Executable Trading-Partner Agreements in Electronic Commerce[1]

Martin Sachs[2], Asit Dan, Thao Nguyen, Robert Kearney, Hidayatullah Shaikh, Daniel Dias

IBM T. J. Watson Research Center
Yorktown Hts, NY 10598

Abstract

In business to business electronic commerce, the terms and conditions describing the electronic interaction between businesses can be expressed as an electronic contract or trading-partner agreement (TPA) from which configuration information and code which embodies the terms and conditions can be generated automatically. This paper first discusses issues related to contracts and more generally to inter-business electronic interactions. Next, we describe the basic principles of electronic TPAs. The TPA expresses the rules of interaction between the parties to the TPA while maintaining complete independence of the internal processes at each party from the other parties. It represents a long-running conversation that comprises a single unit of business. Next, we describe our TPA language. We then describe tools for authoring TPAs and generating code from them. Finally, we describe an example of an application which can benefit from TPAs.

1. Introduction

Contracts describe legally enforceable terms and conditions in all kinds of interactions between people and organizations. Examples of interactions are marriage, employment, real estate purchases, and industrial supply arrangements. In business to business electronic commerce, there is a need to agree not only on the traditional terms and conditions but also on IT procedures from communication protocols to business protocols (Dan & Parr 1997a). Today, such contracts, or trading-partner agreements (TPAs), are generally written in human languages and then turned into code by programmers.

Business to business electronic commerce will be given considerable impetus by expressing the IT terms and conditions as electronic TPAs from which the code to perform the terms and conditions can be automatically generated at each party's business to business server. This will speed up the reduction of the terms and conditions to code and ensure that the code at each business partner's site will accurately embody the desired terms and conditions. In the longer term, electronic TPAs will also facilitate electronic negotiation of terms and conditions, at least for the simpler situations which need not involve extensive legal negotiation. Electronic negotiation in turn opens the possibility for spontaneous electronic commerce, i.e. quick and easy setup of business to business deals on the Internet (Dan *et al* 1998).

---

[2] Contact: mwsachs@us.ibm.com

In recent years, there has been a large amount of activity in modeling and analyzing various electronic commerce methods using contract or agreement approaches. Dan & Parr 1997b and Weigand & Ngu 1998 discuss how interoperable transactions in electronic commerce differ from traditional ACID (atomicity, consistency, isolation, durability) transactions (Gray & Reuter 1993) and the importance of distinguishing between the contract (communication behavior) and the task (the meaningful unit of work) and propose a scheme for specifying the contract which is suitable for analyzing the process.

Many academic publications discuss conceptual contracts as part of their models but the do not suggest a specific business to business contract language or discuss embodiment of a system based on such a contract. Dan & Parr 1997a discuss the general principles in business to business electronic commerce and mention the use of a business to business electronic contract but provide no details. Dan *et al* 1998 discuss the specific functions needed in a business to business electronic contract and describe the architecture of the prototype of a business to business server built at IBM Research but do not describe a specific contract language.
In this paper, we focus on the language for an electronic TPA and the tools to assist in composing the TPA and to generate code from it.

The paper is organized as follows. In section 2, we detail the issues that need to be addressed in business to business interactions. Section 3 discusses the principles of business to business electronic TPAs. In section 4, we describe our TPA language. In section 5, we describe the tools for creating TPAs and generating code from them. Finally, in section 6, we describe an application example which illustrates the use of the TPA.

2. Issues in Inter-Business Electronic Interactions

Increased automation of business processes within a business organization leads naturally to automation of business to business (B2B) interactions (Dan & Parr 1999). The issues of privacy, autonomy, heterogeneity in software and platforms, and more importantly, managing complexity of interactions, however, make this a challenging task. Some of these issues, e.g., heterogeneity of programming languages and platforms in which the application components are developed, and stateful interactions across program components, are also addressed in the automation of business internal processes and integrating application components. Total knowledge and control in the design of the business process within an organization make this a manageable task.

Component architectures such as CORBA (Corba 1998) and Enterprise Java Beans (Ejb 1999) provide middleware for integrating application components written in different languages. For the purpose of interaction, an application component needs to know only the interfaces to other components written in a suitable middleware integration language (e.g., Interface Definition Language or IDL in CORBA). In such environments, typically, the applications are executed as short ACID transactions. The underlying middleware provides necessary runtime services, e.g., naming, transaction, resource allocation. A long-duration application is modeled as a sequence of short independent steps invoked either manually or in an automated manner (Dan & Parr 1999,Wfmc 1998, Garcia-Molina & Salem 1987).

Most methodologies reported in the academic literature for automation of internal processes of individual businesses are not directly applicable for the automation of B2B interactions. First and foremost, no common shared underlying middleware can be assumed for distributed applications spanning organizational boundaries. Setting up such a common software bus requires tight coupling of the business partners' software platforms (e.g., consider the issues on security, naming, component registration).

Even if such a software bus can be established, ACID and/or complex extended transaction models of stateful interactions are not appropriate for such B2B interactions. First, implementation of such protocols necessitates tight coupling of operational states across business applications, which is highly undesirable. The application components in one organization may hold locks and resources in other organizations for an extended period of time, resulting in loss of autonomy. Rollback and/or compensation of application steps is no longer under the control of a single organization. Finally, in real-world business operations the states always move forward, and explicit recourse actions are taken by business partners to move to a more desirable operational state. An example is cancellation of a prior purchase or reservation.

