The Role of B2B Engines in B2B Integration Architectures

Christoph Bussler
Oracle Corporation
500 Oracle Parkway, Redwood Shores, CA 94065, USA
Chris.Bussler@Oracle.com

ABSTRACT

Semantic B2B Integration architectures must enable enterprises to communicate standards-based B2B events like purchase orders with any potential trading partner. This requires not only back end application integration capabilities to integrate with e.g. enterprise resource planning (ERP) systems as the company-internal source and destination of B2B events, but also a capability to implement every necessary B2B protocol like Electronic Data Interchange (EDI), RosettaNet as well as more generic capabilities like web services (WS). This paper shows the placement and functionality of B2B engines in semantic B2B integration architectures that implement a generic framework for modeling and executing any B2B protocol. A detailed discussion shows how a B2B engine can provide the necessary abstractions to implement any standard-based B2B protocol or any trading partner specific specialization.

1. INTRODUCTION

Exchanging B2B events with trading partners requires the following functionality:

- Definition of B2B events (e. g. purchase order).
- Definition of the protocol of the B2B events as public processes visible to trading partners (for example, a purchase order has to be followed by a purchase order acknowledgment).
- Definition of the security to be used (e. g. encryption, signatures as well as non-repudiation).
- Storing all sent and received B2B events in clear text as well as in encrypted and signed form for audit and non-repudiation.
- Network connectivity to connect to various networks (e. g. Internet [9], VANs [15] or FIN [8]).
- Trading partner management to define legal trading partners of a business, their unique identification and other relevant information like address and contact information.
- Trading partner agreements to define the rules of engagement between trading partners (e. g. the definition that two trading partners exchange purchase orders over the RosettaNet protocol [12]).

A B2B integration architecture is responsible for the complete processing of B2B events. It not only has to provide the trading partner connectivity, but also the back end system connectivity as well as process management in order to management the B2B event migration from trading partners to back end applications and vice versa. An active sub-component, the B2B engine, is responsible for the communication aspects of B2B events with trading partners. It implements the above listed functionality and makes it available to the overall B2B integration architecture.

Section 2 introduces a B2B integration example. Section 3 introduces a model for B2B protocols and discusses several examples. Section 4 introduces a B2B engine as an architectural component that implements the B2B protocol model and provides B2B integration functionality to an B2B integration architecture. The complete architecture is introduced in Section 5.

The contribution of this paper is to focus and to describe the detailed internal architecture of a B2B engine. [2] and [3] assume that a B2B engine is available as a fundamental component, however, without describing its internal structure at all.

2. B2B INTEGRATION EXAMPLE

Figure 1 shows the external B2B event exchange between two trading partners (TPs). TP 1 sends purchase orders (POs) according to the B2B protocol EDI [7] to TP 2. TP 2 responds with one EDI purchase order agreement (POA) for each PO received.

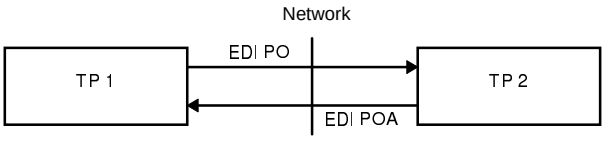

Figure 1: PO-POA round-trip B2B event exchange between two trading partners

The trading partner internal processing of the two B2B events is not shown.

Figure 2 shows a complete model of the integration internal to TP 2. This integration does not only shows that TP 2 supports public processes to receive and send EDI POs and POAs, but also those of RosettaNet (RN). In addition, the private process for the B2B event processing is shown. It includes the execution of a business rule requiring approval if the PO amount exceeds a specific value (70000). Furthermore, the integration with two back end applications (Oracle [11] and SAP [13]) is depicted. The binding makes sure that the public processes (as well as the application processes) can be bound to the private process. This example is in detail discussed in [3].

This detailed example shows that a trading partner can have several public processes that need to be implemented for trading partner communication with different trading partners. The B2B engine of TP 2's integration architecture must support two B2B protocols and implement two public processes, one for EDI and one for RosettaNet.

The example in Figure 2 only shows the model of the integration, not the additional necessary trading partner specific technical configurations of the B2B protocols involved like security or transport packaging. These additional configurations are required per trading partner (like which security to implement or which transport to use). This is discussed in detail in Section 4.

