Business-to-business integration with tpaML and a business-to-business protocol framework

by A. Dan
D. M. Dias
R. Kearney
T. C. Lau
T. N. Nguyen
F. N. Parr
M. W. Sachs
H. H. Shaikh

In business-to-business interactions spanning electronic commerce, supply chain management, and other applications, the terms and conditions describing the electronic interactions between businesses can be expressed as an electronic contract or trading partner agreement (TPA). From the TPA, configuration information and code that embody the terms and conditions can be generated automatically at each trading partner's site. 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. This paper summarizes the needs of interbusiness electronic interactions. Then it describes the basic principles of electronic TPAs, followed by an overview of the proposed TPA language. The business-to-business protocol framework (BPF) provides various tools and run-time services for supporting TPA-based interaction and integration with business applications. Finally, we describe examples of solutions constructed using TPAs and BPF.

As we enter the new millennium, fundamental changes are happening to trade and the way it is organized. There is a growing shift toward an electronically connected world in which ideas, information, and services are replacing the traditional reliance on physical goods production as the primary generator of wealth and employment. In this new economy, market dynamics will dictate a business model that provides for the integration of different partners in a value chain. Using a variety of technologies from information technology (IT), this model can enable highly coordinated trading communities, each with the ability to operate like a "virtual enterprise."

The emerging e-business Web economy requires an agile enterprise that can work more directly with suppliers and customers and respond more rapidly and intelligently to change. Technologies such as the Internet are beginning to transform traditional business models. Business pressures—margin erosion, channel proliferation, rising customer expectations, time-based competition, and faster product commoditization—are placing increased emphasis on how organizations operate and interoperate with other enterprises.

To enable customers to adapt to these dramatic changes in the business environment, middleware will increasingly be required to provide dynamic and flexible integration between partners in the value chain. Although new technologies will be needed to enable such integration, they will have to work seamlessly with existing interenterprise business processes (e.g., EDI—Electronic Data Interchange) and leverage investments in existing enterprise application integration (EAI).

This paper describes new middleware technology for business-to-business integration developed and prototyped by IBM Research. The central innovation is trading-partner agreement Markup Language

0018-8670/01/$5.00 © 2001 IBM

(tpaML)—an Extensible Markup Language (XML)[1,2] grammar for expressing electronic trading-partner agreements and a candidate for standardization. IBM Research has also designed and prototyped the business-to-business protocol framework (BPF). This runtime framework enables business protocols expressed in tpaML to be automatically deployed. We describe BPF and its use in some example scenarios to make the case that tpaML-based integration is both feasible and effective.

In the next section of the paper, we detail the issues that need to be addressed in business-to-business interactions. Subsequently, we discuss the principles of business-to-business electronic trading-partner agreements (TPAs). Then we describe our TPA language. The fifth section describes the architecture and initial implementation of BPF. Afterward, we describe the tools for creating TPAs and generating code from them. Finally, we describe application examples that illustrate the use of the TPA and BPF.

Interbusiness electronic interactions

In this section, what is required for electronic interactions between businesses and the technical procedures to implement the interactions are described.

Business requirements. Facilities such as EDI have successfully provided electronic document interchange between companies and their suppliers for a number of years. However, the high cost of EDI and its dependency on specialized deployment skills have always proved a barrier to adoption by all but the largest enterprises. Although EDI will continue to evolve, utilizing pervasive networks such as the Internet to reduce costs, complementary technologies are emerging that are able to provide some of the key capabilities necessary to enable dynamic business process integration. The basis of these technologies is the formulation of:

- A "common language" that can be employed by existing or potential trading partners to specify how they will interact
- An "electronic contract" that employs this common language in order to define and enforce the interaction protocols with which they will do business

Business-to-business protocols such as Open Buying on the Internet** (OBI**)[3] and RosettaNet**[4] are beginning to set a standard for business interactions (albeit currently fragmented). Our research work has demonstrated that tpaML and BPF provide a comprehensive tool set for the specification, configuration, customization, and execution of electronic TPAs.

Business-to-business interchanges based on EDI have long been defined by informal textual documents called TPAs. These TPAs are contracts that define both the legal terms and conditions and the technical specifications that both partners must implement to put the electronic trading relationship into effect. They are given to each partner's programmers to implement the technical specifications and are therefore subject to misinterpretation, resulting in implementation errors that must be corrected before electronic exchanges can begin. In contrast, an electronic TPA can be used to automatically generate (using suitable tools) the necessary customization information in each partner's system, thus assuring that the systems are compatibly and correctly set up for electronic business.

IBM has submitted a draft of tpaML to the Electronic Business XML (ebXML) initiative[5] for standardization. Standardization of the TPA language is a key element of interoperability between the business-to-business servers of different vendors.

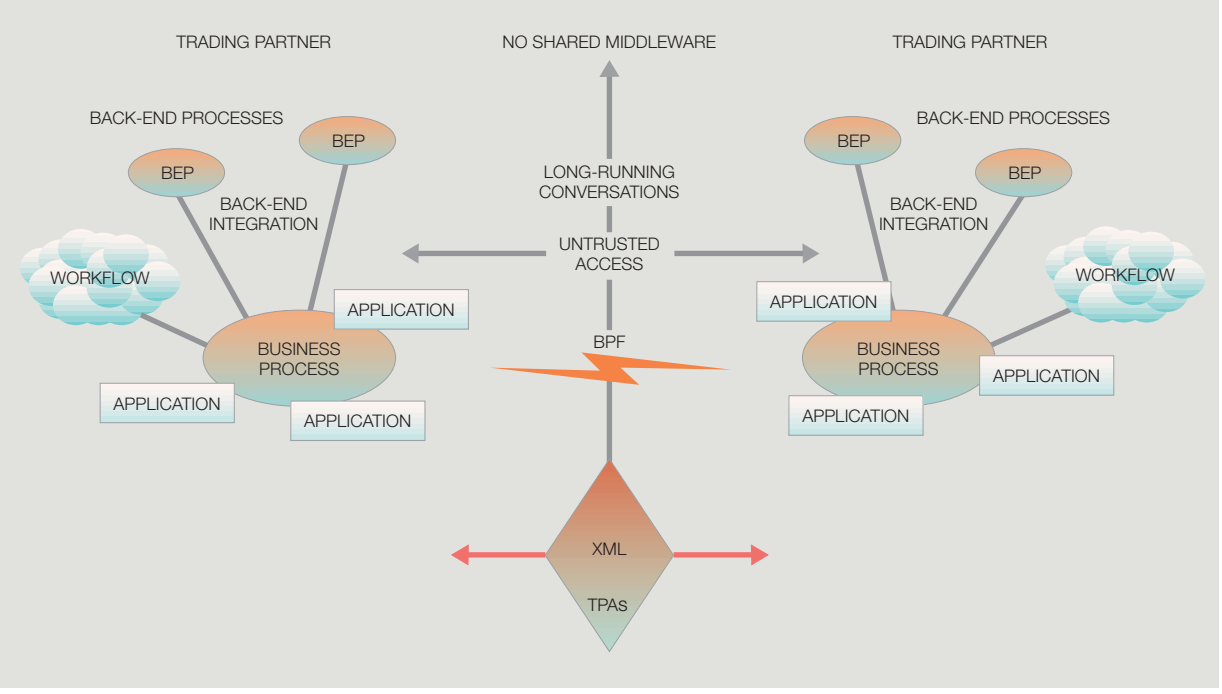
Technologies such as XML and TPAs, coupled with advances in middleware and workflow software, provide the key building blocks needed to underpin an electronic business-to-business integration infrastructure. Supporting the extensible and easy-to-use TPA format with the BPF framework and middleware from IBM's MQSeries* and WebSphere* families enables dynamic business process integration by providing:

- Integration of internal processes, using modifiable "business rules" to route information between the various internal business information systems
- Secure, reliable, and auditable e-document interchange between organizations
- Externalization of appropriate business functions and processes to suppliers, customers, and partners

In addition, this infrastructure can provide support for:

- The use of a broad range of standard message formats, transport and business protocols, and network connections with the capability to dynamically connect new trading partners

Figure 1 BPF—business-to-business protocol framework



- Easy-to-use, business-oriented "single point of control" for interactions across an extended or virtual enterprise
- Extensible open interfaces with flexible connectors to link to existing applications

The BPF addresses these requirements. Figure 1 illustrates the use of BPF for communication between two trading partners. The trading partners are on the left and right sides of the figure, connected by BPF, shown as a lightning bolt. Each partner has a business-to-business server that hosts the application and includes either BPF or equivalent compatible middleware. At each partner the business process can be seen, consisting of one or more applications and some back-end processes such as workflow. Listed in the center are three requirements for successful business-to-business middleware: no shared middleware, long-running conversations, and support for untrusted access. These requirements are discussed later.

Technical basis for interbusiness electronic interactions. 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 ranging from communication protocols to business protocols.[6] Today such contracts, the trading-partner agreements, or TPAs, are generally written in human languages and then turned into code by programmers.

Adoption of business-to-business electronic commerce will be considerably accelerated 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. Use of electronic TPAs 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 that need not involve extensive legal negotiation. Electronic negotiation in turn opens the pos-

sibility for spontaneous electronic commerce, i.e., quick and easy setup of business-to-business deals on the Internet.[7]

In recent years, there has been much activity in modeling and analyzing various electronic commerce methods using contract or agreement approaches. Dan and Parr[8] and Weigand and Ngu[9] discuss how interoperable transactions in electronic commerce differ from traditional ACID (atomic, consistent, isolated, durable) transactions[10] and the importance of distinguishing between the contract (communication behavior) and the task (the meaningful unit of work). They propose a scheme for specifying the contract that is suitable for analyzing the process.

