A Business-Aware Web Services Transaction Model

Mike P. Papazoglou and Benedikt Kratz

Tilburg University, Infolab
P.O. Box 90153, 5000 LE Tilburg, The Netherlands
{mikep, B.Kratz}@uvt.nl

Abstract. Advanced business applications typically involve well-defined standard business functions such as payment processing, shipping and tracking, managing market risk and so on, which apply to a variety of application scenarios. Although such business functions drive transactional applications between trading partners they are completely external to current Web services transaction mechanisms as they are only expressed as part of application logic. To remedy this situation, this paper proposes a business aware Web services transaction model and support mechanisms. The model allows expressing and blending business and QoS aware transactions on the basis of business agreements stipulated in SLAs and business functions.

1 Introduction

As enterprises follow the path to e-business, business processes are becoming increasingly complex and integrated both within internal corporate business functions (e.g., manufacturing, design engineering, sales and marketing, and enterprise services) and across the external supply chain. In this environment there is a clear need for advanced business applications to coordinate multiple Web services into a multi-step business transaction. This requires that several Web service operations or processes attain transactional properties reflecting business semantics, which are to be treated as a single logical (atomic) unit of work. For example, consider, a manufacturer that develops Web service based solutions to automate the order and delivery business functions with its suppliers as part of a business transaction. The transaction between the manufacturer and its suppliers may only be considered as successful once all products are delivered to their final destination, which could be days or even weeks after the placement of the order, and payment has ensued.

In contrast to Web service transactions, which are driven by purely technical requirements such as coordination, data consistency, recovery, and so on, business transactions are driven by economic needs and their objective is accomplished only when the agreed upon conclusion among trading parties is reached, e.g., payment in exchange for goods or services.

This approach requires distilling from the structure of a business collaboration the key capabilities that must necessarily be present in a business transaction

A. Dan and W. Lamersdorf (Eds.): ICSOC 2006, LNCS 4294, pp. 352–364, 2006.

and specifying them accurately and independently of any specific implementation mechanisms. The business transaction then becomes the framework for expressing detailed operational business semantics.

Conventional approaches to business transactions, such as Open EDI (http://www.iso.org) and more recently ebXML, focus only on the documents exchanged between partners, and ignore important constituents in a business transaction such as business operations and their behavioral semantics. A more natural approach to business transactions is to make common business operational requirements and operational level relationships between trading partners first class tenets in a business transaction. This requires providing a common core of well-understood business operational principles (or business transaction functions) such as ordering, transport, distribution and payment that can be rationalized and appropriately combined across any supply chain to create semantically enhanced business transactions. Developers can then build transactional applications by using, combining, and, possibly specializing, these constructs in a similar way that abstract data types are used in programming languages.

This paper focuses on introducing an advanced business transaction model and on providing operational business principles for specifying and modeling business transactions along with their QoS characteristics. The approach taken mimics business operation semantics and does not depend upon underlying technical protocols and implementations. The paper also presents a Business Transaction Model Language (BTML) that is used at design time to specify the elements of a business transaction. Run-time support for this environment is provided by conventional Web services standards such as BPEL, WS-Coordination and WS-Transaction and will be briefly highlighted in the context of a reference architecture. Detailed descriptions of the run-time environment as well as run-time transformations between the BTML and equivalent constructs supported by Web services transaction standards are outside the scope of this paper.

2 The Business Transaction Model

An important requirement in making cross-enterprise business process automation happen is the ability to describe the collaboration aspects of the business processes, such as business commitments, mutual obligations and exchange of monetary resources, in a standard form that can be consumed by tools for business process implementation and monitoring. This gives raise to the concept of a *business transaction model* that encompasses a set of business transaction functions and several standard business primitives and conventions that can be utilized to develop complex business applications involving transactional and monetary exchanges.