3. B2B PROTOCOL MODEL

3.1 MODEL DEFINITION

According to [2] a B2B protocol consists of the following elements:

- **Document types**. Document types define the structure of B2B events. For example, the definition of the contents of a purchase order is a document type. At runtime, a B2B event can be created following this document type definition. Document types can be defined using XML or non-XML syntax (as in the case of EDI).
- **Document semantics**. The document semantics defines the legal values of document element like the set of all possible city names in an address. Furthermore, consistency rules define the combination of elements that have to be populated.
- **Public process definition**. The public process definition states which B2B events can be received and send as well as in which order (see Figure 2: after the receipt of a PO a POA has to be sent back). More sophisticated definitions include retry-semantics and time-outs to achieve reliable communication of B2B events.
- **Exchange sequence**. Exchange sequences define if an acknowledgment has to be returned for a given B2B event. This acknowledgment is not a business level acknowledgment but a transport level acknowledgment telling that the message was received.
- **Packaging**. The packaging defines how the B2B event is packaged with header information specific to B2B protocols (e.g. containing the sender identification) as well as further elements like attachments.
- **Transport binding**. The transport binding defines how a packaged B2B event is extended with transport specific headers and trailers.

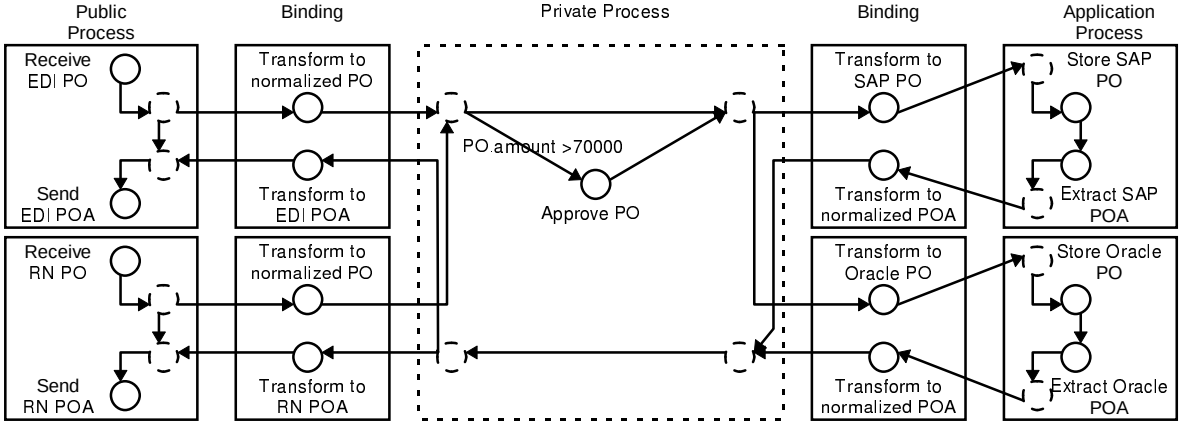

Figure 2: TP 2's internal integration model for the PO-POA round-trip B2B event exchange

- **Security**. One form of security is the signing of a B2B event to prove that it was not altered. Encryption can be used to ensure that no other entity can read the B2B event enroute from the sending to the receiving trading partner.

Figure 3 shows these elements as different layers whereby security and document types are elements available to all layers (e.g. encryption can happen on the B2B event level as well as the transport level).

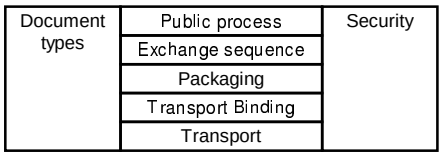

Figure 3: B2B protocol layers

According to this model a data base schema (B2B engine schema) is designed that the B2B engine uses during execution to retrieve B2B protocol definitions. For each supported B2B protocol the above elements have to be modeled in the B2B engine schema. B2B protocols are modeled independent of trading partners since they can be reused by several of them.

Additional configuration data is necessary for each trading partner to allow B2B event exchange:

- **Non-repudiation**. Several forms of non-repudiation exists [1] that have to be enforced. The ultimate goal of non-repudiation is that neither trading partner can deny the sending or receiving of B2B events and their specific content.
- **Endpoint management**. Endpoints are the network addresses that allow a trading partner to deliver a B2B event. E. g. a HTTP address might be the entry point for delivering PO B2B events.
- **Key management**. For encryption, decryption as well as creation and verification of signatures trading partners' keys have to be stored.
- **B2B protocol parameters**. Different trading partners might require different time-out or retry parameters. If a network is slow, time-outs are set to higher values. If a network often fails, the retry-count might have to be higher.
- **Audit**. Audit ensures that at any point in time it is possible to find out where B2B events are and in what processing state. An audit contains all state changes of a B2B event as well as each state value at each given point in time.