In Dan and Parr 1997b, a conversational model of interactions is proposed where, based on the conversation history, each partner explicitly specifies the permissible operations. For management purposes, the internal business process is separated from external interactions. Each trading partner manages and is responsible for its own internal activities in the B2B application and may use ACID transactions within its own domain. The model furthermore structures the external interactions as actions consisting of requests, responses, modifications or cancellations, groups of actions that together satisfy certain interaction rules, and conversations demarcating interaction contexts. Interactions in one conversation may trigger actions in other conversations via execution of internal business logic.

The invocation of application components across organizational boundaries needs to be controlled and monitored (Dan and Parr 1997a, Dan *et al* 1998). First, without rigorous testing and cooperation in software development across organizations, the correct execution of such complex distributed applications can not be assumed. Second, in such automated interactions, trust becomes an overarching concern. During runtime, explicit checks are necessary to ensure that business partners are not violating any policy constraints (e.g., cancellation of a reservation must be within the allowable time window) .

In the Coyote (Cover Yourself Transaction Environment) project (Dan *et al* 1998), we address all of the above issues by setting up a B2B interaction via a composable interaction stack based on an electronic TPA. The automated process of setting up this interaction from an unambiguous formal specification and enforcing contractual agreements is termed an ***executable TPA***. The Coyote server provides additional services for supporting long running applications, e.g., application development, asynchronous event driven execution, compensation framework, maintaining correlation of conversations, logging and querying the activity on a conversation. However, these are not the focus of the current paper.

3. Principles of Business to Business Electronic TPAs

The purpose of the electronic TPA is to express the IT terms and conditions to which the parties to the TPA must agree in a form in which configuration information and the interaction rules which must be executable can be automatically generated from the TPA in each party's system. It should be understood that the information in the TPA is not a complete description of the application but only a description of the interactions between the parties. The application must be designed and programmed in the usual manner. As a simple example, the TPA may define requests such as "reserve hotel". The "reserve hotel" function must be designed, coded, and installed on the hotel server. That function may, in turn, invoke various site-specific functions and back-end processes whose details are completely invisible to the other party to the TPA.

We emphasize that the TPA is formulated to ensure that each party maintains complete independence from the other party both as to the details of the implementations and as to the nature of the business processes and back-end functions (database, transaction monitors, ERP functions, etc.) used. For example, as previously mentioned, the TPA neither requires, nor provides the means for, ACID transactions involving both parties.

In this paper, we use the terms "client" and "server" in the usual way. A client requests services of a server. However we envision applications in which a given party may play both server and client roles at different times. In other words, a party may both request services of the other party and receive service requests from the other party. In the simplest applications, there may be two parties, one of which is a always a server and the other, always a client. An example is a travel application involving a travel agency (client) and airline company (server). Even in such a simple case, however, the parties may exchange roles. For example, the airline company may issue requests to the travel agency for more information about the traveler or itinerary.

The TPA is represented at each party which acts as a server by an object, called a TPA object or (or equivalent code for non-object-oriented implementations), which performs rule checking and translation of the request messages from the form defined in the TPA to the actual method calls at the parties which act as servers. A similar TPA object, generated at each party which can act as a client to other party, performs the inverse translation, from local method calls to the request messages, as defined in the TPA, which are sent to the other party. A party which can act as both a client and as a server has both kinds of TPA object. Use of the TPA objects is illustrated in the examples in section 6.

The TPA represents a single long-running conversation, which is a set of related interactions, dispersed in time, that comprises a single unit of business. For example, in a travel application, the TPA might define the interactions between the travel agent and a hotel company starting with making the different reservations needed by the traveler, to the check-in processes during the trip, and ending when the traveler checks out at the last stop. This sequence of steps is a single long-running conversation. A unit of business is performed under the TPA by instantiating the TPA as a long-running conversation. To perform many units of business, the TPA may be instantiated as many long-running conversations (serially or concurrently) as is appropriate to the application and the processing capabilities of the parties' systems.

Figure 1 shows the main functions provided by the TPA. We now give a brief

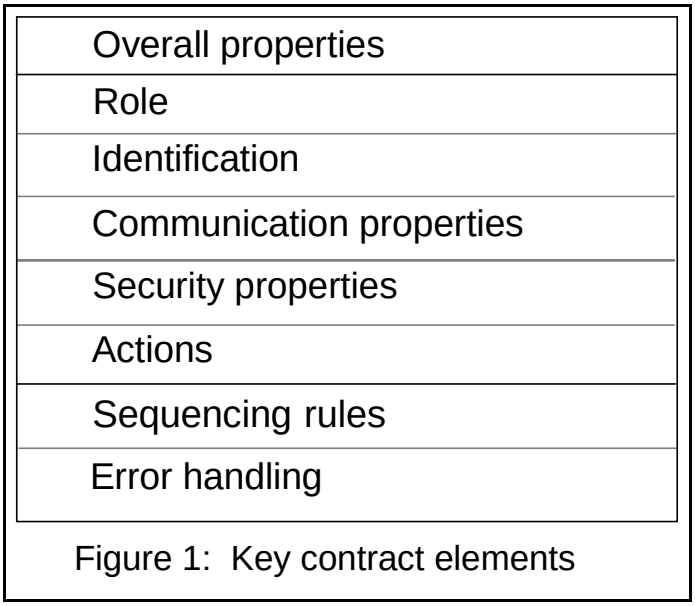

Figure 1: Key contract elements

overview of these functions. Section 4 describes the actual TPA language.