Central to the business transaction model is the notion of a business transaction. Business transactions cover many domains of activity that businesses engage in, such as request for quote, supply chain execution, purchasing, manufacturing, and so on. A *business transaction* is defined as a trading interaction between possibly multiple parties that strives to accomplish an explicitly shared

business objective, which extends over a possibly long period of time and which is terminated successfully only upon recognition of the agreed conclusions between the interacting parties. A business transaction is driven by well-defined business tasks and events that directly or indirectly contribute to generating economic value, such as processing and paying an insurance claim. If a business transaction completes successfully then each participant will have made consistent state changes, which, in aggregate, reflect the desired outcome of the multi-party business interaction. The purpose of a business transaction is to facilitate specifying common business procedures and practices in the form of business application scenarios that allow expressing business operational semantics and associated message exchanges as well as the rules that govern business transactions. Such rules include operational business conventions, agreements, and mutual obligations. The combination of all these factors characterizes the nature of business relationships among the parties involved in a business transaction. It enforces trading parties to achieve a common semantic understanding of the business transaction and the implications of all messages exchanged.

The business transaction is initiated by a single organization and brings about a consistent change in the state of a business relationship between two or more trading parties. A *business relationship* is any distributed state held by the parties, which is subject to contractual constraints agreed by those parties. A business transaction needs to express features like the parties that are involved in the transaction; the entities under transaction; the destination of payment and delivery; the transaction time frame; permissible operations; links to other transactions; receipts and acknowledgments; and finally, the identification of money transferred outside national boundaries.

The previous definition of a business transaction has been derived from (classical) commerce models and serves as a common high-level, non-technical view of how business organizations interact with each other. The definition emphasizes the operational business view of a transaction. There are four key components in a business transaction model that help differentiate it from (general) message exchange that business processes involve. These are: (1) commitment exchange; (2) the party (or parties) that has the ability to make commitments; (3) business constraints and invariants that apply to the message exchanged between the interacting parties; and (4) business objects (documents) that are operated upon by business activities (transactional operations) or by processes. These terms are introduced and explained below, while Fig. 1 represents them and their inter-relationships in UML.

A *commitment exchange* occurs between two or more interacting *parties* and concerns tasks or functions to be carried out and usually involves formal trading partner agreements. A commitment exchange identifies such things as the overall business process, the partner roles, the business documents used, message and document flow, legal aspects, security aspects, business level acknowledgments and status, and so on. Partners inside a transaction have distinct roles (such as *buyer* and *seller*) and the ability to make commitments, being held responsible for, having rights and obligations, in the context of the business transactions.

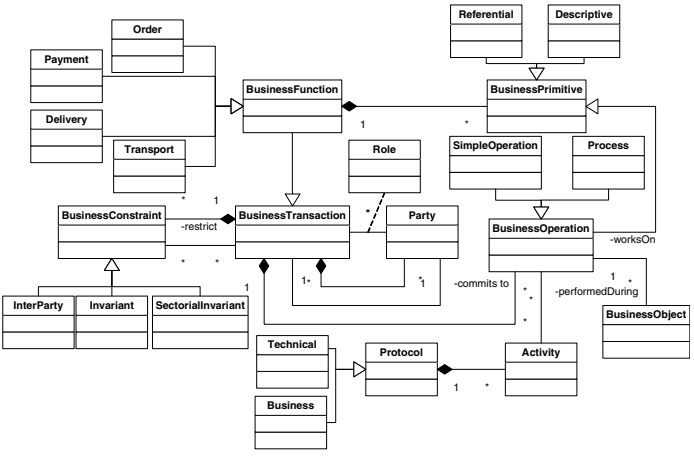

Fig. 1. UML view of a business transaction

One party can act as the initiator of the transaction while the others can act as responders.