These configuration data are trading partner specific. For each trading partner this configuration data has to be stored in the B2B engine schema. Only then B2B event exchange with trading partners is possible.

Finally, trading partner agreements have to be modeled between trading partners that define which B2B protocol is used for exchanging B2B events, which public processes are executed and which trading partner is initiating an exchange. E. g. in Figure 1 TP 1 and TP 2 both have a trading partner agreement that defines that they use EDI as the protocol, that they execute the EDI public process for PO-POA exchange as shown in Figure 2 and that TP 1 is the initiating trading partner.

3.2 MODEL SCOPE

The model described in Section 3.1 is defined in such a way that any B2B protocol like ebXML [6], EDI [7] or RosettaNet [12] can be represented independent of the specifics. E. g. XML cannot be always assumed since EDI is non-XML based. Trading partner agreements cannot be assumed since RosettaNet does not have this concept. The more complete a B2B protocol is defined, the more similarity it has with the given model. ebXML falls into this category since ebXML's goal is to be a complete XML standard. The concepts of any B2B protocol have to be mapped into the defined model so that the B2B protocol can be executed by the B2B engine. Therefore the defined model is a meta-model since it can capture the specific models and concepts of specific B2B protocols.

3.3 EXAMPLES

In the following some examples are introduced that are supported by the above model. Exhaustive lists of B2B protocols can be found in [2] and [5].

3.3.1 ROSETTANET

An example that nicely illustrates the B2B protocol model above is RosettaNet. RosettaNet is a standard that defines all the model elements related to B2B protocols. RN provides document types as well as document semantics in form of technical dictionaries. Public process definitions are so-called Partner Interface Processes (PIPs) that define the exchange sequence of particular B2B events. The exchange sequence is defined by requiring an acknowledgment event for each B2B event sent in order to confirm the receipt. RosettaNet B2B events are mime packaged and a transport binding for http as well as smtp are defined.

3.3.2 WEB SERVICES

Web services are not a complete B2B protocol in contrast to RosettaNet. Web services define only the "lower" layers of B2B protocols like packaging and

transport binding. This is done through the SOAP [14] as well as WSDL [16] standard. Document types and document semantics, public processes, exchange sequences are not defined. This functionality must be provided otherwise [4]. For example, several proposals are available that propose definition languages for defining public processes [17] [18].

3.3.3 EBXML CPP AND CPA

ebXML [6] is a collection of several different standards. One of them deals with trading partner property ("profile") definition (CPP, collaboration partner profile) as well as trading partner agreement definition (CPA, collaboration partner agreement). The model of a generic B2B must be able to model those, too.

4. B2B ENGINE

4.1 ARCHITECTURE

The B2B engine is part of an B2B integration architecture (see Section 5). It provides the functionality required to execute any B2B protocol and to manage the actual B2B event exchange over networks. Figure 4 shows the layered components of the B2B engine.