Ajisaka[11] discusses software as an object of electronic commerce and proposes an architecture for managing custom software development by contract. Sandholm[12] describes algorithms for modeling electronic commerce transactions that do not require enforcement. Sandholm and Lesser[13] discuss issues in automated negotiation among agents whose rationality is bounded by computational complexity. Konana et al.[14] describe an approach to improve quality of service in multimedia information delivery based on conceptual contracts between end users and surrogate servers and among the surrogate servers.

Many of the publications cited above discuss conceptual contracts as part of their models, but they do not suggest a specific business-to-business contract language or discuss embodiment of a system based on such a contract. Dan and Parr[6] 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.[7] 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 discuss the language for an electronic TPA and the tools to assist in composing the TPA and generating code from it.

Increased automation of business processes within a business organization leads naturally to automation of business-to-business interactions.[15] 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.

The invocation of application components across organizational boundaries needs to be controlled and monitored.

Component architectures such as Common Object Request Broker Architecture** (CORBA**)[16] and Enterprise JavaBeans**[17] 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 run-time services, e.g., naming, transaction, and resource allocation. A long-duration application is modeled as a sequence of short independent steps invoked either manually or in an automated manner.[15,18,19]

Most methodologies reported in the academic literature propose a specific request protocol or "software bus" for automating the internal processes of individual businesses. These methodologies are not directly applicable to the automation of business-to-business 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, and component registration).

Second, even if such a software bus can be established, ACID or complex extended transaction models of stateful interactions, or both, are not appropriate for such business-to-business interactions. 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 or compensation of application steps, or both, is no longer under the control of a single organization. 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.

As discussed in Dan and Parr,[8] the middleware and TPA provide a conversational model of interactions wherein, based on the conversation history, each partner explicitly specifies the permissible operations. We refer to such a model as a long-running conversation. The long-running conversation is the model by BPF of a single unit of business. The long-running conversation is not an ACID transaction; it is simply a grouping of the related operations that comprise the unit of business. Each partner's system tracks the state of the conversation and maintains a log that can be used for purposes such as audit trail and recovery.

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 business-to-business 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. In this way, BPF can manage a supply-chain situation in which a business partner, during a long-running conversation, may call upon subcontractors.

The invocation of application components across organizational boundaries needs to be controlled and monitored.[6,7] First, without rigorous testing and cooperation in software development across organizations, the correct execution of such complex distributed applications cannot be assumed. Second, in such automated interactions, trust becomes an overarching concern. During run time, 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).

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 executable interaction rules 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 encoding the endpoint business logic 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, enterprise resource planning functions, etc.) used. For example, as previously mentioned, the TPA neither requires, nor provides the means for, ACID transactions involving both parties.

In describing the TPA language, 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 are two parties, one of which is always a server and the other always a client. An example is a travel application involving a travel agency (client) and an 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.

In our implementation, the TPA is represented in the run-time system at each party that acts as a server by an object, called a TPA object. The TPA object performs rule checking and translation of the request

messages from the form defined in the TPA to the actual method calls at the parties that act as servers. A similar TPA object, generated at each party, that can act as a client to the 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 that 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 given later in the paper.

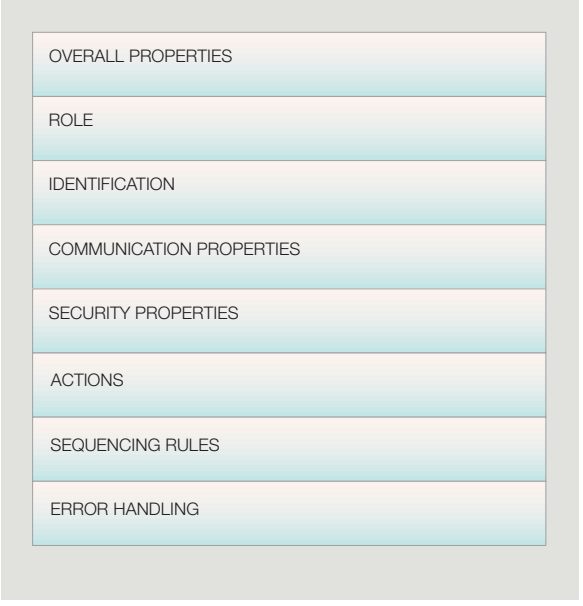
The TPA execution instance represents a single long-running conversation, which is a set of related interactions, dispersed in time, comprising a single unit of business. For example, in a travel application, the TPA might define the interactions between the travel agent and a hotel company starting from the point where the different reservations needed by the traveler are made, 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 2 shows the main information content and function provided by the TPA. We now give a brief overview of these functions. The next section describes the actual TPA language.

Overall properties of the TPA include its name, partner names, starting and ending dates, and similar global parameters. Definitions of roles are also provided. Communication and security properties include communication protocol (e.g., HyperText Transfer Protocol, or HTTP, and Simple Mail Transfer Protocol, or SMTP), communication addresses, authentication and nonrepudiation protocols, and certificate parameters.

For each party that can act as a server, there is an action menu listing the actions that the other party can request and defining 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.

Figure 2 Key elements of TPA



Today's informal trading-partner agreements often include terms and conditions related to the application protocol. For example, a procurement TPA might include price lists and requirements on delivery time. In contrast, at this time, tpaML is concerned only with the protocols for the message exchanges between the partners in performing the application protocol. Higher-level issues (i.e., matters relating to the content of the message payloads) are the responsibility of the application program or higher-level application framework, or both.

Business-to-business TPA language

The TPA is an XML document from which code is generated or customized at each of the trading partners' computer systems. Authoring and code generation or customization tools are provided, as will be described later. The TPA document is described by an XML document type definition (DTD) or XML-Schema document, which defines the tree structure of the TPA tags and XML syntactic rules. Some rules defining specific values of the tags or the semantic interrelations among the tags can be defined in the DTD, and more can be defined using XML Schema. However, some TPA semantics defined in the tpaML specification cannot be defined in the DTD or XML Schema; these semantics are understood by the authoring tool, which uses them to aid in the creation of a valid

TPA. In this section, the term "framework" is used to represent the generic run-time code (such as BPF) that supports the TPA and the interactions between business partners.

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 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-->
    </DocExchange>
    <BusinessProtocol>
         <ServiceInterface> <!-- for each
                                provider-->
              ... <!--Action definitions
                                   etc.-->
         </ServiceInterface>
    </Business Protocol>
</TPA>
```

Layer structure of TPA. The <BusinessProtocol>, <DocExchange>, and <Transport> sections above 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) that parties to the TPA can request of each other and sequencing rules that determine the order of requests. The business protocol layer of BPF 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 defines various general properties of the documents exchanged by the parties. The document exchange layer of BPF 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 reverse process takes place for received messages.

Transport layer. The transport layer defines the communication protocol. Transport security (encryption and authentication) definitions are also included. The transport layer of BPF is responsible for message delivery using the selected communication and security protocols.

TPA information. Overall properties of the TPA include its name, partner names, 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 contact information such as e-mail and postal service addresses. It also optionally specifies an outside arbitrator to be used for settling disputes.

When a given TPA can be repeatedly reused for different pairs 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 parameters and the 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. Following 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 entities named 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.

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 details include the protocol to be used by both parties (e.g., HTTP and 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>
               <LogOnURL>url</LogOnURL>
               <RequestURL>url</RequestURL>
               <ResponseURL>url</ResponseURL>
          </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., Secure Sockets Layer, or SSL), and the certificate parameters.

Document exchange layer. Information included in the document exchange layer includes 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 (symmetric encryption using certificate-based encryption to exchange the shared secret key) and certificate-based nonrepudiation. Any document formats agreed to by the two parties may be used. We discuss the action definition in a later subsection.

Delivery channels. A delivery channel consists of one transport definition and one document exchange definition. It corresponds to a single communication and document-processing path. The TPA may include more than one transport definition and more than one document exchange definition. These definitions can be grouped into delivery channels with different characteristics. Each action definition may specify a particular delivery channel, thus allowing the business partners to specify a different path for each message if necessary. The definition of delivery channels also permits the TPA to specify that the framework may dynamically choose a delivery channel for each message based on conditions such as path congestion and failures.

Business protocol layer. 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. 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 two-->
           ... <!-- action menu and other
                definitions-->
      </ServiceInterface>
</BusinessProtocol>
```