A *business transaction constraint* is defined as an explicitly stated rule that prescribes, limits, or specifies any aspect of a business transaction that forms part of the commitment(s) mutually agreed to among the interacting parties. *Business invariants* are constraints external to constraints agreed by interacting parties in a transaction and include universal legal requirements, commercial and/or international trade and contract terms, public policy (e.g., privacy/data protection, product or service labeling, consumer protection), laws and regulations that are applicable to parts of a transaction. Invariants ensure the nature of the business transaction and/or the goods or services delivered while guaranteeing that no regulations are compromised. Business invariants are universal (or horizontal) in nature and apply regardless of the type of business or sector within which the business occurs. There are, however, constraints external to parties that are of a sectorial nature called *Sectorial invariants* which can be found in sectors such as telecommunications, transportation and delivery, financial/banking, and so on. Universal and sectorial invariants can be combined with inter-party business constraints for building application use scenarios. It is important to understand that in such situations invariants take precedence over internal constraints in a business transaction.

Business transactions may be characterized by universally acceptable business operational primitives (or simply business functions), which represent functions that are critical to the conduct of business. A *business function* is a description of a well-defined and commonly acceptable critical business principle, e.g., payment or delivery of goods or services, that transforms business values and causes state changes to transaction participants, e.g., transforms an unpaid order to paid order. To achieve this the business function uses contextually aware polymorphic business operations, e.g., cancel an order or cancel a payment, constraints

and dependencies, (see Sect. 4 for further details). Business transactions usually operate on business (document-based) objects. These are traditionally associated with items such as purchase orders, catalogues (documents that describe products and service content to purchasing organizations), inventory reports, ship notices, bids and proposals. Document objects are usually associated with agreements, contracts or bids.

In a Web services environment business transactions are used to capture and define the integration between business operational requirements and technical transactional requirements. Business transactions are found only in the application (business-logic) level and essentially trigger transactional Web service interactions between organizations at the systems-level (using Web services-based business processes and transactional standards) in order to accomplish some well-defined shared business objective. A business transaction in its simplest form could represent an order of some goods from some company. The completion of an order results in a consistent change in the state of the affected business: the back-end order database is updated and a document copy of the purchase order is filed. More complex business transactions may involve activities such as payment processing, shipping and tracking, determining new product offerings, granting/extending credit, and so on.

At run-time the business transaction model requires support from systems-level transactional frameworks provided by Web Services standards that include the Web Services Coordination and Transaction [1,2,3] and the Web Services Composite Application Framework (WS-CAF) [4]. Objective of systems-level transactional support is to automate the internal flow of transaction processing, spanning multiple disparate applications provide solutions for reliable, consistent, and recoverable composition of back-end services. Important systems-related aspects of a business transaction include features like the ability to support long-running interactions; to specify exceptional conditions; to support compensatible and contingency transactions; to make use of alternate transactions; to reconcile and link transactions with other transactions; to support secure transactions and to allow transactions to be monitored, audited/logged and recovered.

3 Integrated Logistics Example

In this section we present a simple integrated logistics example based on standard business protocol RosettaNet PIPs [5], which we shall enhance in subsequent sections with transactional functions and business operational semantics as well as QoS features.

Fig. 2 depicts an integrated logistics scenario involving a customer, suppliers and a logistics service provider. This logistics model consists of forecast notification, forecast acceptance, inventory reporting, shipment receipt, request and fulfil demand, consumption and invoice notification processes provided by RosettaNet PIPs. In Fig. 2 PIP 4A2 supports a process in which a forecast owner sends forecast data to a forecast recipient. PIP 4A5 provides visibility of available