Overall properties of the TPA include its name, starting and ending dates, and similar global parameters. The role section provides the means to define a TPA in terms of generic roles such as airline and hotel and to produce a specific instance of the TPA by substituting specific parties for the role parameters. The identification section specifies the organization names of the parties and various contact information such as e-mail and postal service addresses. It also optionally specifies an outside arbitrator to be used for settling disputes. Communication and security properties include communication protocol (e.g. HTTP, SMTP), communication addresses, authentication and nonrepudiation protocols, certificate parameters, etc.

For each party which can act as a server, there is an action menu which lists the actions that the other party can request and various characteristics of those actions. Sequencing rules specify the order in which actions can be requested on each server. Error handling rules are various conditions related to error conditions, such as the maximum waiting time for the response to a request.

4. Business to Business TPA Language

The TPA is an XML document from which code is generated at each of the trading partners' computer systems. Authoring and code-generation tools are provided, as will be described later. The TPA document is described by an XML Document Type Definition (DTD) or XML-Schema file, which defines the tree structure of the TPA tags and some XML syntactic rules but not rules defining specific values of the tags or the semantic interrelations among the tags. These semantics are defined by a textual design document and are embodied in rules, understood by the authoring tool, which aid in the creation of a valid TPA.

4.1 Overall Structure

The overall XML structure of the TPA is as follows. Each of these tags is the top level of a subtree of tags (subelements). We will illustrate the following discussion with snippets of XML.

```
<TPA>
     <TPAInfo>  <!-- TPA preamble -->
          ...  <!--TPAname, role definitions,
               participants, etc.-->
          </TPAInfo>
     <Transport>
          ... <!--communication and transport
                   security information-->
          </Transport>
     <DocExchange>
          ... <!--document-exchange and message security
              information-->
          </Security>
     <BusinessProtocol>
          <ServiceInterface>  <!-- for each provider-->
               ...  <!--Action definitions etc.-->
               </ServiceInterface>
          </DocExchange>
     </TPA>
```

4.2 Layer Structure of 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.

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.

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.

Transport layer: The transport layer is responsible for message delivery using the selected communication protocol. Transport security (encryption and authentication) definitions are also provided.

4.3 Roles

When a given TPA can be repeatedly reused for different groups of parties, a prototype TPA or template can be written in terms of role parameters rather than specific party names. The authoring tool can then generate a specific TPA by substituting party names for the role parameters and filling in specifics of those parties such as their electronic addresses. The role definitions are included under the <TPAInfo> tag. Each <RoleDefn> tag supplies a pair of role parameter and actual name The <RoleName> tag defines the name of each role. The <RolePlayer> tag has a blank value in a TPA template and the name of an actual party in a specific TPA. Here is the XML for the role definitions for a TPA between an arbitrary airline (@airline) and an arbitrary hotel (@hotel). In this example, the tags under <Role> particularize the TPA to an agreement specifically between Hotelco and Airlineco.

```
<Role>
     <RoleDefn>     <!--one or more-->
          <RoleName>@hotel</RoleName>
          <RolePlayer>Hotelco</RolePlayer
          </RoleDefn>
     <RoleDefn>
          <RoleName>@airline</RoleName>
          <RolePlayer>Airlineco</RolePlayer>
          </RoleDefn>
     </Role>
```

When the authoring tool replaces the role parameters by actual party names, it either asks the author for party-specific information or finds this information in a previously-built database.

4.4 Transport Layer

In the transport layer, the communication properties section (<Communication> tag) defines the details of the system to system communication used in the application. These include the protocol to be used by both parties (e.g. HTTP, SMTP), each party's address parameters, maximum allowed network delay, and other parameters. Following is an example of the communication definition for HTTP:

```
<Communication>
   <HTTP>
     <Version>version</Version>
     <HTTPNode> <!--One for each party-->
          <OrgName Partyname=name/>
          <HTTPAddress>
               <URL URLName=type>url</URL>
               <!--additional URL tags as needed>
               </HTTPAddress>
          </HTTPNode>
     <NetworkDelay>time</NetworkDelay>  <!--Optional-->
     </HTTP>
   </Communication>
```

The transport-security properties tags (not shown) define the security protocols to be used in transporting messages. Protocols are defined for encryption and authentication. Encryption information includes the name of the encryption protocol and various parameters defining the certificates. Information supplied for authentication includes the type of authentication (e.g. password or certificate), the specific protocol (e.g. SSL), and the certificate parameters.

4.5 Document-Exchange Layer

Information included in the document-exchange layer includes the name of the protocol, such as OBI, the message-encoding choice (example: BASE64), whether or not duplicate messages should be detected, and the message-security definition. Message security may be either or both of digital-envelope (secret-key encryption using certificate-based encryption to exchange the secret keys) and certificate-based nonrepudiation.

