RolePlayer>Pens Are We</RolePlayer>
       </RoleDefn)
      </Role>
    <Participants>
<!--*************************************************************************-->
<!-- Specification of Buyer                                                  -->
<!--*************************************************************************-->
      <Member IdCodeType="ZZ" MemberId="777777777777777">
        <PartyName Partyname="obibuyer">Large Co</PartyName>
        <CompanyTelephone>914-945-3000</CompanyTelephone>
        <Address>
          <AddressType>location</AddressType>
          <AddressLine>Large Co</AddressLine>
          <AddressLine>HQ Building</AddressLine>
          <AddressLine>1 Main Street</AddressLine>
          <City>SmallTown</City>
          <State>NY</State>
          <Zip>10000</Zip>
          <Country>USA</Country>
        </Address>
        <Address>
          <AddressType>billing</AddressType>
          <AddressLine>Large Co</AddressLine>
          <AddressLine>Accounting Department</AddressLine>
          <AddressLine>100 Bean Counters Road</AddressLine>
          <City>Any City</City>
          <State>CT</State>
          <Zip>06000</Zip>
          <Country>USA</Country>
        </Address>
        <Address>
          <AddressType>shipping</AddressType>
          <AddressLine>Large Co</AddressLine>
          <AddressLine>Procurement Department</AddressLine>
          <AddressLine>99 Purchase Road</AddressLine>
          <City>Buy City</City>
          <State>NY</State>
          <Zip>10001</Zip>
          <Country>USA</Country>
        </Address>
        <Contact Type = "primary">
          <LastName>Smith</LastName>
          <FirstName>John</FirstName>
          <MiddleName>L.</MiddleName>
          <Title>Senior Buyer</Title>
          <ContactTelephone Type = "primary">914-111-6789</ContactTelephone>
          <ContactTelephone Type = "secondary">914-111-6790</ContactTelephone>
          <EMail Type = "primary">jjsmith@largeco.com</EMail>
          <EMail Type = "secondary">
            http://www.largeco.com/procurement/jsmith.html
            </EMail>
          <Fax>914-111-6780</Fax>
        </Contact>
```

```
        <Contact Type = "secondary">
          <LastName>Blow</LastName>
          <FirstName>Joe</FirstName>
          <MiddleName>J.</MiddleName>
          <Title>Buyer</Title>
          <ContactTelephone Type = "primary">914-111-6722</ContactTelephone>
          <ContactTelephone Type = "secondary">914-111-6725</ContactTelephone>
          <EMail Type = "primary">jblow@largeco.com</EMail>
          <Fax>914-111-6780</Fax>
        </Contact>
      </Member>
<!--*********************************************************************-->
<!-- Specification of Seller                                             -->
<!--*********************************************************************-->
      <Member IdCodeType="ZZ" MemberId="888000009000000">
        <PartyName Partyname="obiseller">Pens Are We</PartyName>
        <CompanyTelephone>945-123-1000</CompanyTelephone>
        <Address>
          <AddressType>location</AddressType>
          <AddressLine>Pens Are We</AddressLine>
          <AddressLine>Building 001</AddressLine>
          <AddressLine>123 High Street</AddressLine>
          <City>EarthQuake City</City>
          <State>CA</State>
          <Zip>94567</Zip>
          <Country>USA</Country>
        </Address>
        <Contact Type = "primary">
          <LastName>Doe</LastName>
          <FirstName>Jane</FirstName>
          <MiddleName>E.</MiddleName>
          <Title>Vice President of Internet Sales</Title>
          <ContactTelephone Type = "primary">945-123-4567</ContactTelephone>
          <ContactTelephone Type = "secondary">945-123-4570</ContactTelephone>
          <EMail Type = "primary">janedoe@pensarewe.com</EMail>
          <EMail Type = "secondary">
            http://www.pensarewe.com/sales/jdoe.html
            </EMail>
          <Fax>945-123-9999</Fax>
        </Contact>
      </Member>
    </Participants>
  </TPAInfo>

...