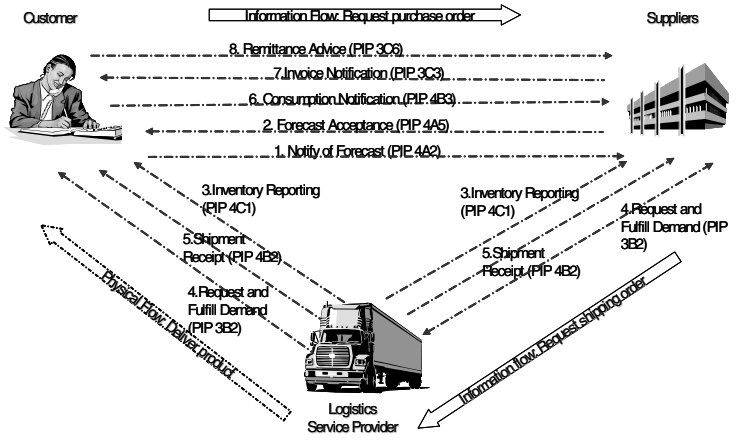

Fig. 2. Integrated logistics example using RosettaNet PIPs

forecasted product quantity between two trading partners. PIP 4C1 supports a process in which an inventory information provider reports the status of the inventory to an inventory user. The inventory report can include any product, active or inactive, held in inventory. PIP 3B2 allows a shipper to notify a receiver that a shipment has been assigned. This notification is often a part of the shipment process. PIP 4B2 supports a process used by a consignee to report the status of a received shipment to another interested party, such as a shipper. The customer then issues an invoice notification (PIP 4B3) to communicate material consumption to the supplier, allowing the supplier to trigger invoicing for the consumed material. PIP 3C3 enables a provider to invoice another party, such as a buyer, for goods or services performed. Finally, PIP 3C6 enables a payer to send remittance advice to a payee (in this case the supplier) which indicates which payables are scheduled for payment.

4 Operational Business Principles and QoS Functions

In the previous we argued for the identification of functional capabilities necessary to support business transactions and introduced the concept of critical business functions. Figure 3 describes the elements of the business functions in the business transaction model which is described schematically in Fig. 1. Note that Fig. 3 due to reasons of brevity depicts only constraints and not invariants. In particular, this figure illustrates that the business transaction model divides trade into four broad areas - ordering, paying, delivery and transportation, which are referred to as *operational business principles* (or business functions) in this figure. These areas represent common business functions that are generic, industry neutral and re-usable and can be used to develop business transactions in a multiplicity of business scenarios. In this way we remove excess complexity from the business transaction, allowing common business functions such as ordering,

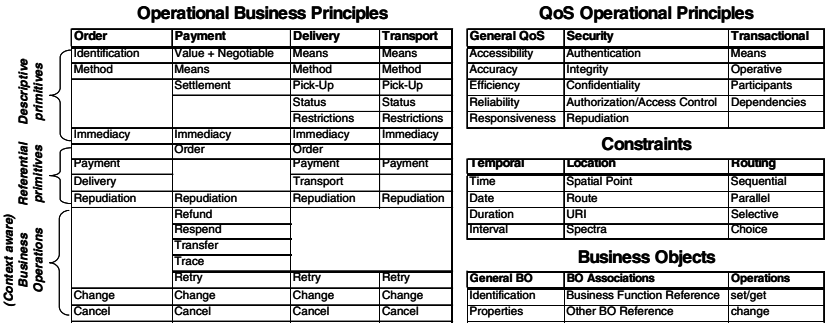

Fig. 3. Description of common business functions, QoS principles and constraints

distribution and payment to be expressed in a form analogous to abstract data types and rationalizing them across an integrated supply chain.

Figure 3 shows that each business function uses a number of *descriptive primitives* (or attributes) that describe a certain business function, e.g., the means of payment. There are also *referential primitives* that refer to other business functions, e.g., payment refers to an associated order. Finally, the *context aware business operations* introduce a set of polymorphic business operations that collectively transform business values and cause state changes to the business transaction participants.

Business functions not only help streamline and rationalize common business practices across an integrated supply chain, they also help enforce participant commitments. They introduce a mandatory set of four *business level atomicity criteria* that reflect the operational semantics the four standard business functions (ordering, payment, delivery and transportation). For instance, payment atomicity affects the transfer of funds from one party to another in the transaction. This means that the transaction would fail if payment is not made within a pre-specified time period that was agreed between a supplier and a customer. Delivery atomicity, on the other hand, implies that the right goods will be delivered to a customer at the time that has been agreed.

