RC21502 (96990) 8JUN99
Computer Science/Mathematics 19 pages

Research Report

Executable Business to Business Contracts in Electronic Commerce

Martin Sachs, Asit Dan, Thao Nguyen, Robert Kearney, Hidayatullah Shaikh, Daniel Dias
IBM Research Division
T.J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Executable Business to Business Contracts in Electronic Commerce

Martin Sachs[1], 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 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. We describe the basic principles of electronic contracts. The contract expresses the rules of interaction between the parties 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 contract language. We then describe tools for authoring contracts and generating code from them. Finally, we describe examples of applications which can benefit from contracts.

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 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 contracts 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 will accurately embody the desired terms and conditions. In the longer term, electronic contracts 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).

[1] 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 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.

Ajisaka 1995 discusses software as an object of electronic commerce and proposes an architecture for managing custom software development by contract. Sandholm 1997 describes algorithms for modeling electronic commerce transactions that don't require enforcement. Sandholm & Lesser 1995 discuss issues in automated negotiation among agents whose rationality is bounded by computational complexity. Konana *et al* 1997 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 mentioned above 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 contract and the tools to assist in composing the contract 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 contracts. In section 4, we describe our contract language. In section 5, we describe the tools for creating contracts and generating code from them. Finally, in section 6, we give application examples which illustrate the use of the contract.

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. An application component invokes other application components synchronously either within an existing ACID transaction context (Corba 1998, Ejb 1999) or in a new transaction context for execution of the invoked application components. 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). Various extended transaction semantics, e.g. SAGAS (Garcia-Molina & Salem 1987), nested transactions (Gray and Reuter 1993), compensation sphere (Leymann & Roller 1997) may be associated for managing the execution state of such long running applications. Additional issues such as people assignment for executing a step are addressed by workflow systems (Wfmc 1998, Leymann & Roller 1997).

The above methodologies 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 software platforms (e.g., consider the issues on security, naming, component registration). The manual process of setting up such a bus across hundreds of business partners will be time consuming and highly undesirable. Finally, all the business partners have to agree to implement a common standard and many different issues on implementation standard need to be sorted out in terms of management of such a shared bus. Currently, Internet Protocol (IP) is the only common standard implemented by all partners.

Even if such a software bus can be established, the ACID and/or complex extended transaction models of stateful interactions (as mentioned above) 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.

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 contract or trading partner agreement. The automated process of setting up this interaction from an unambiguous formal specification and enforcing contractual agreements is termed an ***executable contract***. 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 Contracts

The purpose of the electronic contract is to express the IT terms and conditions to which all parties to the contract must agree in a form in which configuration information and the interaction rules which must be executable can be automatically generated from the contract in each party's system. It should be understood that the information in the contract 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 contract 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 parties to the contract.

We emphasize that the contract is formulated to ensure that each party maintains complete independence from the other parties 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 contract neither requires, nor provides the means for, ACID transactions involving multiple 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 other parties and receive service requests from other parties. 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. One of the examples in section 6 includes a party which is both a client and a server.

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

The contract 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 the travel application described in the section 6, the contract might define the interactions starting with making the different reservations needed by the traveler, to the check-in processes during the trip, and ending with the confirmations between the airline and hotel when the traveler checks out. This sequence of steps is a single long-running conversation. A unit of business is performed under the contract by instantiating the contract as a long-running conversation. To perform many units of business, the contract may be instantiated as 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 contract. We now give a brief overview of these functions. Section 4 describes the actual contract language.

Overall properties
Role
Identification
Communication properties
Security properties
Actions
Sequencing rules
Constraints
Cancellation rules
Error handling

Figure 1: Key contract elements

Overall properties of the contract include its name, starting and ending dates, and similar global parameters. The role section provides the means to define a contract in terms of generic roles such as airline and hotel and to produce a specific instance of the contract 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. HTTPS, IIOP), 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 other parties can request and various characteristics of those actions. Sequencing rules specify the order in which actions can be requested on each server. Constraints state conditions which must be fulfilled in order for an action to be considered successful. Cancellation rules indicate whether the result of a completed action is cancelable, and if so, any constraints on cancellation such as a time range during which cancellation is permissible. 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 Contract Language

The contract 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 contract document is described by an XML Document Type Definition (DTD) file, which defines the tree structure of the contract 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 contract.