Action definition. For each party to the TPA, an action menu 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 section on application examples).

```
<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
                    from seller to buyer-->
   </Request>
   <Response>
       <ResponseName>handleOBIPO
       </ResponseName>
   </Response>
   <ResponseServiceTime>
       <ServiceTime>3600</ServiceTime>
            <!-- 1-hour maximum time -->
   </ResponseServiceTime>
</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 a message from the OBI seller server to the OBI buyer server and that the response causes the handleOBIPO action to be invoked at the issuer of the request (here, the OBI seller server). The response transmits a completed purchase order (OBIPO). The <ResponseName> tag identifies an action at the other party that will process the response message. The <ResponseServiceTime> tag specifies the worst case service time for the server (in this one case, the OBI seller server) until the response is returned. Here, it is 3600 seconds, i.e., one hour. If the specified time is exceeded, it is up to the requester's application logic to decide what to do next. Not shown here is the definition of the handleOBIPO action in the seller's service interface.

The value of the <RequestMessage> tag defines the format of the business document sent in the message. For XML documents, the value may be the uniform resource locator (URL) of the document type definition document or XML-Schema document that defines the document format. Alternatively, the value might be a keyword that could represent an entry in a local (at each business partner) dictionary that defines the agreed-to format. Both tpaML and BPF can, by means of plug-in parsers and document generators, support any document format, standardized or not, agreed upon by the partners. With the appropriate plug-ins, even traditional EDI messages can be supported. In future applications, it is expected that XML will be the language of choice for structuring documents. However, other formats, such as the current EDI formats, must also be supported today.

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 that 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.

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 contact an arbitrator to determine whether a TPA violation occurred. A tag in the <TPAInfo> section identifies the arbitrator. Duplicate messages are most likely to arise during recovery, when incomplete actions are retried. The TPA can specify that the duplicate can be ignored if the recipient recognizes a duplicate message. If the duplicate is a request message, the server can then resend the response message.

Business-to-business protocol framework

BPF is a general mechanism to support various business protocols and business-to-business processes, both custom-designed and existing protocols such as OBI. Dan et al.[7] describe a prototype of such a mechanism that was developed in a research project at IBM Research. In the BPF architecture, each business protocol is supported by creating a personality for it, based on the specification in a TPA. We describe the use of the TPA with OBI later in this paper. Many business-to-business processes, such as request for

quote (RFQ), request for information (RFI), and other processes, are similar to the OBI flow and can be implemented by the same framework. These other processes will be implemented largely by changing the TPA. Tie-in to back-end systems is provided by invoking an extensibility framework from the "business logic interface" that is triggered by incoming requests, such as partial or completed purchase orders in the case of OBI.

BPF provides the following functions, among others: TPA installation, routing of messages to the specified action at the destination, sequencing rules, business document generation and parsing, security, correlation of conversations, logging, and recovery.

It should be understood that if two partners have agreed on a TPA specifying how they will interact, they are still free to choose how to provide the implementation of the business protocol. A BPF can be used to generate the implementation. The partners could have different BPF frameworks supporting the TPA standard or could deploy any business protocol implementation consistent with the TPA specification. This independence is essential for doing business over an open medium such as the Internet. One should not dictate to others which technology to use to send and receive messages. BPF is an open and extensible framework in which various functions defined in the TPA can be supported by plug-ins. Examples of functions conveniently provided by plug-ins are document parsers and generators and security functions. Therefore, BPF is ideal for setting up loosely coupled trading agreements in which, for example, a business is free to replace suppliers by others without having to make a large IT investment to support the new supplier. As long as the business protocol, and hence the TPA, does not fundamentally change, replacing business partners by others is practical.

BPF architecture. Figure 3 illustrates the role of BPF as the gateway, coordinator, and control point of choice between intrabusiness and interbusiness processes. It is positioned between the buy and sell component of a local business, the remote businesses, and the back-end systems. The applications spanning multiple businesses (e.g., procurement and supply chain management) are numerous and can be built using this framework.

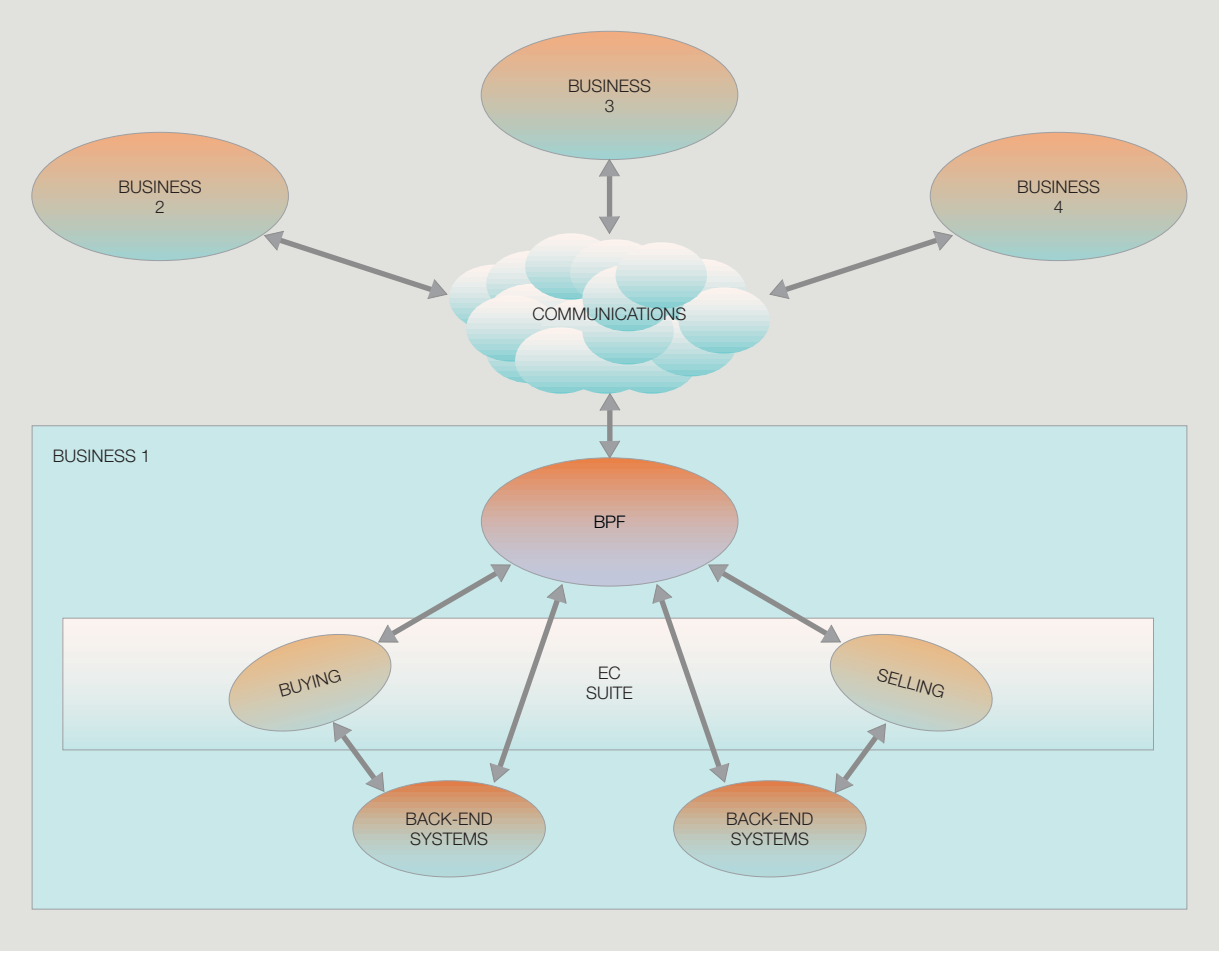
Figure 4 depicts the generality of BPF for different kinds of business protocols. With OBI as an example, the middle node in the figure could be a business sell-side OBI server, with requests coming from the requisitioner (top left). The server on the left of the figure would then be the OBI buy-side server, and the server on the top right would be the payment gateway. The local business process at the sell-side node (middle) could be the fulfillment process, which is invoked using the BPF framework. As another example, the figure could be the case of an RFQ process in which the buyer (node at the lower left) could submit an RFQ request to the seller or supplier (middle node). The seller would invoke its local business process to determine whether to respond to the RFQ; this may, in turn, involve requests to remote suppliers (on the right), which would be done using BPF. The seller could then send a response to the RFQ to the buyer. The buyer, in turn, could select from the responses received and send a purchase order to the selected seller.