Each atomicity criterion is treated as a single indivisible logical unit of work, which determines a set of viable outcomes for a business transaction. The outcomes of a business transaction may involve non-critical partial failures, or selection among contending service offerings, rather than the strict *all-or-nothing* assumption of conventional ACID transactions, and govern the duration and character of participation in a transaction. They also allow provisional results to be revealed deliberately to allow such business activities as probabilistic inventory management. Atomicity criteria can be characterized as *vital* or *non-vital*. If a business level atomicity criterion is characterized as vital and fails then the transaction aborts at the system-level. If, however, the atomicity criterion is characterized as non-vital then a contingency activity may be issued in case that a given atomicity criterion, e.g., transportation, fails. For instance, using another shipper in case that the chosen one fails to deliver. The above characteristics give

```
Means <!- The means of the delivery (depends on nature of goods) -->
•setMeans
      •setChannel
            •Online <!- intangible goods --> / offline <!- tangible goods -->
      •setDeliveryMeans
            •Air / Sea / Ground / Combinations
Method <!- Method of the delivery -->
•setDeliveryMethod <!- How will goods be delivered (e.g., batch, all in once, etc) -->
      •setNumberOfDeliveries <!- How often will goods be delivered (if in batches) -->
•setDeliveryOptions
      •Express
            •Type
                  •Next day / Two day / ...
                  •Boolean Delivery_Signature_Required
•setTransportCompany <!- Which company is responsible for the transport -->
      •setTransportCompanyDetails
•setDeliveryPeriod
      •Temporal
```

Fig. 4. Describing the *delivery* business function attributes

the ability to a business transaction to explicitly describe business operational semantics, specify the proper behavior of common business functions and their implications in case of success or failure.

The business transaction model not only expresses the purpose of each business collaboration interaction but is also capable of capturing the timing and sequence of message exchanges. The model has fixed sequencing semantics which require that *ordering* occurs first and is followed by *transport* and *delivery*. *Payment* can happen before or after the *delivery* function. For instance, in the integrated logistics scenario described in the previous section, there might be an implicit or explicit agreement that the delivery of goods must take place before the payment and that payment always follows the confirmation of an order. This situation is depicted by the following code snippet that uses BTML:

```
<BTx>
 <name>LogisticsScenario</name> ...
 <sequence>
  <BF> <name>Order</name> </BF>
  <BF> <name>Delivery</name> </BF>
  <BF> <name>Payment</name> </BF> ...
 </sequence>
</BTx>
```

By using these constructs, each participant can understand and plan for conformance to the business protocol being employed.

Figure 4 shows two of the attributes of the *delivery* business function. These are *means* and *method* which describe the means and method of delivery, respectively. The *delivery* business function is seen from Fig. 3 to also use referential primitives to refer to such other functions as *payment*, and *transport*. It also uses context aware polymorphic business operations, such as *retry* to retry a failed delivery, *change* to change a delivery and *cancel* to cancel a delivery. All attributes and operations in Fig. 3 have been defined and formalized and are available on request.

Finally, an important element of the business model is the quality of service required from the functional capabilities for the business transactions. For

instance, one of the referential primitives used in business functions such as ordering, payment, transport and delivery, is the issue of non-repudiation using digital signatures, see Fig. 3. In this way business transactions can also become QoS-aware and QoS principles can be blended with constraints and business requirements enforced by the business functions. Other QoS primitives that can be attached to a business transaction and govern its behavior may include general QoS primitives such as desired performance rates, mean time to respond, accessibility periods, time-to-repair a service that has failed, desirable security protocols and tokens, and so on. These are also depicted in Fig. 3. QoS criteria can be registered in a Service Level Agreement, which specifies the agreements and commitments of trading partners involved in a business transaction. More specifically, they form part of the agreed service-level objectives, which define the levels of service that both the service customers and the service providers agree on, and usually include a set of service level indicators, like availability, performance and reliability.