4.1 Overall Structure

The overall XML structure of the contract 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.

```
<CoyoteContract>
<ContractInfo>  <!-- contract preamble -->
     ...  <!--contractname, role definitions,
          participants, etc.-->
     </ContractInfo>
<Communication>
     ... <!--communication information-->
     </Communication>
<Security>
     <!--security information-->
     </Security>
<ServiceInterface>  <!-- one block for each provider-->
     ...
     </ServiceInterface>
<LegalText>
     <!--text for legal conditions which do not generate
         code-->
     </LegalText>
</CoyoteContract>
```

4.2 Roles

When a given contract can be repeatedly reused for different groups of parties, it can be written in terms of role parameters rather than specific party names. The authoring tool can then generate a specific contract 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 `<ContractInfo>` tag. Here is the XML for the role definitions for a contract between an arbitrary airline (`@airline`) and an arbitrary hotel (`@hotel`). Each `<RoleDefn>` tag supplies a pair of role parameter and actual name. In this example, the tags under `<Role>` particularize the contract 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.

4.3 Communications and Security

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 all parties (e.g. HTTP, HTTPS, MQSeries, IIOP) each party's address parameters, maximum allowed network delay, and other parameters. The `<ContractIdempotency>` tag specifies whether messages are to be checked for duplicates. Following is an example of the communication definition for HTTPS:

```
<Communication>
     <Protocol>HTTPS</Protocol>
     <Version>version</Version>
     <Node>
          <!--One node for each party-->
          <OrgName>name_of_party</OrgName>
          <HTTPSAddress>
               <URL>url</URL>
               <!--additional URL tags as needed>
               </HTTPSAddress>
          </Node>
     <Encoding>code_name</Encoding>
```

```
          <!--Currently, must be BASE64-->
     <ContractIdempotency>option</ContractIdempotency>
          <!--option is yes or no-->
     <NetworkDelay>time</NetworkDelay>
     </Communication>
```

The security properties section (`<Security>` tag) defines the security protocols to be used in performing the contract. Protocols are defined for transport security (authentication) and message security (nonrepudiation). Information supplied for authentication includes the type of authentication (e.g. password or certificate), the specific protocol (e.g. SSL), and various parameters defining the certificate. Information supplied for nonrepudiation includes the protocol (e.g. Digital Signature), hash function, encryption algorithm, signature algorithm, and certificate parameters.

4.4 Action Definition

For each party to the contract 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 Examples"). For reading convenience, we have highlighted the main subdivisions.

```
<Action>
  <Request>
    <RequestName>processOBIPOR</RequestName>
    <RequestMessage>
      <MsgType>HTTP POST</MsgType>
      <MsgContent>OBIPOR</MsgContent>
      </RequestMessage>
    <ResponseCode>
      <RCType>HTTP RETURN CODE</RCType>
      <RCContent>HTTP 200 OK</RCContent>
      </ResponseCode>
    </Request>
  <CallBack>
    <CallBackName>handleOBIPO</CallBackName>
    <CallBackMessage>
      <MsgType>HTTP POST</MsgType>
      <MsgContent>OBIPO</MsgContent>  <!--completed PO-->
      </CallBackMessage>
    <ResponseCode>
      <RCType>HTTP RETURN CODE</RCType>
      <RCContent>HTTP 200 OK</RCContent>
      </ResponseCode>
    </CallBack>
  <Cancellation>
    <Cancelable>No</Cancelable>
    </Cancellation>
  <Rules>
    <Idempotency>Yes</Idempotency>
      <!--Yes means discard duplicate messages-->
    <Constraint>
      <!--constraint expressions go here-->
      </Constraint>
    </Rules>
  <ActionServiceTime>
    <AsynchronousReply>
      <ServiceTime>3600</ServiceTime>
      <Presume>result</Presume>
           <!--Result is success or fail-->
      </AsynchronousReply>
    </ActionServiceTime>
  </Action>
```

The request name is `processOBIPOR`, i.e. the action transmits a purchase-order request to the OBI buyer. The message type is `HTTP POST` and the message content is the PO request. The response is of type `HTTP RETURN CODE` and is normally `HTTP 200 OK`.