A version of BPF is built on top of an Enterprise JavaBeans (EJB) server and employs a diverse set of technologies for providing various functions and services. Although IBM WebSphere* can be used as the EJB server, there is no specific dependency on WebSphere. A business may communicate with its partners using one of the several protocols such as HTTP, SMTP, and IBM MQSeries*. It may even use different protocols for different sets of actions on a per-TPA basis. Security technologies include transport security (SSL), authentication using digital certificates, as well as digital signatures for nonrepudiation (using MD5, SHA-1). Various data formats include EDI, XML-EDI, or other XML formats. Appropriate message parsers or message generators are provided for converting these documents into an internal format. Independent software components providing many of these technologies can be plugged in to the BPF framework via a vendor-neutral open application programming interface (API), thus allowing implementers to customize solutions of their choice.

BPF provides many different services to the business applications running on this platform. In addition to the basic TPA service, BPF provides services such as time-stamped logging, recovery services, public-private key cryptography, reliable document exchange, and document repository.

Layering. The functionality provided by BPF is layered, as shown in Figure 5, in order to provide different levels of abstraction for the business data flow. Furthermore, the layering, along with well-defined interlayer APIs, minimizes the spread of specialized code across the framework. The business data flow

Figure 3 BPF as business-to-business coordinator

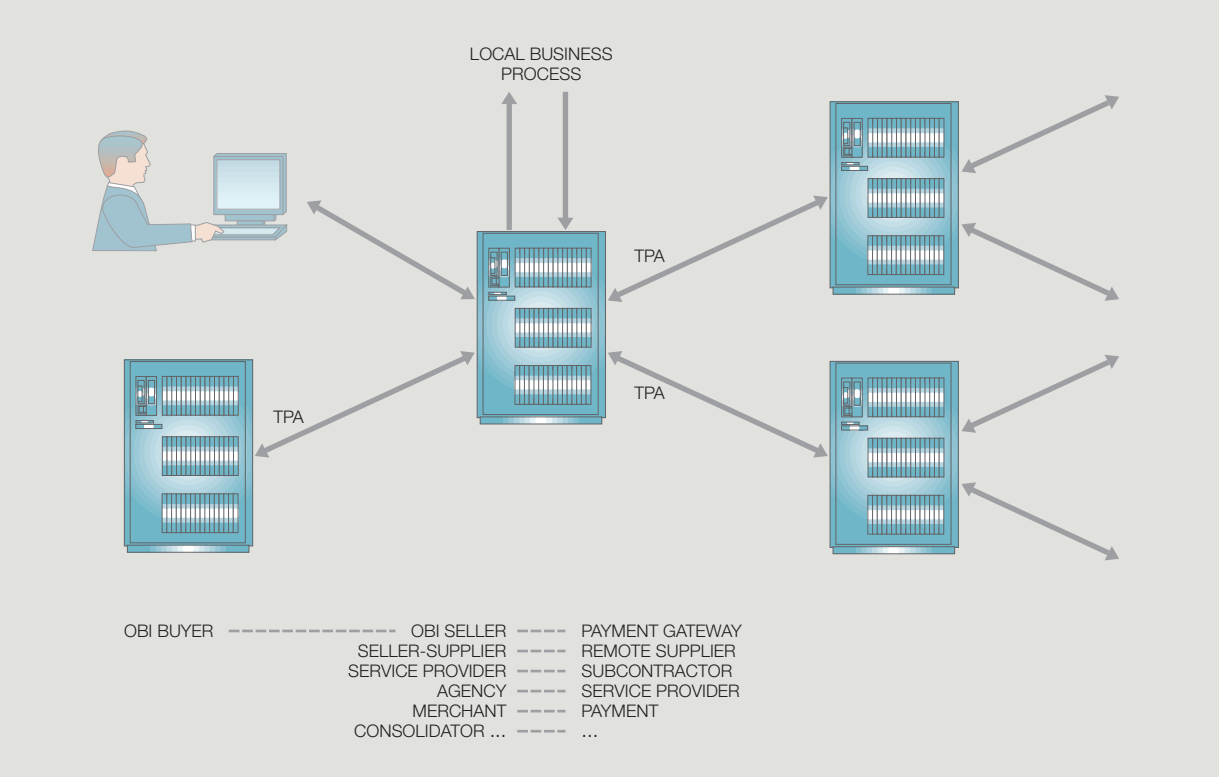


through the layers is driven by the TPA that governs that particular protocol.

The lowest level of the BPF stack is the transport layer, which contains protocol-specific modules that allow the business processes to communicate with the external world using any of the supported communication protocols, including communication-related security such as authentication and encryption. The transport layer interfaces with the document exchange layer, which supports the abstraction of a business document (e.g., EDI document). The document exchange layer provides common document-related functionality such as message data mapping, nonrepudiation, time-stamping, logging, and audit trail. The document exchange layer interfaces with the business protocol layer that provides document-type and trading-partner-specific data-handling functionality based on the business protocol section of the TPA. The business protocol layer in turn provides the business logic interface that connects to the specific business application that may implement the highest-level business logic or serve as a bridge to back-end business processes such as enterprise resource planning, or both. It is the business logic that processes the payloads of the messages exchanged under the TPA. This processing is not governed by the specifications in the TPA but is the responsibility of the application design.

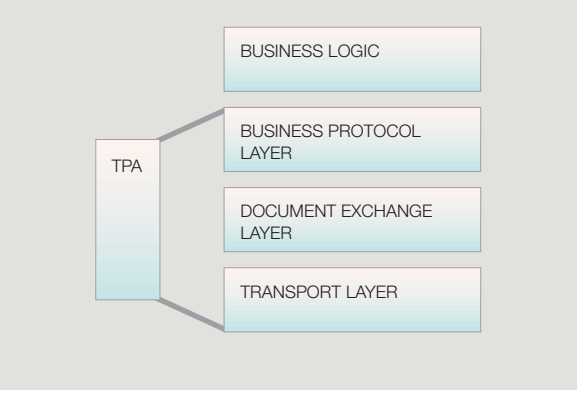
BPF components and flow. The components of BPF are shown schematically in Figure 6. The BPF func-

Figure 4 General process flow for BPF



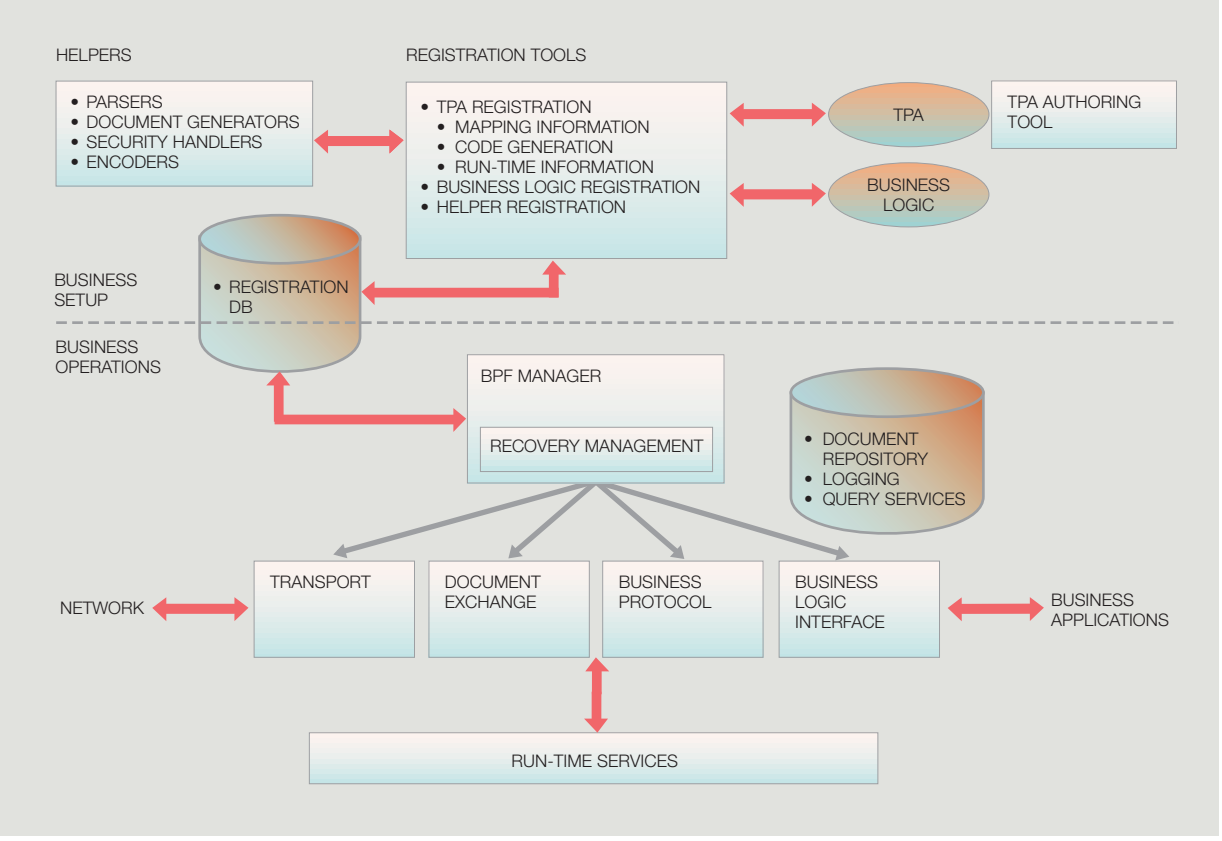
tions can be broadly categorized as those related to setting up the BPF environment and those related to run-time operations for business-to-business interactions. Above the horizontal line in the figure are the various helpers and tools; these will be discussed below. The tools cause information to be placed in the registration database for use during run time by the BPF manager. Below the line are the run-time functions, managed by the BPF manager, which also encompasses recovery management. In the middle of the section below the line are the major components of message flow. From left to right, messages arrive from the network into the transport function. From there, each document is passed through the functions that implement the rest of the functional layers, described previously. The business logic interface provides the interface to the business application that implements the business logic and to back-end processes such as enterprise resource planning. Finally, the run-time services are indicated at the bottom.

Figure 5 BPF functional layers



To support recovery and audit, all requests and replies (i.e., inbound and outbound messages) are time-stamped and logged. The information to be logged

Figure 6 Business setup and operations components



includes the document received, request time, document sent, reply time, owner of the document, internal service to be invoked, etc. The information could be used later for audit trail purposes and for recovering the state of a BPF server. The state of the TPA instance is recorded in an underlying persistence medium. On startup and after restart following a failure, recovery services are invoked to recreate the state of the TPA instance and associated BPF conversations. All incoming messages are analyzed, checking for duplicate copies as well as for allowable sequences specified in the TPA. If the message is acceptable, it is enqueued for handling asynchronously.

When BPF is deployed with IBM WebSphere Commerce Suite (WCS), the components of BPF work with WCS in order to provide functions such as communications (e.g., HTTP), user directory and access control, logging and error reporting, various application commands, and back-end system functions such as the connection to SAP**.

TPA authoring and code generation

In order to utilize an electronic TPA, the TPA must first be composed and agreed to by the participating parties. Because the TPA is a complex document, and a new TPA is needed to do business with each new trading partner, an authoring tool that understands TPA semantics and assists the author in providing the correct information 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 that were specified in the TPA with business functions of each business partner, so

that when an action is requested of a partner, the correct sequence of business functions is called.

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 both parties to the TPA use the same code generator and registration tool. We here describe the tools we developed as part of a project in IBM Research.[7] In our project, these tools are implemented in the Java** programming language.

The code-generation tool uses specification information from the TPA and the registration database to generate the TPA objects that are needed for each TPA to interface with the application. The tool may produce specific code for each TPA, or it may produce the information needed to customize a generic TPA object.

Authoring tool. TPA authoring tools play an important role in preparing TPAs. A TPA authoring tool understands TPA semantics. It guides the author in the process of creating a TPA. It validates the information that the author enters to create each tag in the TPA.