4.6 Business-Protocol Layer

The `<BusinessProtocol>` tag defines the section of the TPA which 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. Each service interface contains some overall parameters and the action menu, which contains the set of definitions of the actions that this party will accept as service requests. The syntax is

```
<BusinessProtocol>
     <ServiceInterface>  <!--one or more-->
          ... <!-- action menu and other definitions-->
          </ServiceInterface>
     </BusinessProtocol>
```

4.7 Action Definition

For each party to the TPA which can act as a server, there is an action menu which identifies the permissible action requests and their characteristics. We discuss the main elements of an action definition using the following OBI buyer action definition (See "Application Example").

```
<Action>
   <Request>
     <RequestName>processOBIPOR</RequestName>
     <RequestMessage>OBIPOR</RequestMessage>
       <!--OBIPOR is a keyword which specifies the format of
       the message, in this case a purchase order request-->
    </Request>
   <Response>
       <ResponseName>handleOBIPO</ResponseName>
```

```
        <ResponseMessage>OBIPO</ResponseMessage>
        <ResponseServiceTime>
             <ServiceTime>3600</ServiceTime>
                  <!-- 1-hour maximum time -->
             </ResponseServiceTime>
        </Response>
    </Action>
```

The request name is `processOBIPOR`, i.e. the action transmits a purchase-order request to the OBI buyer. The `<Response>` tag indicates that the response is by means of an asynchronous message from the OBI seller server to the OBI buyer server and that the response causes the handleOBIPO method to be invoked at the issuer of the action (here, the OBI seller server). The response transmits a completed purchase order (`OBIPO`). The `<ResponseServiceTime>` tag specifies the worst case service time for the server (in this case, the OBI seller server) until the response is returned. Here, it is 3600 seconds, i.e. 1 hour. If the specified time is exceeded, it is up to the requester's application logic to decide what to do next.

Sequencing rules are used to specify the permissible order of action invocations on a given server. The permissible initial action or actions is specified as follows, specified under the `<ServiceInterface>` tag.

```
<StartEnabled>
     <RequestName>action_name</RequestName>
          <!--one for each action permitted as the initial
              action-->
     </StartEnabled>
```

There is one `<StartEnabled>` tag for each party which can act as a server. Only one of the actions whose names are specified under `<StartEnabled>` may be invoked as the first action in a given conversation on that server.

Within each action definition, a sequencing rule specifies which actions can no longer be invoked following the completion of the particular action, and which actions become permissible following the particular action. The specification is as follows:

```
<Sequencing>
     <Enable>   <!--actions permitted after this one-->
          <RequestName>name_of_action</RequestName>
               ...
          </Enable>
     <Disable>  <!--actions not permitted after this one-->
          <RequestName>name_of_action</RequestName>
               ...
          </Disable>
          </Sequencing>
```

The `<Enable>` tag specifies which actions are permissible following the action whose definition contains the `<Sequencing>` tag. The `<Disable>` tag specifies which actions are

no longer permitted after this action. We are investigating the possible need to extend the sequencing rules to cover sequencing of actions across multiple servers.

Many error conditions are handled in standard ways by the framework and their handling is not specified in the TPA. For example, the framework automatically retries for failures to receive transport-level acknowledgments. Some errors, such as sequencing errors, may be severe enough for the parties to invoke the arbitrator to determine whether a TPA violation occurred. Duplicate messages are most likely to arise during recovery, when incomplete actions are retried. The TPA can specify that if the recipient recognizes a duplicate message, the duplicate can be ignored. If the duplicate is a request message, the server can then re-send the response message.

5. TPA Authoring and Code Generation

In order to utilize an electronic TPA, the TPA must first be composed and agreed to by the parties. Then registration information must be extracted from the TPA and the necessary executable code generated. There are many possible designs for the tools. The design choices for the code generator and registration tool, in particular, depend on the specifics of the system in which they work. There can be no requirement that the same code generator and registration tool be used by both parties to the TPA. We here describe the tools we are developing as part of the Coyote project (Dan *et al* 1998). In our project, these tools are implemented in Java.

Because the TPA is a complex document and XML is not an intuitive language, an authoring tool is essential in preparing a TPA. Once the TPA is verified as valid and agreed to by both parties, it is passed to the TPA registration tool at each party's site. This tool extracts some of the content and stores the content in the registration database.

The business logic registration tool is used to associate actions which were specified in the TPA with business functions of which is a service provider, so that when the an action is requested of the service provider, the correct sequence of business functions is called.

The code generation tool uses information from the TPA and the registration database to convert a collection of templates into the executable file.

5.1 Authoring Tool

There are two parts to creating a TPA. They are creating models of the tags and authoring a specific TPA, guided by the models. The authoring tool provides a way for an expert to prepare a model from which a TPA can be constructed by someone with far less knowledge of the required semantics. The model contains the TPA semantic information needed to guide a user in creating a correct TPA.

The authoring tool starts with a DTD or XML Schema document, which provides the syntactic structure of the TPA. Then it constructs a model of a general TPA by asking the model maker to provide examples (semantics) of all parts of the TPA. Once a model is complete, it is available

to any author who, by answering a few specific questions, can create a very complex TPA with a high probability of success. Figure 2 illustrates the process of creating a model and a TPA.