The `<CALLBACK>` tag indicates that the response is by means of a callback from the OBI seller server to the OBI buyer server and that the callback invokes method handleOBIPO at the issuer of the action (here, the OBI seller server). The callback is an `HTTP POST` and transmits a completed purchase order (`OBIPO`). The response to the callback, which goes from buyer to seller, is an `HTTP RETURN CODE`, `HTTP 200 OK`.

The `<Rules>` tag defines rules which must be satisfied in processing the action request. In this example, the `<Idempotency>` tag (value `Yes`) specifies that duplicate messages are to be discarded. Constraint expressions (discussed below) can also be placed under the `<Rules>` tag.

The `<ActionServiceTime>` tag specifies the worst case service time for the server (in this case, the OBI seller server). `<AsynchronousReply>` specifies the worst case service time until the callback, i.e. until the completed purchase order is returned. In this case, it is 3600 seconds, i.e. 1 hour. If the specified time is exceeded, the `<Presume>` tag specifies whether the request can be assumed successful or failed. If the presumption is failure, 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:

```
<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.

Constraints are rules which qualify a state transition. For example, receipt of a callback by the requester results in a transition from the "action pending" to the "action completed" state. The action is successful only if all the constraints are satisfied. A constraint expression is a Boolean expression which is evaluated by the framework during or at the conclusion of executing an action. A single constraint has the form

```
          <Constraint>expression</Constraint>
```

where *expression* is of the form `A op B` and `op` is a comparison operator as defined in Java except that `<` and `>` are replaced by the XML entities `&lt;` and `&gt;`.

Consider an example in which a purchase action is requested and the successful response message includes a delivery time expressed as the number of days from acceptance of the order. The action is successful only if the delivery time is within the agreed-upon time range, less than 5 days. The following constraint expression tests the delivery time:

```
     <Constraint>delivery_time &lt; 5</Constraint>
```

Constraints may be combined by AND and OR as follows:

```
     <Constraint combine=op>  <!--op is "AND" or "OR"-->
          <Constraint>expression</Constraint>
          <Constraint>expression</Constraint>
               ... <!-- as many as needed-->
          </Constraint>
```

We plan to extend the definition of constraints to allow reference to quantities outside the context of the current action. An example is quantities which were in messages prior to the invocation of the current action.

Cancellation rules define the conditions under which the results of an action may be canceled. The value of the `<Cancelable>` tag (`yes` or `no`) indicates whether the action is cancelable. Constraint expressions may be included to further specify whether the action is cancelable. For example, a constraint expression may specify the time period during which a hotel reservation is cancelable (e.g. no later than 4 p.m. on check-in day). An example of such a constraint is