A TPA contains information about the agreed-upon interaction protocol (e.g., messages and allowable sequences) as well as the details of the interacting partners (e.g., contact information, URLs, and certificate parameters). In order to guide the creation of a TPA, the authoring tool captures the tpaML rules for creating TPAs in the form of models of individual tags and of the TPA as a whole.

The models contain this semantic information in a form that can later be used to construct a TPA. A model contains information that describes the details that the author must provide at each point in the TPA. For example, a model for HTTP indicates that URLs must be supplied as communication addresses. A model may also contain information from a specific TPA (e.g., specific URLs). A model based on information from a specific TPA can be used to create a similar new TPA such as a TPA between a different pair of partners. The authoring tool uses the information in the model to guide a user in creating a correct TPA. Thus, the authoring tool provides a way for an expert to prepare a model from which someone can construct a TPA with far less knowledge of the required semantics.

A model of a TPA consists of a collection of models of the tags to be used in the TPA. The models are in a tree structure that 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. A model of a tag can be simple, or it can contain multiple submodels for a particular tag. For example, a general model for the <Communication> tag contains a submodel for each of the supported communications protocols.

In many instances, a business partner (e.g., a seller organization) may use a standard application protocol (e.g., procurement protocols such as OBI, Ariba** cXML, and Metiom**) for interacting with many different partners. Authoring a new TPA based on a standard protocol requires only that the partner-specific information and certain choices (e.g., server-to-server versus server-browser-server in OBI) be updated. The TPA authoring tool can create a model from a sample TPA or a TPA template and can use this model to guide a user to create a TPA based on the sample.

The authoring tool starts with a document type definition 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 7 illustrates the process of creating a model and a TPA.

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 and guide the author in making choices and entering information. 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 or fill in the needed information. The authoring tool then continues with that choice.

Code generation or customization. The code generator or customizer transforms the TPA into registration information and code that enforces the rules

Figure 7 Creating a model

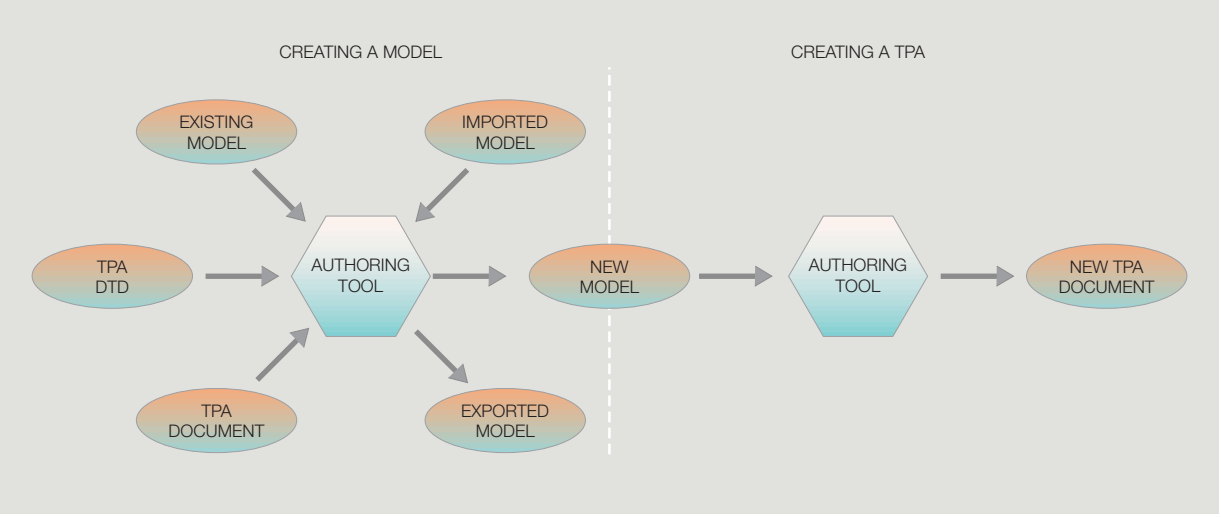
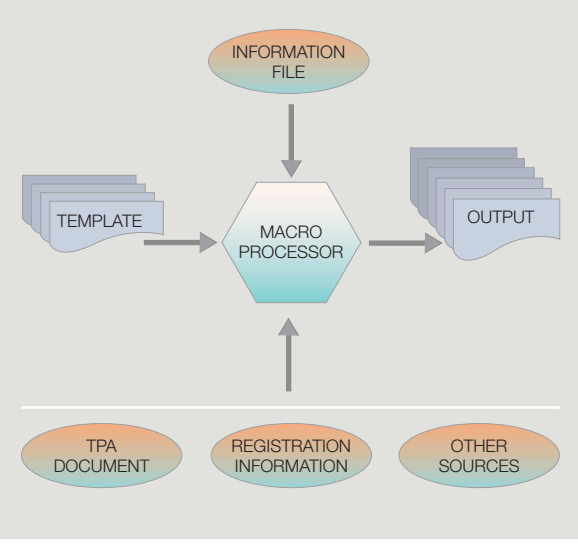


Figure 8 Code generation



of interaction. A TPA object is created at the site of each party to the TPA. Generation of specific code from a TPA is illustrated in Figure 8. Generation of specific code starts from a set of templates that 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 that change the execution flow, depending upon values of variables and functions.

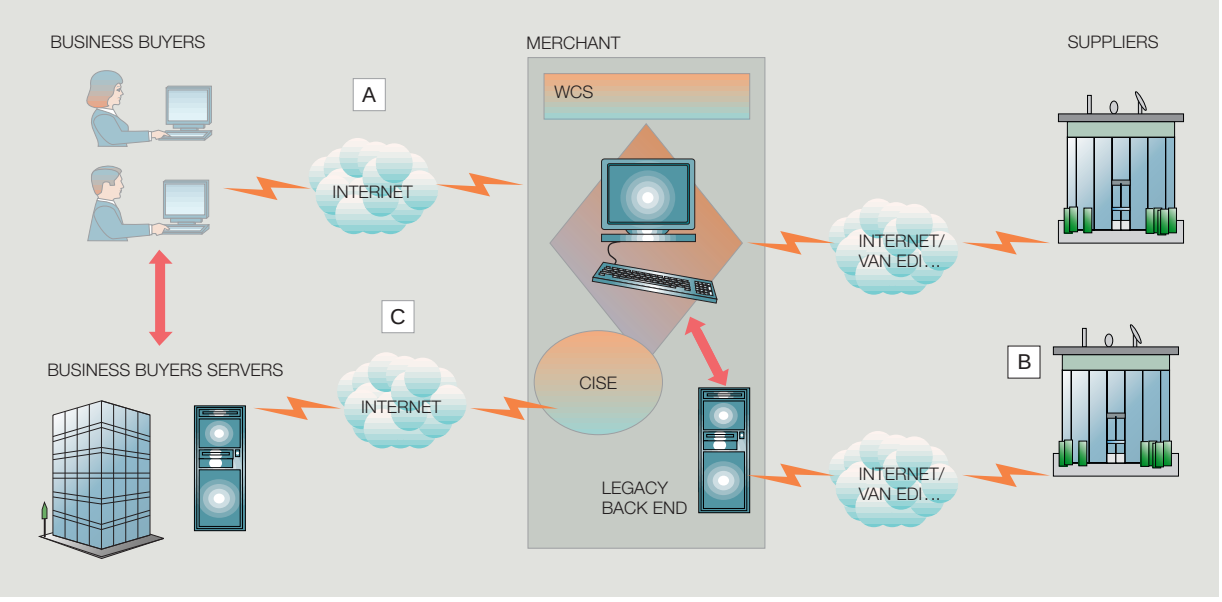
A macro processor scans the template looking for directives. It executes any directives it encounters and handles any native language statements as character strings, performing any needed processing, and writing the processed statements to a file.

The code customization approach starts with a generic TPA object. The registration tool then generates a TPA instance object that contains all the characteristics of each action along with all other runtime information from the TPA. At run time, the generic TPA object is used in conjunction with the TPA instance object, resulting in the behavior that supports the specific TPA.

Application examples

IBM WebSphere Commerce Suite, or WCS (formerly known as IBM Net.Commerce*), is a business application product for merchants to build on-line stores on the Internet. Many such stores are for consumer shopping. However, a growing market segment is the support of business buyers, as well as the support of business-to-business electronic transactions between the merchants and their business part-

Figure 9 WebSphere Commerce Suite and Commerce Integrator, Seller Edition



ners. Figure 9 illustrates some basic scenarios of a merchant WCS system dealing with its business partners. These scenarios are:

A. WCS merchants supporting business buyers, e.g., creating purchase order requests
B. WCS merchants dealing with their suppliers, e.g., checking for inventory
C. WCS dealing with their business buyer systems, e.g., requesting approval of purchase order requests

WCS in conjunction with IBM Commerce Integrator, Seller Edition (CISE) facilitates the support of these business scenarios when possible through the adoption of both open (e.g., OBI) and *de facto* (e.g., Ariba) business-to-business "standards." For example, WCS/CISE supports the OBI standard (Open Buying on the Internet), thus allowing any OBI-compliant business buyers to place purchase order requests with an OBI-enabled merchant and interacting to respond correctly to the approval process of the buying organization (i.e., scenarios A and C). Other examples of business protocols are cXML (supported by Ariba), Metiom (formerly known as Intelisys), and MySAP** (supported by SAP). A merchant chooses the business protocol to be used with a business buyer partner and then installs the specific CISE connector technology required by the customer. Electronic TPA and BPF technologies were used as a base to develop the tooling and run time for the CISE OBI support announced in September 2000.

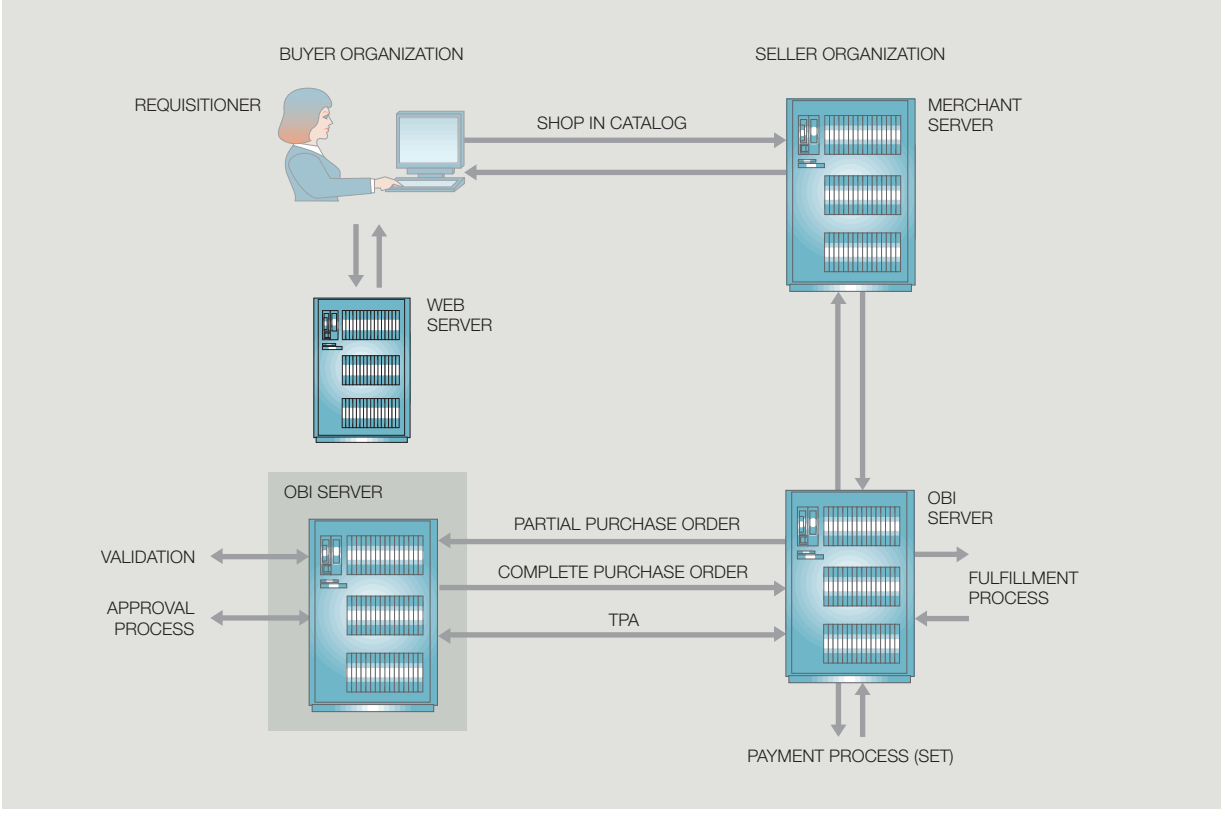
The following subsections describe in more detail the OBI business protocol, which is supported in WCS/CISE, and an OBI-like on-line procurement application of WCS.

Open Buying on the Internet. We now describe an example of the TPA and server structure for an existing public protocol, OBI.

Open Buying on the Internet (OBI),[3] 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 the purchasing activity of an organization. Here we present OBI, how it can be described by a TPA, and a schematic view of a possible implementation.

Figure 10 illustrates the participants in an OBI transaction and the basic information flows. The requisitioner is a member of the buyer organization (e.g., an employee of a company) and is permitted to place

Figure 10 Open buying on the Internet



orders directly with the merchant server of the seller organization. The requisitioner can browse a catalog and place an order with the seller organization by means of a browser. When the requisitioner has placed an order, the server of the seller organization sends a partial purchase order (purchase order request) to the server of the buyer organization. The buyer organization validates the purchase order request and transforms it into a complete purchase order that it returns to the seller organization. The seller organization then prepares an invoice or otherwise arranges for payment and ships the ordered merchandise. The payment process handles electronic payments. Using the browser, the requisitioner can also view and update information at the buyer organization server such as the requisitioner's profile, outstanding requests, etc. The requisitioner can also check the status of an order at the seller organization.