<!--*********************************************************************-->
<!-- Specification of Transport Protocol #01                             -->
<!--*********************************************************************-->
  <Transport TransportId="transport01">
   <Communication>
    <HTTP>
     <HTTPNode>
      <OrgName Partyname="obibuyer"/>
      <HTTPAddress>
        <RequestURL>
            https://www.largeco.com/jackal/servlet/OBIBuy</RequestURL>
      </HTTPAddress>
...
     </HTTPNode>
     <HTTPNode>
      <OrgName Partyname="obiseller"/>
      <HTTPAddress>
        <LogonURL>
```

```
            https://www.pensarewe.com/BPF/servlet/OBILogon</LogonURL>
          <RequestURL>
            https://www.pensarewe.com/BPF/servlet/OBIsell</RequestURL>
          <ResponseURL>
            https://www.pensarewe.com/BPF/servlet/OBIsell</ResponseURL>
        </HTTPAddress>

...
    </HTTPNode>
    <NetworkDelay>300</NetworkDelay>
    </HTTP>
   </Communication>
<!--*************************************************************************-->
<!-- Specification of Transport Security Protocol                            -->
<!--*************************************************************************-->
   <TransportSecurity>
      <Encryption>
...
        <Certificate>
...

          <Party>
            <OrgName Partyname="obibuyer"/>
            <IssuerOrgName>VeriSign, Inc.</IssuerOrgName>
          </Party>
          <Party>
            <OrgName Partyname="obiseller"/>
            <IssuerOrgName>GTE, Inc.</IssuerOrgName>
          </Party>
         </Certificate>
        </Encryption>
       <Authentication>

...

          <Party>
            <OrgName Partyname="obibuyer"/>
            <IssuerOrgName>VeriSign, Inc.</IssuerOrgName>
          </Party>
          <Party>
            <OrgName Partyname="obiseller"/>
            <IssuerOrgName>GTE, Inc.</IssuerOrgName>
          </Party>
        </Certificate>
      </Authentication>
     </CertificateAuthentication>
    </TransportSecurity>
   </Transport>
<!--*************************************************************************-->
<!-- Specification of DocExchange                                            -->
<!--*************************************************************************-->
   <DocExchange DocExchangeId="docexchange01">
...

<!--*************************************************************************-->
<!-- Specification of Message Security                                       -->
<!--*************************************************************************-->
    <MessageSecurity>
      <NonRepudiation>

...

        <Certificate>
          <Party>
```

```
...

            <OrgName Partyname="obibuyer"/>
            <IssuerOrgName>Verisign Inc.</IssuerOrgName>
          </Party>
          <Party>
            <OrgName Partyname="obiseller"/>
            <IssuerOrgName>GTE Inc.</IssuerOrgName>
          </Party>
        </Certificate>
      </NonRepudiation>
    </MessageSecurity>
  </DocExchange>

...
  <BusinessProtocol>
  <MessageSet>OBIMessages</MessageSet>
  <BusinessInterface>
<!--************************************************************************-->
<!-- Specification of Service Interface 01 - Buying organization          -->
<!--************************************************************************-->
  <ServiceInterface InterfaceId="interface01">
    <OrgName Partyname="obibuyer"/>
    <ActionMenu>
      <Action ActionId="action01">

...

        <ResponseServiceTime>
          <ServiceTime>3600</ServiceTime>

...

        </ResponseServiceTime>
       </Response>
...

      </Action>
    </ActionMenu>
    <ServerServiceTime>
       <ServiceTime>3660</ServiceTime>

...

    </ServerServiceTime>

...

  </ServiceInterface>
<!--************************************************************************-->
<!-- Specification of Service Interface 02 - Selling organization         -->
<!-- This interface below is for initiating shopping and for solicited or -->
<!-- UnSolicited OBIPO from                                               -->
<!-- buying organization to selling organization                          -->
<!--************************************************************************-->
  <ServiceInterface InterfaceId="interface02">
    <OrgName Partyname="obiseller"/>
    <ActionMenu>
      <Action ActionId="action02">
...

      </Action>
    </ActionMenu>
```

```
    <ServerServiceTime>
      <ServiceTime>3660</ServiceTime>

...

    </ServerServiceTime>

...

  </ServiceInterface>
  </BusinessInterface>
  </BusinessProtocol>
</TPA>
```

Appendix 6. Mapping of TPA Constructs to Message Header

This appendix will describe how specific constructs defined in the TPA are mapped to the proposed ebXML message header standard. See ref. 1 in 1.1, "Normative References".