```
  <Constraint>Deadline - CancelTime &gt; 8 hours</Constraint>
```

Many error conditions are handled in standard ways by the framework and their handling is not specified in the contract. In some cases, such as failure to receive a callback, the recovery is to cancel (if cancellation is allowed) the results of the action, if it was actually completed, using a cancellation action specified in the contract. Some errors, such as sequencing errors, may be severe enough for the framework to invoke the arbitrator. Duplicate messages are most likely to arise during failure recovery when incomplete actions are retried. The contract can specify that if the recipient recognizes a duplicate message, it can be ignored.

5. Contract Authoring and Code Generation

In order to utilize an electronic contract, the contract must first be composed and agreed to by the parties. Then registration information must be extracted from the contract 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 all parties to the contract. We here describe the tools we are developing as part of the Coyote project (Dan *et al* 1998).

Because the contract is a complex document and XML is not an intuitive language, an authoring tool is essential in preparing a contract. Once the contract is verified as valid and agreed to by all parties, it is passed to the contract 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 contract 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 contract and the registration database to convert a collection of templates into the executable file, as is discussed later.

5.1 Authoring Tool

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

The authoring tool starts with a DTD, which provides the syntactic structure of the contract. Then it constructs a model of a general contract by asking the model maker to provide examples (semantics) of all parts of the contract. Once a model is complete, it is available to any author who, by answering a few specific questions, can create a very complex contract with a high probability of success. Figure 2 illustrates the process of creating a model.

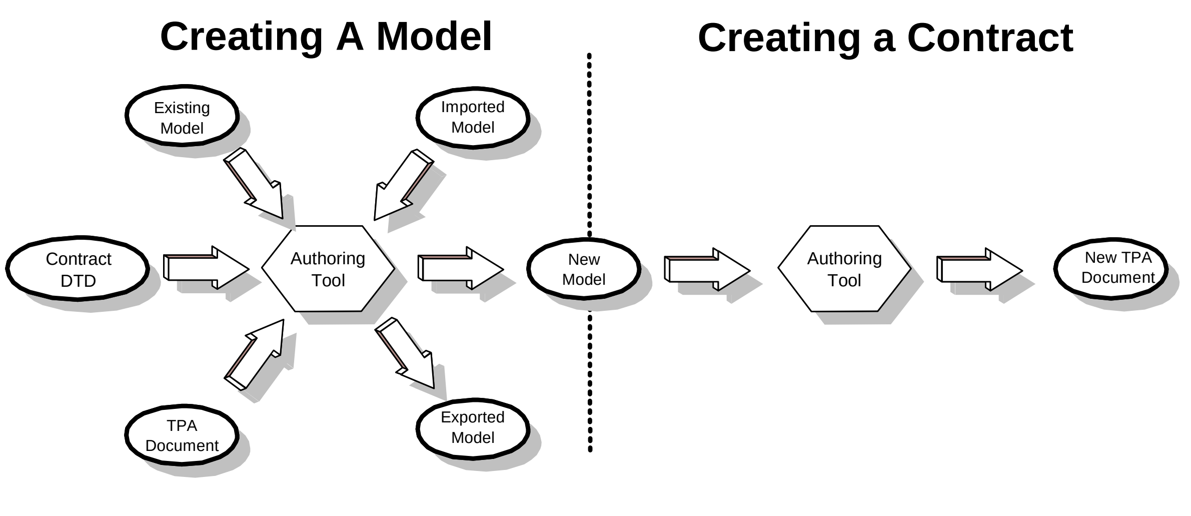

Figure 2: Creating a Model