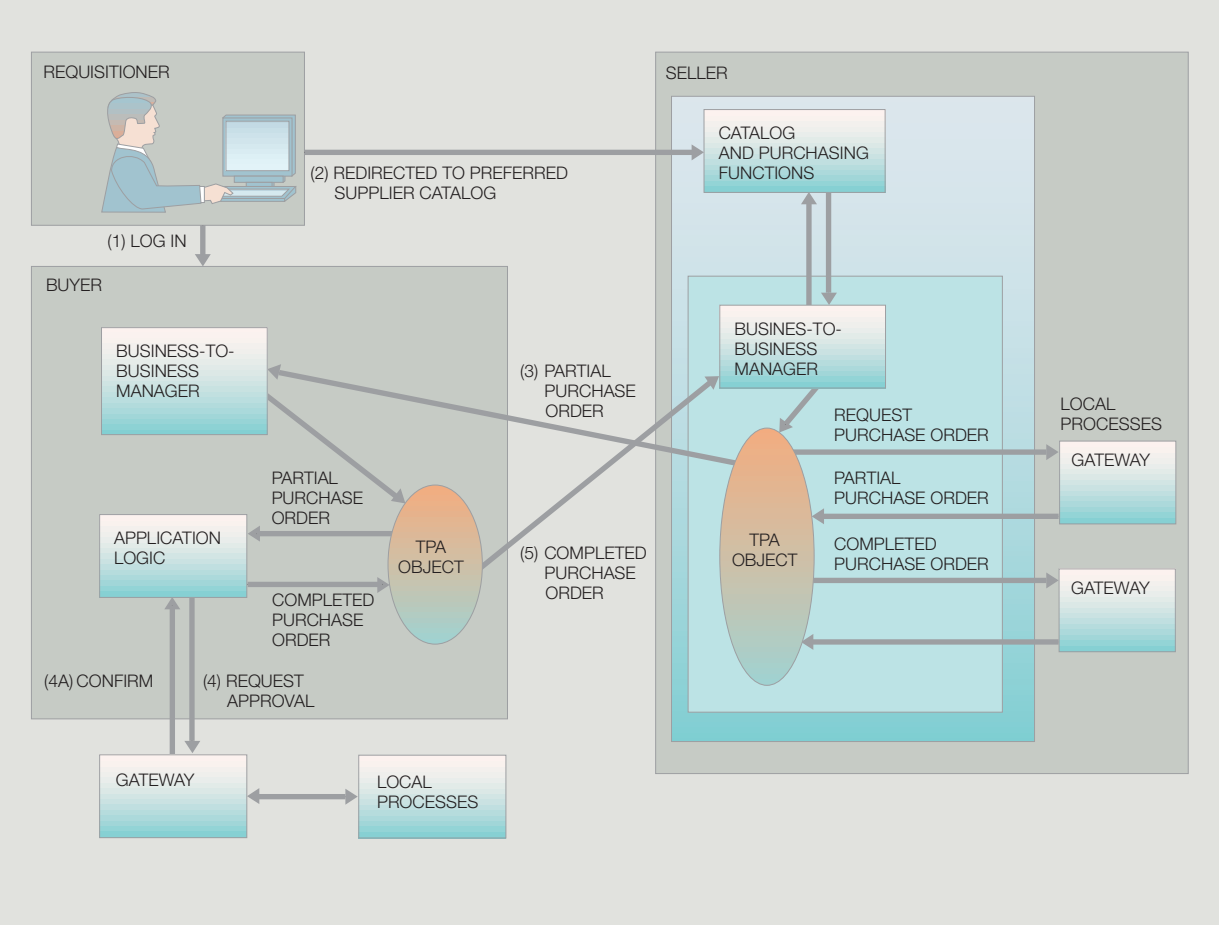
An additional possibility is to have the buyer organization send an "unsolicited" purchase order to the selling organization without a prior request and partial purchase order initiated by a requisitioner. This mode might be used, for example, when a purchasing department buys large volumes to supply a stock room.

As shown in Figure 10, there is a TPA between the business-to-business servers of the buyer organization and the seller organization. The payment process is outside the scope of the two-party TPA between buyer organization and seller organization. It is a back-end process of the seller organization.

Following are the main functions included in the OBI TPA:

- Organization names of the parties to the TPA
- Communication protocol definition, which in this case 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

Figure 11 OBI implementation



- Action menus for the buyer and the seller. The action list for the buyer is illustrated in the earlier subsection entitled "Action definition." It consists of one action, "Process OBI Purchase Order Request." The completed purchase order is returned to the seller by means of an action request to the seller organization, "Handle OBI Purchase Order." The buyer organization may also use the "Handle OBI Purchase Order" action to submit an unsolicited purchase order to the seller organization.

Figure 11 depicts the basic system structure and flow of an implementation of OBI based on BPF. 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.

The process starts when a requisitioner contacts (1) the buyer server via a browser and is redirected (2) to the URL for the seller's catalog and purchasing functions. 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 seen 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 (4A), 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 `handleOBIPO` action request (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 and payment. They also initiate a confirmation message to be returned to the requisitioner via the browser (not shown in the figure).

IBM-customer pilot on on-line procurement. IBM and a large bank collaborated on a pilot of on-line procurement. The bank wanted to reduce its procurement costs by automating direct purchases of IBM computers by its employees. Bank employees can use this system to purchase IBM personal computers and accessories directly from the IBM Personal Systems Group (PSG).

The selling organization (IBM PSG) system was based on BPF alpha code and WebSphere Commerce Suite. It maintained the catalog of products and prices for use by the purchasing organization (the bank). The IBM platform consisted of an IBM Netfinity* server running the Microsoft Windows NT** Server operating system, IBM WebSphere Application Server Advanced Edition Version 3, the IBM HTTP Server Web server, and IBM Universal Data Base.

The procurement system of the bank was based on the Metiom Enterprise Purchasing System**. The Metiom system provided services for creating, approving, tracking, and modifying purchase orders. It provided a single point of access, with a single sign-on, to all supplier catalogs used by the bank. IBM provided a personal-computer catalog. Bank employees could browse this catalog and then place orders.

The message exchange protocol between the buyer and seller systems is a private protocol designed by Metiom, which is based on OBI, and is enhanced for this application with the addition of order-acknowledgment and invoice messages. A TPA was written to describe the process and used to configure BPF at the IBM PSG server.

Figure 12 shows the main components and message flows. The IBM PSG server consists of BPF and WCS. Following are the blocks in the BPF part of the server. Supplier Business Logic is the procurement application. The OBI business-to-business protocol block is the TPA object. The OBI messaging block is the document-exchange layer of BPF. The HTTP/HTTPS block is the BPF transport layer. The IEC-Link** is Metiom's Enterprise Commerce Link**, which is a mailbox communication system. A poll message is passed to the IEC link, which polls the mailboxes for messages and returns them to the procurement system. The use of the IEC link in this application will be discussed below.

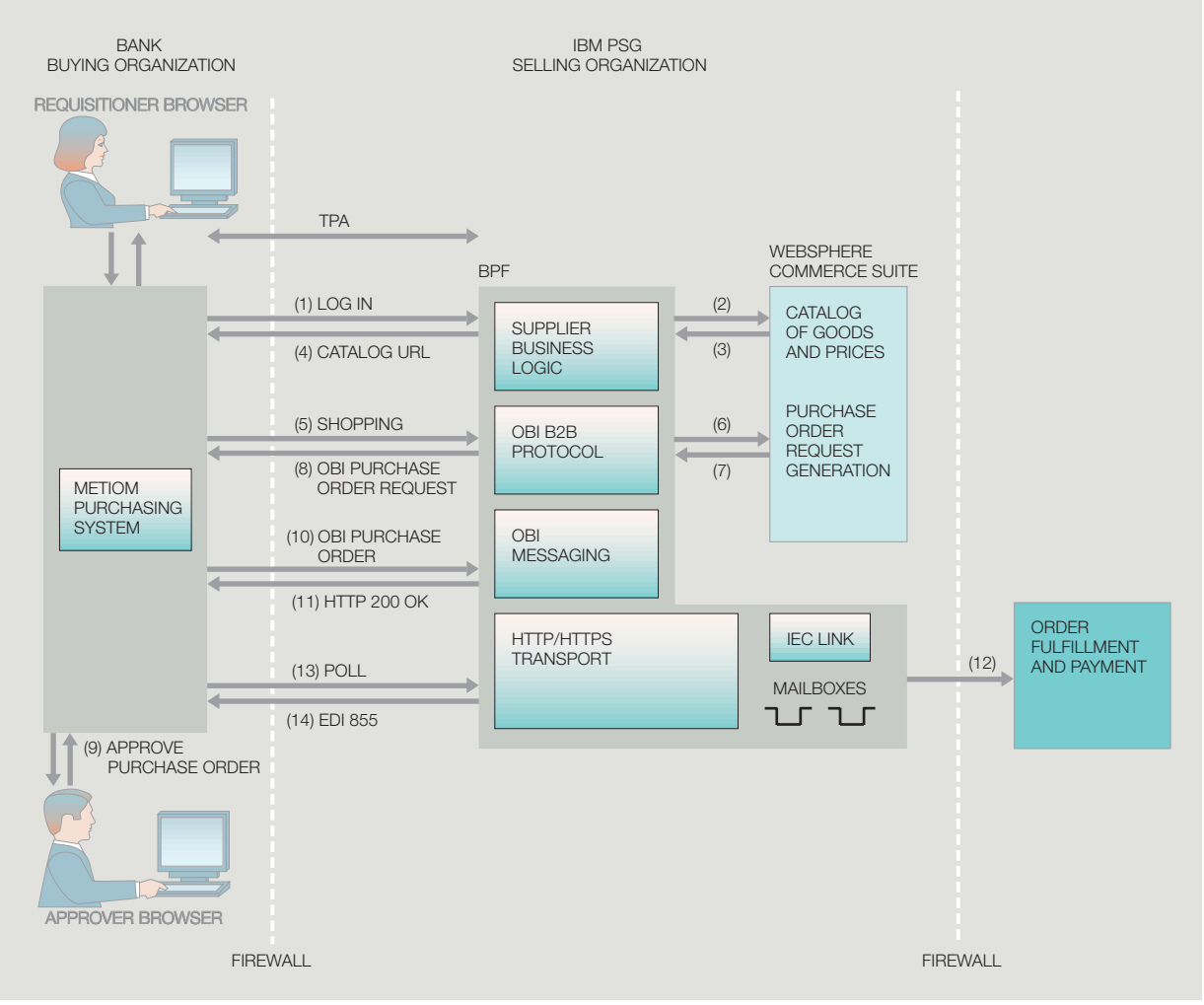
The employee (requisitioner) who wants to purchase a personal computer contacts the bank purchasing system using a browser. The purchasing system presents the employee with a list of suppliers that are approved by the bank. The employee selects IBM PSG. The purchasing system then logs onto the IBM PSG system (1) via BPF. BPF in turn contacts the catalog system in WCS (2), which returns the URL for the IBM PSG catalog appropriate for this bank (3). The URL is passed to the requisitioner's browser (4).

The requisitioner is now enabled to shop (5, 6) and fill a shopping cart. The WCS server creates an order, based on the shopping cart, and passes it to the seller server (7), which generates a purchase order request as an OBI Purchase Order Request message and returns it to the buyer server (8). An approval process now occurs at the bank system, which may involve a human approver at a browser (9). When the approval process is completed, the bank server issues a purchase order (10) to the IBM PSG server, which responds with an HTTP acknowledgment (11). The purchase order is passed to the order fulfillment and payment subsystem (12), and the purchase is shipped directly to the requisitioner. The bank system then polls the IBM PSG system for messages (13) and receives an EDI 855 order-confirmation message (14).

We have demonstrated end-to-end operation of the pilot. At the time this paper was written, the pilot was ready to be put into service by the bank for actual purchases by employees.

This example shows that a practical e-commerce application will often have localized but significant differences from the documented standard. In this example, the Metiom system requires the use of a user identifier and password for authentication instead

Figure 12 On-line procurement study



of certificate authentication, probably because it is easier to administer than client certificates. It also requires the use of a server-browser-server protocol in which response messages from the selling organization are returned to the buying organization via the requisitioner's browser in order to support the firewall of the bank. Further, the Metiom system requires the use of the EDI 855 order-confirmation message, which is not part of the OBI standard. The flexibility in tpaML, combined with the TPA authoring tool and its tag models, exactly meets the need for easy local tailoring of the protocol and specific partner addressing information defined by a standard protocol.