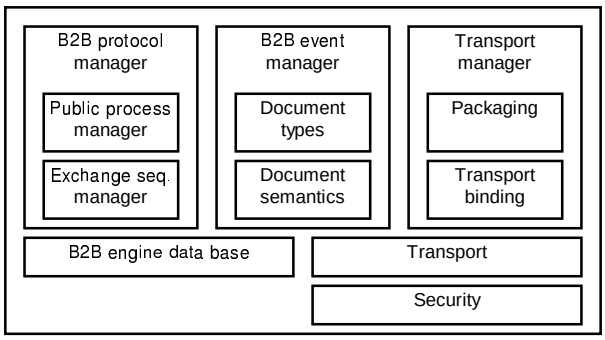

Figure 4: B2B engine architecture

- **B2B engine data base**. The data base contains all data of the model as described in Section 3. All components retrieve and store data into this data base schema. Data required to store state of instances like the state of a public protocol are stored in the data base, too.
- **Security**. The security component provides the functionality for encryption, decryption, signing and signature verification and non-repudiation.
- **Transport**. Transport provides the connectivity to the networks like the Internet, VANs or FIN.
- **B2B protocol manager**. The B2B protocol manager is a component that is solely responsible for executing the B2B protocols as defined in the B2B engine schema. It has several components in itself that together ensure correct public process execution.
  - **Public process manager**. The public process manager is responsible for the execution of public processes as defined by the trading partner agreement.
  - **Exchange sequence manager**. The exchange sequence manager processes transport acknowledgments in case they are required by a trading partner.
- **B2B event manager**. The B2B event manager is responsible for the B2B event processing.
  - **Document types**. This component can instantiate B2B events according to defined document types as well as verify that B2B events are of a known type.
  - **Document semantics**. This component verifies that B2B events to be send or that have been received are semantically correct as per the validation specifications.
- **Transport manager**. The transport manger is responsible for sending and receiving events as well as providing the correct packaging and transport binding.
  - **Packaging**. This component packages B2B events according to the packaging directive of the B2B protocol for which it has to be packaged. A popular mechanism is mime or s/mime.
  - **Transport binding**. This component binds packaged B2B events to the transport to be used according to the trading partner agreement.

An interesting observation is that ebXML mentions a subset of the above listed architectural components. However, ebXML has a specific set of model concepts. Since the introduced architectural components have to be able to execute any model of any B2B protocol (and not just ebXML), they have to be more general than those introduced by ebXML. ebXML is therefor seen as a proof point that the above list of architectural components can support any B2B protocol.

4.2 EXTERNAL INTERFACES

There are two external interfaces of the B2B engine. One is the transport component through which events are sent to trading partners as well as received from trading partners. The other external interfaces is the B2B event manager. External components that want

to send or receive events interface with the B2B event manager to obtain or to create events. The private process manager in Section 5 is interfacing this way.

4.3 EXECUTION

In this section the detail event flow within the B2B engine is described for outbound and inbound events.

4.3.1 OUTBOUND B2B EVENT

If a B2B event has to be sent to a trading partner (outbound) the B2B engine processes the event as follows. First, the event has to be created in the B2B event manager. The result is a consistent event according to the document semantics as enforced by the document semantics component. Then, the appropriate public process is executed for the trading partner the B2B event has to be sent to. As part of this public process execution the event (like a PO) has to be sent to the appropriate trading partner. This means to package and to bind the event to the appropriate transport. If the event has to be encrypted the security component is invoked to encrypt the event. The event is then finally sent to the trading partner's endpoint through the transport component. If the transport connection is encrypted, the security component is used to retrieve the encryption key. As a result, an acknowledgment might come back that is processed by the exchange sequence manager as part of the B2B protocol execution. For example, the sending of the EDI PO from TP 1 to TP 2 in Figure 1 would follow the above execution sequence.

4.3.2 INBOUND B2B EVENT

If a B2B event is received by the transport component in the inbound case, the outbound processing is reversed. First, the event is decrypted. This ensures that it is coming from the trading partner the event claims to come from. If it cannot be decrypted then the B2B event is discarded. Once a B2B event is decrypted, it has to be de-bound and de-packaged in order to get the clear text event. The clear text event is checked for consistency violation by the B2B event manager using the document semantics component. Once the clear text event is regarded as correct, an existing public process is searched in order to continue the event processing. If no one can be found, a new public process is instantiated for this event. This explains that each event is either a continuing event (like the POA in Figure 1) or an initiating event (like the PO in Figure 1). Once the public process is found, the event processing is achieved in the B2B engine. In case an acknowledgment has to be sent the exchange sequence manager will send one back to the trading partner as an outgoing B2B event and the outbound B2B event processing takes place.

5. B2B INTEGRATION ARCHITECUTRE

5.1 STRUCTURE

Figure 5 shows the complete B2B integration architecture. Not all components are shown in all their details.