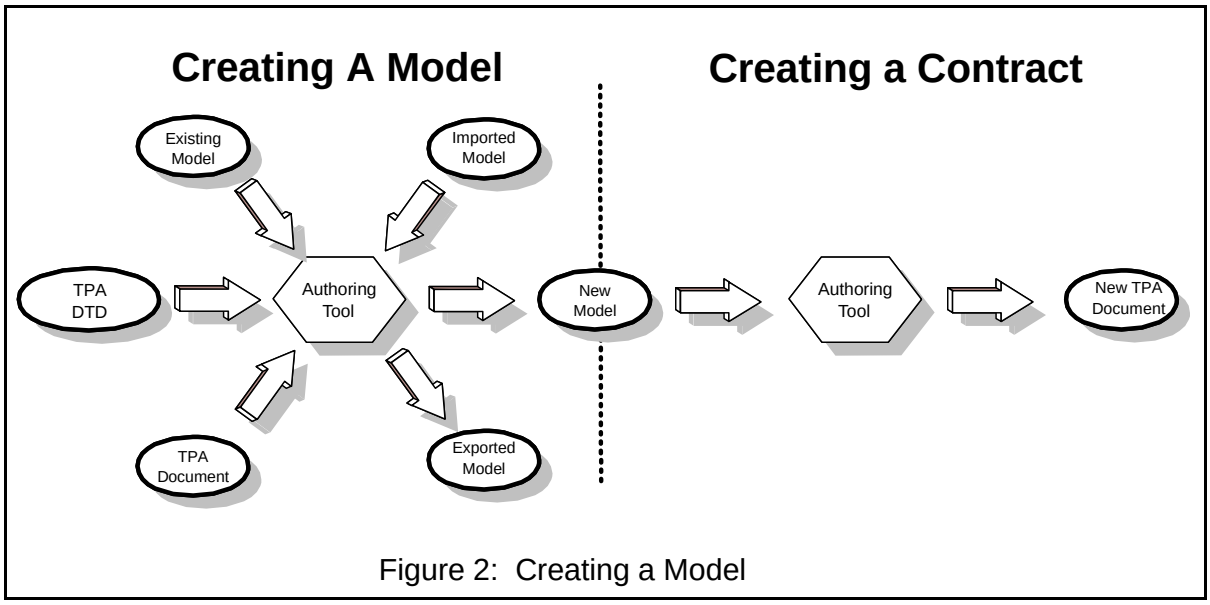

Figure 2: Creating a Model

A model consists of a collection of models of the tags to be used in the TPA. The models are in a tree structure which corresponds to the tree structure of the tags in the TPA. Each model of a tag is an example of the subtree under the tag. For example, a tag representing a communications protocol section has, as its subtree, information specific to a particular protocol.

The TPA author starts the authoring procedure after a model has been loaded. The authoring tool now uses the model to drive the authoring procedure. Starting with the root of the model, the authoring tool examines the choices for models beneath the root. If there is no choice to be made, the authoring tool accepts the model, proceeds to the next level, and repeats the above procedure for each child. If choices are to be made, a panel is displayed asking the user to select the correct model. The authoring tool then continues with that choice.

5.2 Code Generation

The code generator transforms the TPA into registration information and code which enforces the rules of interaction. A TPA object is created at the site of each party to the TPA. The code generation process is illustrated in Figure 3.

Code generation starts from a set of templates which consist of a combination of native (Java or any other) language and macro-style directives. These directives are written in a macro language consisting of information such as a basic set of data types, a basic set of functions used to obtain information from the TPA and other external sources, declaration statements, assignment

statements, and conditional statements which change the execution flow, depending upon values of variables and functions.

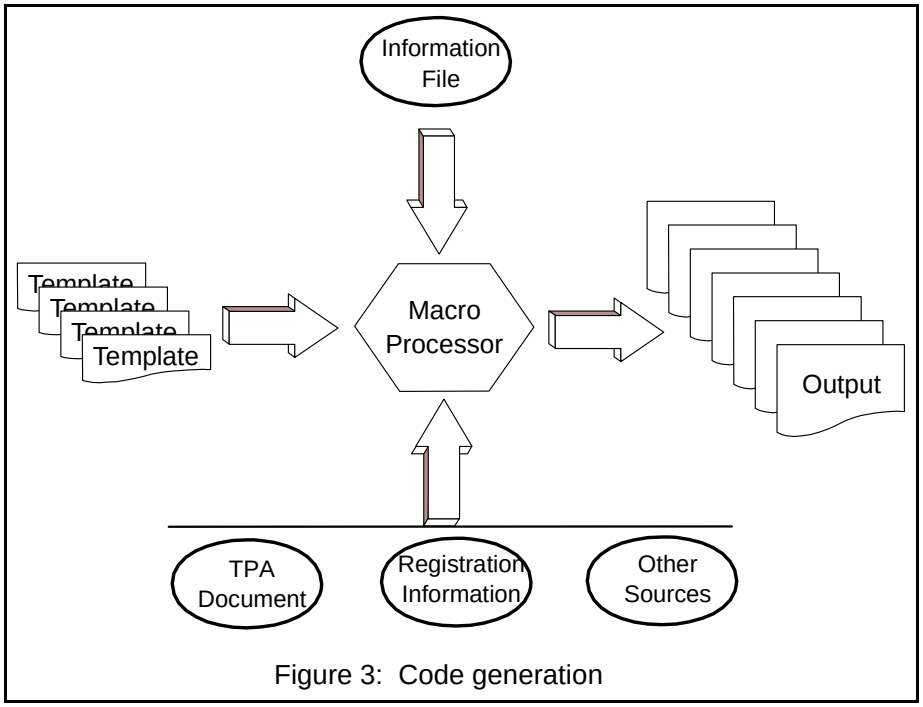

Figure 3: Code generation

A macro processor scans the template looking for directives. It executes any directives it encounters, and handles any native language statements as character strings, performing any needed processing, and writing the processed statements to a file.