Summary, conclusions, and future work

A large number of business-to-business interaction protocols have emerged in recent years for automating various e-commerce processes, such as Open Buying on the Internet (OBI), cXML supported by Ariba, Commerce One RoundTrip** Service, and RosettaNet Partner Interface Processes** (PIPs**). In keeping with this pace of automation, many more business protocols in the form of new protocols, enhancements to existing protocols, and even private protocols across a set of businesses will be used in the future. The complexities of these protocols are also expected to increase in order to capture many

more aspects of the real-world interactions. Implementation of all such protocols from the start is time-consuming and expensive. BPF provides a comprehensive set of tools and enablers for ease of specification, configuration, plug-in, and customization for setting up such business-to-business interactions. To use BPF, electronic TPAs, which may specify either standard or custom application protocols, are created according to the tpaML specification and registered to BPF along with the internal business processes to be invoked using these TPAs. BPF generates code for linking and enforcing these TPAs and provides many other services (e.g., conversation correlation, cancellation, and rule-based invocation of multistep logic) for writing complex business applications. We have described two applications of TPA and BPF that use OBI or a variant of OBI. As e-commerce becomes pervasive, many new applications (marketplaces, agencies, distributors, etc.) will be built on this foundation.

We are extending the TPA ideas and language to areas such as TPA hierarchy, linking of multiple TPAs, TPA life-cycle management, 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.

Acknowledgments

The authors express their appreciation to the following for contributions to the design of BPF and the formulation of the TPA principles and language: Satwinder Brar, Catherine Crawford, Christine Draper, Christopher Gibson, Vibby Gottemukkala, John Ibbotson, Richard King, George Kleon, Linh Lam, Keith Mantell, Paul Norris, Stewart Palmer, Chris Sharp, and Colin Thorne. The authors also thank Nagui Halim and Anant Jhingran for their management vision.

Cited references and note

1. XML is a "meta-language" that can be used to define and describe markup languages for various classes of documents. It is based on Standard Generalized Markup Language (SGML), an international standard used to define electronic documents.
2. Extensible Markup Language (XML), 1.0, World Wide Web Consortium (1998).
3. *Open Buying on the Internet Technical Specifications*, Release V1.1, The Open Buying on the Internet (OBI) Consortium, http://www.openbuy.org (1998).
4. *RosettaNet Specifications*, RosettaNet Consortium, http://www.rosettanet.org (1999).
5. Electronic Business XML initiative established by the United Nations body for Trade Facilitation and Electronic Business and the Organization for the Advancement of Structured Information Standards, http://www.ebxml.org (2000).
6. A. Dan and F. Parr, "An Object Implementation of Network Centric Business Service Applications (NCBAs)," *OOPSLA Business Object Workshop*, Atlanta, GA (September 1997).
7. A. Dan, D. Dias, T. Nguyen, M. Sachs, H. Shaikh, R. King, and S. Duri, "The Coyote Project: Framework for Multi-Party e-commerce," *Proceedings of Research and Advanced Technology for Digital Libraries*, *Second European Conference*, ECDL'98, Heraklion, Greece (September 1998), Springer-Verlag, Berlin (1998), pp. 873–889.
8. A. Dan and F. Parr, "The Coyote Approach for Network Centric Business Service Applications," *HPTS Workshop*, Asilomar, CA (1997).
9. H. Weigand and A. Ngu, "Flexible Specification of Interoperable Transactions," *Data and Knowledge Engineering* **25**, 327–345 (1998).
10. J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan-Kaufmann, San Mateo, CA (1993).
11. T. Ajisaka, "Electronic Commerce for Software," *Proceedings of Research and Advanced Technology for Digital Libraries*, *Second European Conference*, ECDL'98, Heraklion, Greece (September 1998), Springer-Verlag, Berlin (1998), pp. 791–800.
12. T. Sandholm, "Unenforced e-commerce Transactions," *IEEE Internet Computing* **1**, No. 6, 47–54 (November–December 1997).
13. T. Sandholm and V. Lesser, "Issues in Automated Negotiation and Electronic Commerce: Extending the Contract Net Framework," *Proceedings of First International Conference on Multi Agent Systems*, ICMAS 95, San Francisco, CA (June 1995), AAAI Press, Menlo Park, CA (1995), pp. 328–335.
14. P. Konana, A. Gupta, and A. Whinston, "Digital Contract Approach for Consistent and Predictable Multimedia Information Delivery in Electronic Commerce," *Multimedia Computer and Networking 1997*, San Jose, CA (February 1997), *Proceedings of SPIE—International Society for Optical Engineering* **3020** (1997), pp. 275–281.
15. A. Dan and F. Parr, "Long Running Application Models and Cooperating Monitors," *HPTS Workshop*, Asilomar, CA (1999).
16. *The Common Object Request Broker Architecture and Specification*, Rev. 2.2, Object Management Group, http://www.omg.org (1998).
17. *Enterprise JavaBeans Specification*, ver. 1.1, http://www.javasoft.com/products/ejb (1999).
18. *The Workflow Management Coalition Specification*, http://www.wfmc.org (1998).

19. H. Garcia-Molina and K. Salem, "SAGAS," *Proceedings of ACM SIGMOD Conference*, New York (1987), pp. 249–259.

Accepted for publication October 10, 2000.

Asit Dan *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: asit@us.ibm.com).* Dr. Dan has been with IBM Research since 1990 and currently manages the business-to-business integration department, working on the development of infrastructure for supporting business-to-business e-commerce applications. He is at the forefront in the research and development of transaction processing architectures and video servers. He holds several top-rated patents in these areas and has received two IBM Outstanding Innovation Awards, seven Invention Achievement Awards, and the honor of Master Inventor for his work in these areas. Dr. Dan received a Ph.D. from the University of Massachusetts, Amherst. His doctoral dissertation, *Performance Analysis of Data Sharing Environments*, received an Honorable Mention in the 1991 ACM Doctoral Dissertation Competition and was subsequently published by the MIT Press. He has published extensively, including several book chapters, and a book on multimedia servers.