A model consists of a collection of models of the tags to be used in the contract. 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. In the contract model, this tag may be represented by several models, one for each protocol which might be used in the business-to-business exchange. In building the model, the model maker brings in all of the subtrees which will be required in any final contract. Then the contract author merely chooses the needed subtree, and from there, makes minor refinements to that subtree.

In addition to examples of the tag's subtree structure, a model consists of information such as the number of times the tag is required, constraint information such as "if this model is included, then this other model must also be included", and details for a leaf tag such as data type, maximum and minimum length or value, units of numeric fields, etc., and data value, if any.

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

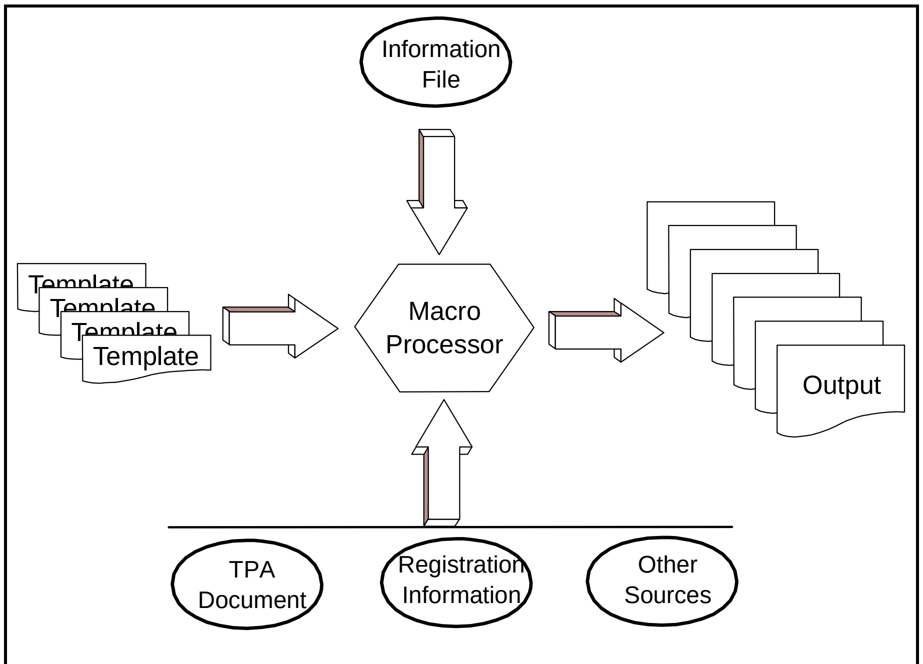

Figure 3: Code generation

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 contract and other external sources, declaration statements, assignment statements, and conditional statements which 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 macro processor also makes use of an information file which provides information such as the root of the local directory tree, additional data types and the Java classes which implement them, additional functions and the Java classes in which they are found as static members, and additional templates, grouped together in named collections.

6. Application Examples

This section describes two examples of the contract and server structure. The first is a 3-party example. The second shows the application to an existing public protocol.

6.1 3-Party Example

Figure 4 illustrates a 3-party application. The contract is among a travel agent (the client at the upper left) an airline company, and a hotel company. Contract objects are generated from the contract at the airline company, hotel company, and (not shown) the travel agent. At each of the parties is also shown the local application code. In this example, the hotel desk clerk also functions via the hotel contract object, though the desk clerk is not included in the contract and could equally well interact directly with the hotel application code.

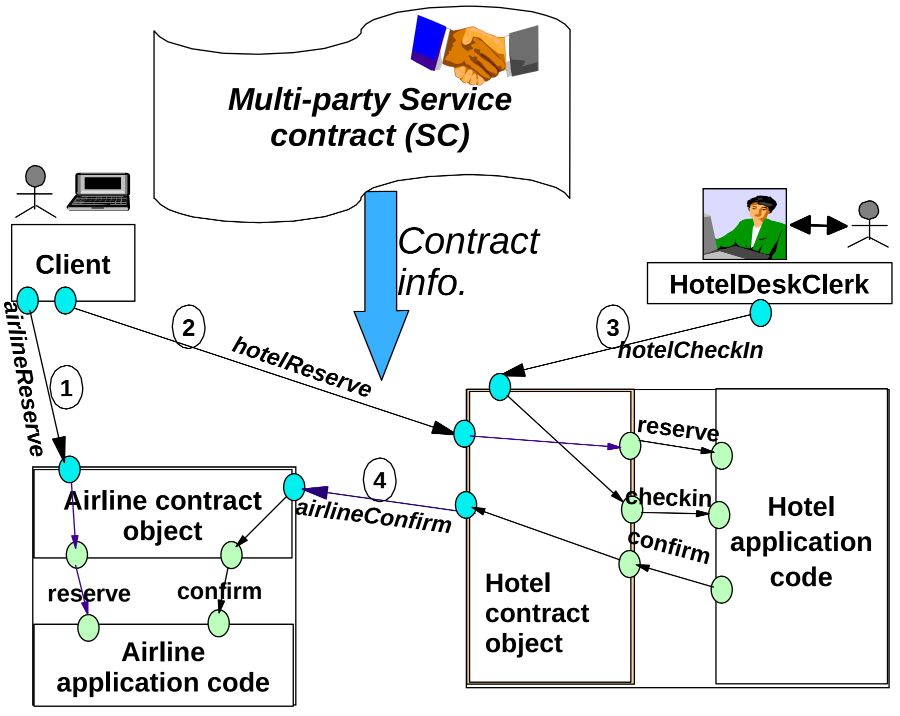

Figure 4: 3-Party Application