6. Application Example

This section describes an example of the TPA and server structure. for an existing public protocol, OBI.

Open Buying on the Internet (OBI), Openbuying 1998, is a protocol for business-to-business Internet commerce. It was designed by the Internet Purchasing Roundtable and is supported by the OBI Consortium. OBI defines the procedures for the high-volume, low-dollar purchasing transactions that make up most of an organization's purchasing activity. In this section, we describe OBI, how it can be described by a TPA, and a schematic view of a possible implementation. Figure 4 illustrates the participants in an OBI transaction and the basic information flows. A complete OBI TPA is shown in the appendix.

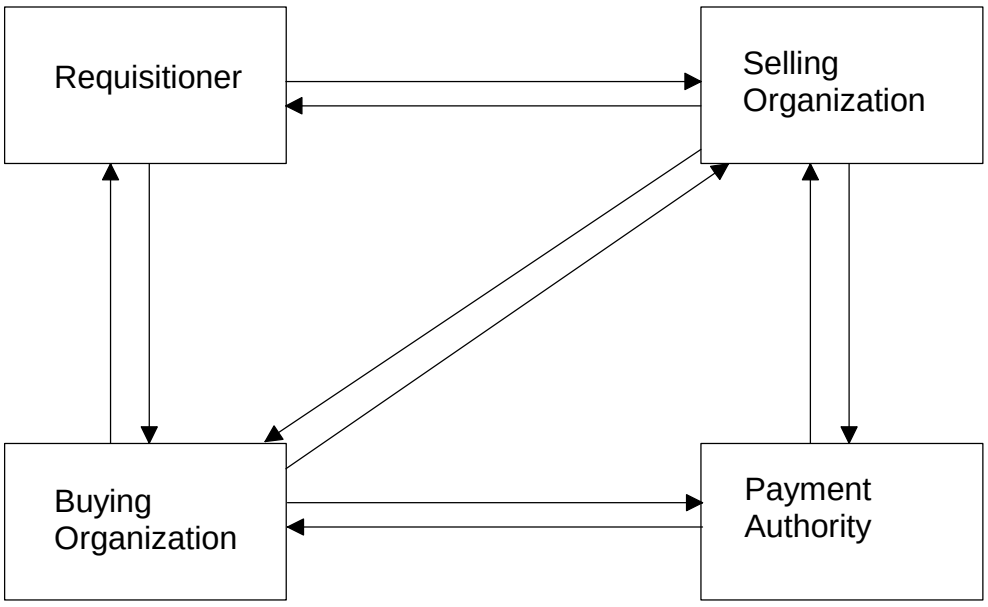

Figure 4: OBI Participants and Flows

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. 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 which 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. This mode might be used, for example, when a purchasing department purchases large volumes to supply a stock room.

In a one possible implementation of OBI, there is a TPA between the buying organization and the selling organization, each of which has a business to business server. In OBI terms, the TPA is a trading partner agreement (TPA). The payment authority, if present, is outside the scope of the

2-party TPA between buying organization and 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.

Following are the main functions included in the OBI TPA:

- Organization names of the parties to the TPA.
- Communication protocol definition. In this case it is HTTP, and includes the specific URLs of the buyer and seller.
- Security information such as the protocol (SSL in this case) and various certificate parameters
- Action menus for the buyer and the seller. The action list for the buyer is illustrated above in "Business to Business TPA Language". It consists of one action, "Process OBI Purchase Order Request". The completed purchase order is returned to the seller by means of a callback. The action list for the seller also consists of one action, "Process OBI Unsolicited Purchase Order".

Figure 5 shows the basic system structure and flow of an implementation of OBI. Shown in the figure are the TPA objects generated from the TPA at the buyer and seller servers. These objects provide the interfaces between various processes controlled by the TPA (in particular, the action requests) and the application logic at each server.