Daniel M. Dias *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York (electronic mail: dias@us.ibm.com).* Dr. Dias received the B. Tech. degree from the Indian Institute of Technology, Bombay, India, and the M.S. and Ph.D. degrees from Rice University, all in electrical engineering. He has been with the IBM Research Center in Yorktown Heights since 1983. He manages the Parallel Commercial Systems department, which currently has projects focusing on scalable and high-performance Internet servers, business-to-business e-commerce, and performance management. His recent work includes scalable and highly available Web servers, frameworks for business-to-business e-commerce, high-performance scalable Web caches, scalable video servers, highly available clustered systems, and performance analysis. Technologies developed in these projects have been used for the 2000 Olympics Web site and large customer sites. Some are now available as IBM products such as Network Dispatcher, Web Cache Accelerator, and HACMP ES. Dr. Dias has published more than 100 papers in refereed journals and conferences. He has won two best paper awards, IBM Outstanding Innovation and Outstanding Technical Achievement Awards, ten Invention Achievement Awards, and Research Division Awards. He holds 18 U.S. patents, with 15 additional patents pending.

Robert Kearney *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York (electronic mail: firefly@us.ibm.com).* Dr. Kearney has been employed at IBM for 30 years, with the last 18 at the Research Center. He is currently working in developing e-commerce frameworks. Prior to this work, he helped develop frameworks for clinical information systems, networked document retrieval and processing systems, and insurance industry systems, all as partnerships between industry, government organizations, and IBM Research. His education is in mathematics, having attended the University of Massachusetts, University of Wyoming, and Pennsylvania State University.

Terry C. Lau *IBM Canada Laboratory, 1150 Eglinton Avenue East, North York, Ontario, Canada M3C 1H7 (electronic mail: lautc@ca.ibm.com).* Dr. Lau is a senior system architect in the Department of Electronic Commerce Development at the IBM Canada Laboratory. His current activity is business-to-business e-commerce. Previously, he has been in various technical and management positions in the areas of data communications, imaging, and graphical user interface application development tools. Before joining IBM, he was a faculty staff member at the University of Hong Kong and a development manager at Northern Telecom in data communications. Dr. Lau received a B.Sc. from the University of Hong Kong and a Ph.D. in computer science from the University of Waterloo.

Thao N. Nguyen *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: tnnguyen@us.ibm.com).* Dr. Nguyen received a B.S.E.E. degree from the University of New South Wales, Australia, and M.S.E.E. and Ph.D.E.E. degrees in microelectronics from Stanford University. Since joining the Research Center in 1983, he has worked on a broad range of research and development projects and held various technical and management positions. During the first six years he performed and managed research in VLSI processing technologies and material, and process characterization. He spent the next two years in semiconductor product development at the IBM East Fishkill semiconductor facilities, first as an executive technical assistant and later as senior engineering manager of process integration. From 1991 to 1996 he participated in the development of an advanced RISC microprocessor for RS/6000™ and then led the floorplanning and chip integration work in a project to produce a highly successful single-chip CMOS microprocessor for S/390™ systems. His recent activities are focused on software and systems for business-to-business e-commerce. He has worked on the development of a framework for business-to-business applications and integration as well as an OBI supplier solution package. He is currently engaged in an effort to develop and deploy business-to-business commerce servers for IBM as a supplier to large enterprises and e-marketplaces. Dr. Nguyen has authored or coauthored more than 40 technical papers and has been awarded several patents in silicon and circuit technologies. He is a recipient of several Research Division Awards, a Technical Group Award, and two IBM Outstanding Technical Achievement Awards. He has served on the Technical Program Committee of several IEEE conferences including the Symposium on VLSI Technologies and Symposium on VLSI Circuits.

Francis N. Parr *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: fnparr@us.ibm.com).* Dr. Parr is a research staff member in the Computer Sciences department at the Research Center and also responsible for the Transaction and Messaging Technology Institute—a joint program between IBM Research and the IBM Hursley Development Laboratory. He is currently engaged in research on business-to-business middleware with previous interests in messaging and message brokering, object middleware, parallel database, and scalable transaction systems. Before joining IBM, Dr. Parr was a lecturer in computing at Imperial College of Science and Technology, London University. He received a Ph.D. in applied math from Harvard University and a B.A. in mathematics from Cambridge University.

Martin W. Sachs *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: mwsachs@us.ibm.com).* Dr. Sachs is a research staff member in the Department of Computer Sciences at the Research Center. His current activity is business-to-business e-com-

merce, focusing on electronic trading-partner agreements. He is leading the ebXML team that is developing the specification for the standardized version of the IBM tpaML electronic trading-partner agreement language. Previously, he specialized in I/O interconnect architecture including contributions to the IBM System/390 fiber-optic ESCON™ I/O Architecture and the ANSI Fiber Channel standard. Before joining IBM, he worked in nuclear reactions and in computer-based nuclear data acquisition at the Weizmann Institute of Science, Israel, and Yale University. Dr. Sachs received an A.B. degree in physics from Harvard University and M.S. and Ph.D. degrees in nuclear physics from Yale University. He is an IEEE Fellow and a member of Sigma Xi, the ACM, and the American Physical Society.

Hidayatullah H. Shaikh *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: hshaikh@us.ibm.com).* Mr. Shaikh is an advisory software engineer in the Parallel Commercial Systems department at the Research Center. His current activity is business-to-business e-commerce, focusing on defining a flexible and scalable framework for new and existing business-to-business protocols. His contributions include ebXML header specification and IBM tpaML electronic trading-partner agreement language. Previously, he has been involved in the architecture and design of the IBM Supplier Live solution for Ariba Buyer and implementation of Java Transaction Services. Mr. Shaikh received an M.S. degree in computer engineering from Syracuse University.