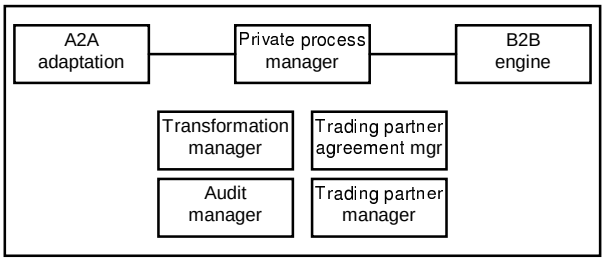

Figure 5: B2B Integration Architecture

The components are

- **B2B engine**. The B2B engine is part of the B2B integration architecture responsible for the communication of B2B events with trading partners.
- **A2A adaptation**. This component is responsible for the connectivity with back end applications like ERP systems as discussed before. It's functionality is to connect to the various interfaces of various back end systems (e.g. through J2EE Connector Architecture [10]). In addition, it executes the application processes as shown in Figure 2. This component is not detailed out and shown as one component in Figure 5 in analogy to the B2B engine.
- **Private process manager**. The private process manager is responsible for executing the private processes and the bindings to public processes as well as application processes (see Figure 2). It orchestrates the architecture-internal communication with the A2A adapter and B2B engine.
- **Trading partner manager**. The trading partner manager implements the part of the model in Section 2 that stores trading partner specific data. This is not part of the B2B engine since trading partner information is relevant for private processes, too. For example, when an event comes in from a specific trading partner a specific private process might have to be executed.
- **Trading partner agreement manager**. Correspondingly, the part of the model in Section 2 that

deals with trading partner agreement data is managed by this component. Like the trading partner manager it is not part of the B2B engine since other components have to access it, too, to determine processing information.

- **Audit manager**. The audit manager is available to all components in the B2B architecture. This makes sure that all components write audit information consistently to a central place. Furthermore, all audit information can be retrieved from one consistent schema.
- **Transformation manager**. The transformation manager is responsible for event transformation. This is relevant in the bindings (see Figure 2). When events are sent by back end systems and have to be sent on to trading partners, the event format has to be transformed in almost all cases. For example, a PO coming from the SAP ERP must be transformed into the representation as defined by EDI. Transformation technology enables this and the transformation manager is responsible for the execution of transformations.

5.2 EXECUTION

This section describes the event flow within the overall B2B integration architecture.

5.2.1 OUTBOUND EVENTS

Once an event is ready to be sent from a back end application to a trading partner the A2A adaptation component detects it. It retrieves the event from the application and passes it on to the private process manager. The private process manager executes the appropriate binding for the event. That might involve transformations. After this the appropriate private process is executed. It determines both, the binding and the private process by querying the trading partner component. Once the private process is finished, the private process manager retrieves the correct binding for the public process by querying the trading partner agreement component. After the binding is executed the event is passed to the B2B engine for further processing. Audit data is written at various processing stages.

5.2.2 INBOUND EVENTS

Inbound events are processed exactly the opposite way starting at the B2B engine and finishing at the A2A adaptation.

5.2.3 MANY-TO-MANY PROCESSING

Even though not mentioned so far, it is possible to send one event to several trading partners and to send a trading partner event to several back end applications. Also, a trading partner event could be re-routed to another trading partner not touching a back end system at all.

6. REFERENCES

[1] Blixt, K.-F.; Hagstroem, A.: Adding Non-Repudiation to Web Transactions. www.it.isy.liu.se/~asa/publications/NonRepudiation.html

[2] Bussler, C.: B2B Protocol Standards and their Role in Semantic B2B Integration Engines. In*: Bulletin of the Technical Committee on Data Engineering*. March 2001, Vol. 24, No. 1. IEEE Computer Society

[3] Bussler, C.: Modeling and Executing Semantic B2B Integration", In: *Proceedings of the 12th International Workshop on Research Issues on Data Engineering: Engineering E-Commerce / E-Business Systems (RIDE-2EC'2002) (In conjunction with ICDE-2002*), IEEE CS Press, San Jose, USA, February 24-25, 2002

[4] Casati, F.; Shan, M.-C.: Dynamic and adaptive composition of e-services. In: *Journal of Information Systems* 26, 2001

[5] Cover, R.: The XML Cover pages. www.oasis-open.org/cover/

[6] ebXML. www.ebxml.org/

[7] EDI. www.x12.org

[8] FIN services. www.swift.com/index.cfm?item_id=3184

[9] Internet. webopedia.internet.com/TERM/I/Internet.html

[10] J2EE Connector Architecture. java.sun.com/j2ee/connector/

[11] Oracle E-Business Suite. www.oracle.com/applications/index.html?content.html

[12] RosettaNet. www.rosettanet.org

[13] SAP. www.sap.com/

[14] SOAP. www.w3.org/TR/soap12-part1/ and www.w3.org/ TR/soap12-part2/

[15] VAN. www.edi-info-center.com/html/vans.html

[16] WSDL. Web Services Description Language. Version 1.1. Ariba, IBM, Microsoft. January 2001. msdn.microsoft.com/ xml/general/wsdl.asp

[17] WSFL. Web Services Flow Language. Version 1.0. IBM Software Group, May 2001. www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

[18] XLANG. www.gotdotnet.com/team/xml_wsspecs/