The contract is among three particular companies, Apex Travel, Super Airlines, and Sleepy Hotel. In this example, if a traveler flies on Super and checks in at Sleepy the same day, the traveler will receive an extra discount in addition to whatever normal discounts are available from Super and Sleepy.

The conversation begins when Apex sends reservation requests, airlineReserve (1) and hotelReserve (2), to Super and Sleepy respectively. Acceptance of the requests is indicated by callbacks from Super and Sleepy to Apex. Sometime later, the traveler flies to the destination and checks in at Sleepy. The desk clerk software issues the hotelCheckin (3) action request. Each of the aforementioned action requests is received by the airline or hotel contract object and transformed into the corresponding internal message to the application code as shown by the arrows in the figure.

When the traveler checks out of Sleepy, Sleepy issues the airlineConfirm (4) action request to Super. Super verifies that the traveler flew on Super the same day that the traveler checked in to Sleepy. When Sleepy receives the affirmative callback, it applies the combined discount to the traveler's bill. The traveler pays and returns home. At some time later, Super and Sleepy settle their respective shares of that discount. In this illustration, the settlement is assumed to be via the local back end processes at Super and Sleepy (not shown here) but settlement could also be by means of additional contract-defined action requests and responses. At this point, the conversation ends and the parties may choose to move the information logged during the conversation into their back end databases.

6.2 Trading Partner Agreement for 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 contract, and a schematic view of a possible implementation. Figure 5 illustrates the participants in an OBI transaction and the basic information flows.

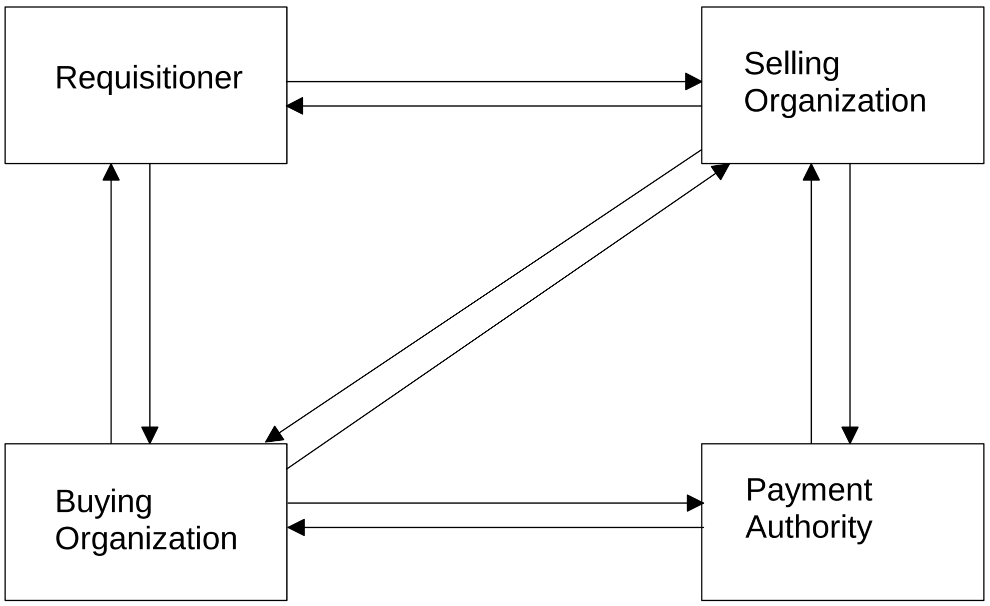

Figure 5: OBI Participants and Flows

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 contract between the buying organization and the selling organization, each of which has a business to business server. In OBI terms, the contract 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 contracts 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 HTTPS, 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 Contract 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 6 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 contract (in particular, the action requests) and the application logic at each server.