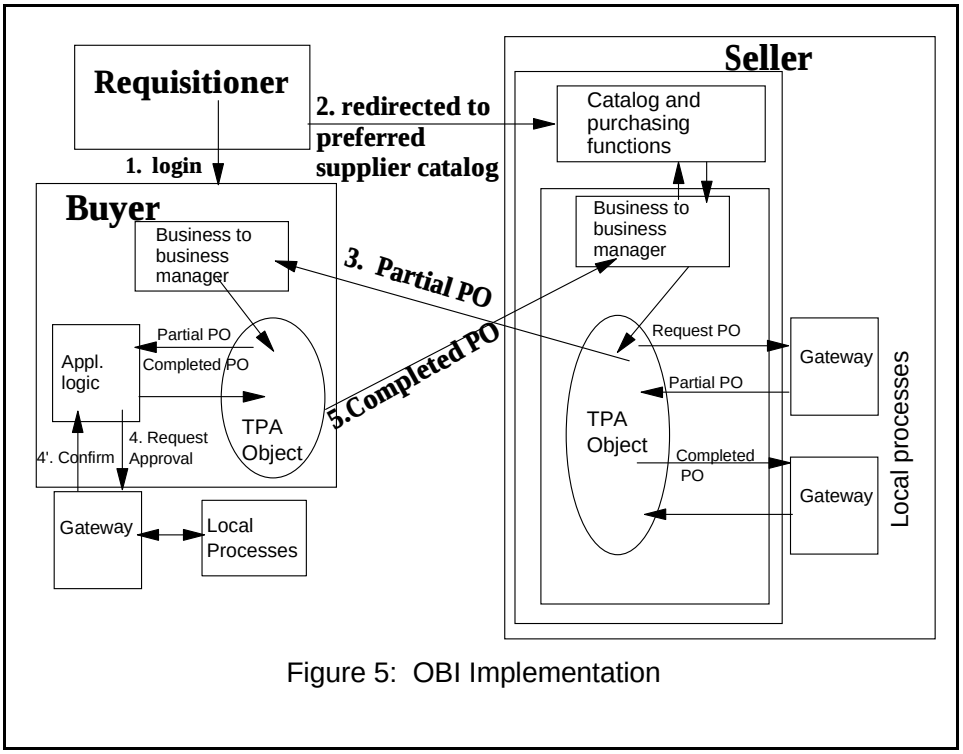

Figure 5: OBI Implementation

The process starts when a requisitioner contacts(1) the buyer server via a browser and is redirected(2) to the URL for the seller server. The requisitioner is shown the supplier catalog

appropriate to the requisitioner's organization. When the requisitioner makes a selection, the request is communicated to the TPA object. The TPA object communicates the purchase request to the local business processes via one of the gateways shown at the far right in the figure. A partial purchase order is returned to the TPA object via the gateway. The TPA object then issues the `processOBIPOR` action request(3) to the buyer server, sending a partial purchase order to the buyer server.

This request arrives at the buyer's TPA object, which evaluates the rules defined in the TPA and then sends the partial purchase order to the buyer application logic. In processing the partial purchase order, the application logic communicates with local business processes, via the gateway shown at the lower left in the figure, to request approval(4) of the purchase order. If the purchase is approved(4'), the approval arrives at the application logic, which completes the purchase order and passes the completed purchase order to the buyer's TPA object. The TPA object then issues the callback(5), sending the completed purchase order back to the seller.

The completed purchase order arrives at the seller's TPA object, which passes it to the local processes via the gateway at the lower right. The local processes handle fulfillment (e.g. shipping) and invoicing/payment. They also initiate a confirmation message to be returned to the requisitioner via the browser (not shown in the figure).

7. Future Work

We are extending the TPA ideas and language to areas such as TPA hierarchy, linking of multiple TPAs, and dynamic negotiation. We are also investigating TPAs in which there are more than two parties.

In addition, we are investigating how to incorporate business constraints into the TPA. Business constraints are conditions placed on data items in response messages. The results of these tests may modify further processing within the TPA. An example is a test of whether a cancellation action (e.g. to cancel a reservation) was issued during the allowed time range after the original action.

8. Summary

This paper has discussed various issues in inter-business electronic interactions and in the use of an electronic TPA for embodying the IT-related and business protocol terms and conditions used in business to business electronic commerce. We have designed an XML-based TPA language and tools for authoring TPAs in that language and generating code from the TPAs. We described examples of two applications which make use of TPAs and showed schematic views of such systems.

Acknowledgments

The authors express their appreciation to the following for contributions to the formulation of the TPA principles and language: Francis Parr, Vibby Gottemukkala, Terry Lau, Satwinder Brar, George Kleon, Gerald Anderson, John Ibbotson, Christine Draper, Linh Lam, Stewart Palmer, Richard King, and Sastry Duri.

References

Corba: *The Common Object Request Broker Architecture and Specification,* Rev. 2.2, Object Management Group, http://www.omg.org, 1998.

Dan, A., Dias, D., Nguyen, T., Sachs, M., Shaikh, H., King, R., and Duri, S., The coyote project: framework for multi-party e-commerce, *Proc. Research and Advanced Technology for Digital Libraries, Second European Conference,* ECDL'98, Heraklion, Greece, Sept. 1998, Springer Verlag, Berlin, Germany, 1998, p. 873-889.

Dan, A. and Parr, F. An object implementation of network centric business service applications (NCBAs), *OOPSLA Business Object Workshop,* Atlanta, GA, USA, Sept. 1997a.

Dan, A. and Parr, F., The coyote approach for network centric business service applications, *HPTS Workshop,* Asilomar, CA, USA, 1997b.

Dan, A. and Parr, F. Long running application models and cooperating monitors, submitted to *HPTS workshop,* Asilomar, CA, 1999.

Ejb: *Enterprise Java Beans Specification,* ver. 1.1, http://www.javasoft.com/products/ejb, 1999.

Garcia-Molina, H. and Salem, K., SAGAS, *Proc. of ACM SIGMOD Conf.,* Association for Computing Machinery, New York, NY,1987, pp. 249-259.

Gray, J. and Reuter, A., *Transaction Processing: Concepts and Techniques,* Morgan-Kaufmann, San Mateo, CA, 1993.

Openbuying: *Open Buying on the Internet Technical Specifications,* Release V1.1, The Open Buying on the Internet (OBI) Consortium, http://www.openbuy.org, 1998.