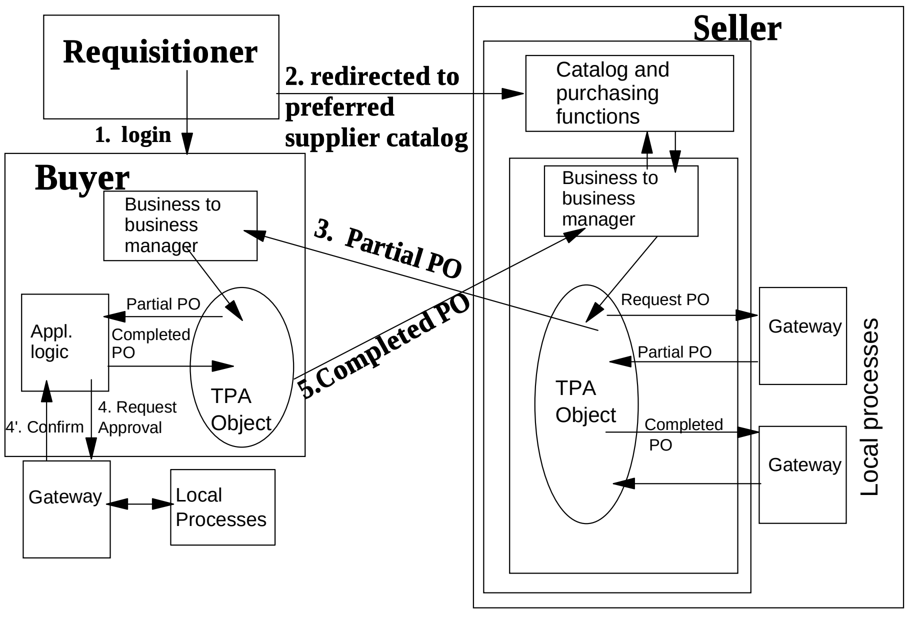

Figure 6: OBI Implementation

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 contract 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. Summary

This paper has discussed various issues in inter-business electronic interactions and in the use of an electronic contract for embodying the IT-related and business protocol terms and conditions used in business to business electronic commerce. We have designed an XML-based contract language and tools for authoring contracts in that language and generating code from the contracts. We described examples of two applications which make use of contracts and showed schematic views of such systems. We are extending the contract ideas and language to areas such as contract hierarchy, linking of multiple contracts, and dynamic negotiation.

Acknowledgments

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

References

Ajisaka, T., Electronic commerce for software, *Proc. Research and Advanced Technology for Digital Libraries, Second European Conference,* ECDL'98, Heraklion, Greece, Sept. 1998, Springer Verlag, Berlin, Germany, 1998, p. 791-800.

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.

Konana, P., Gupta, A., and Whinston, A., Digital contract approach for consistent and predictable multimedia information delivery in electronic commerce, *Multimedia Computer and Networking 1997,* San Jose, CA, USA, Feb. 1997, *Proc. SPIE - Int. Soc. Opt. Eng.* Vol. 3020, 1997, pp. 275-281.

Leymann, F. and Dieter Roller, Workflow-based applications. *IBM Systems Journal* 36, no. 1., 1997, p. 102-123.

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

Sandholm, T. and Lesser, V., Issues in automated negotiation and electronic commerce: extending the contract net framework, *Proc. First International Conference on Multi Agent Systems,* ICMAS 95, San Francisco, CA, USA, June, 1995, AAAI Press, Menlo Park, CA, USA, 1995, p. 328-335.

Sandholm, T., Unenforced e-commerce transactions, *IEEE Internet Comput.* (USA), Vol, 1, No. 6, Nov.-Dec. 1997, p. 47-54.

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.