Weigand, H. and Ngu, A., Flexible specification of interoperable transactions, *Data & Knowledge Engineering,* Vol. 25, 1998, pp. 327-345.

Wfmc: *The Workflow Management Coalition Specification,* http://www.wfmc.org, 1998.

Appendix: OBI TPA

Following is a TPA which defines OBI.

```
<?xml version="1.0"?>
<!DOCTYPE TPA SYSTEM "TPA.xsd" >
<!--************************************************************************-->
<!--              OBI TPA      between Large Co (buying company)           -->
<!--                     and Pens Are We (selling company)                  -->
<!-- (C) Copyright IBM Corporation 2000                                     -->
<!--************************************************************************-->
<TPA xmlns="tpa.xsd">
<!--************************************************************************-->
<!-- 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>
        <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>
<!--*********************************************************************-->
<!-- Specification of Arbitrator                                         -->
<!--*********************************************************************-->
      <Arbitrator IdCodeType="01" MemberId="888000009000001">
        <PartyName Partyname="_XYZArbitrator">XYZArbitrator</PartyName>
        <CompanyTelephone>780-333-1111</CompanyTelephone>
        <Address>
```

```
          <AddressType>location</AddressType>
          <AddressLine>XYZArbitrator</AddressLine>
          <AddressLine>Suite 3</AddressLine>
          <AddressLine>77 Lawyers Blvd</AddressLine>
          <City>ABC City</City>
          <State>MA</State>
          <Zip>01234</Zip>
          <Country>USA</Country>
        </Address>
        <Contact Type = "primary">
          <LastName>Black</LastName>
          <FirstName>Joe</FirstName>
          <MiddleName>K.</MiddleName>
          <Title>Mr.</Title>
          <ContactTelephone Type = "primary">780-333-4040
</ContactTelephone>
          <ContactTelephone Type = "secondary">780-333-4045
</ContactTelephone>
          <EMail Type = "primary">jblack@xyzarbitrator.com</EMail>
          <EMail Type = "secondary">
            http://www.xyzarbitrator.com/jblack.html</EMail>
          <Fax>780-333-5000</Fax>
        </Contact>
      </Arbitrator>
    </Participants>
    <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 Transport Protocol #01                                -->
<!--************************************************************************-->
  <Transport>
   <Communication>
    <HTTP>
     <HTTPNode>
      <OrgName Partyname="_LargeCo"/>
      <HTTPAddress>
        <URL URLName="requestURL">
            https://www.largeco.com/jackal/servlet/OBIBuy</URL>
      </HTTPAddress>
     </HTTPNode>
     <HTTPNode>
      <OrgName Partyname="_PensAreWe"/>
      <HTTPAddress>
        <URL URLName="logOnURL">
            https://www.pensarewe.com/coyote/servlet/OBILogon</URL>
        <URL URLName="requestURL">
            https://www.pensarewe.com/coyote/servlet/OBIsell</URL>
        <URL URLName="responseURL">
            https://www.pensarewe.com/coyote/servlet/OBIsell</URL>
      </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>
        <CertificateAuthen>
        <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>
       </CertificateAuthen>
      </Authentication>
     </TransportSecurity>
    </Transport>
<!--*************************************************************************-->
<!-- Specification of DocExchange Protocol                                   -->
<!--*************************************************************************-->
  <DocExchange>
    <DocExchangeProtocol>OBI</DocExchangeProtocol>
    <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>
<!--************************************************************************-->
<!-- Specification of Service Interface 01                                  -->
<!--************************************************************************-->
   <ServiceInterface InterfaceId="interface01">
    <OrgName Partyname="_LargeCo"/>
    <Client>
      <OrgName Partyname="_PensAreWe"/>
    </Client>
    <ActionMenu>
      <Action ActionId="action01" Type="basic">
        <Request>
          <RequestName>putOPOR</RequestName>
          <RequestMessage>OBIPOR</RequestMessage>
        </Request>
        <Response>
          <ResponseName>getOPO</ResponseName>
          <ResponseMessage>OBIPO</ResponseMessage>
          <ResponseServiceTime>
            <ServiceTime>3600</ServiceTime>
            <Presume>fail</Presume>
          </ResponseServiceTime>
        </Response>
      </Action>
    </ActionMenu>
    <ServerServiceTime>
        <ServiceTime>3660</ServiceTime>
        <Presume>fail</Presume>
        </ServerServiceTime>
    <StartEnabled>
      <RequestName>putOPOR</RequestName>
    </StartEnabled>
   </ServiceInterface>
<!--************************************************************************-->
<!-- Specification of Service Interface 02                                  -->
<!-- This interface below is for UnSolicited OBIPO from                     -->
<!-- buying organization to selling organization                            -->
<!--************************************************************************-->
  <ServiceInterface InterfaceId="interface02">
    <OrgName Partyname="_PensAreWe"/>
    <Client>
      <OrgName Partyname="_LargeCo"/>
    </Client>
    <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>
            <RequestMessage>OBIPO</RequestMessage>
          </Request>
        </Action>
      </ActionMenu>
      <ServerServiceTime>
          <ServiceTime>3660</ServiceTime>
          <Presume>fail</Presume>
      </ServerServiceTime>
    </ServiceInterface>
  </BusinessProtocol>
</TPA>
```