5 Business Transaction Reference Architecture

The reference architecture that supports the business transaction model is depicted in Fig. 5. Application scenarios are specified by using the business related aspects of the model, e.g., business principles, constraints, QoS criteria, and so forth. Both the business aspects of the model are connected to a runtime infrastructure providing the system-level support for executing a business transaction. Each business level construct is appropriately mapped to a corresponding infrastructure primitive(s) that that can be found in Web services standards, such as BPEL, WS-Coordination, WS-AtomicTransaction and WS-BusinessActivity. For instance, constructs such as activities, sequences and roles map directly to BPEL constructs as presented in [6], while vital business level atomicity criteria map directly to WS-AtomicTransaction and non-vital atomicity criteria map to WS-BusinessActivity. Currently, the above set of Web services standards is used to implement business transactions. This infrastructure is based on open an source implementation framework provided by JBoss Transactions (http://www.jboss.org) which supports the latest Web services transactions standards, providing all of the components necessary to build interoperable, reliable, multi-party, Web services-based applications quickly and easily.

Figure 5 also shows how QoS criteria can be registered in an SLA. An SLA contains several entries that are related to a business transaction. These include the scope of the agreement (the services covered in the agreement), penalties (sanctions should apply in case the service provider under-performs and is unable to meet the objectives specified in the SLA), optional services (any services that are not normally required by the user, but might be required in case of an exception) and exclusion terms (specify what is not covered in the SLA). QoS criteria in the context of a business transaction are expressed as assertions by an assertion sub-language of BTML. This assertion language is an extension of the WS-Policy assertion language [7] thereby reusing existing functionality like

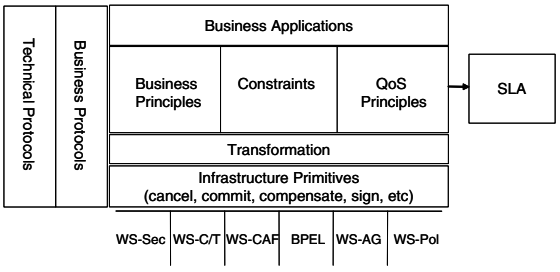

Fig. 5. Business transaction reference architecture

normal form, referential and combined policies. BTML's assertion language also contains context specific assertion definitions for a business transaction. This part of the BTML can then be incorporated as guarantee terms into agreements templates specified by WS-Agreement [8] to enable the specification, negotiation and acceptance of SLAs that are used to drive business transactions.

Finally, the reference architecture supports the use of business and technical protocols in the context of the business transaction model. Currently, the architecture supports one business protocol namely, RosettaNet, and one technical protocol, the Secure Electronic Transactions (SET) [9]. In the following section we however concentrate on illustrating how to semantically enhance the RosettaNet business protocol, which lacks the notion of a business transaction as defined in Sect. 2, by injecting into it business functions, explicit sequencing of interactions, partner commitments and constraints.

6 Emulating Business and Technical Protocols

In this section we will illustrate how we can supplant transactional primitives into the integrated logistics scenario in Fig. 2 to semantically enhance the operational characteristics of the interacting RosettaNet processes.

This procedure is performed according to the following steps. We start first by grouping the individual PIPs into related sets that realize a specific common business function. We observe that PIPs 4A2, 4A5 and 4C1 are all part of the *order* business function. PIP 3B2 is part of the *transport* function and PIPs 4B2 and 4B3 are part of the *delivery* function. The *payment* function is covered by PIPs 3C3 and 3C6.

Following this we need to capture the message and commitment exchange requirements between any trading partners, identifying the timing and sequence of message exchanges. We assume that the trading partners have agreed on a business protocol (developed on the basis of RosettaNet) which requires that *payment* follows *order* and *delivery*. This is specified in BTML as shown in Sect 4. Subsequently, we can specify the business functions using BTML. We assume that in the integrated logistics example the customer and the supplier have agreed on an all or nothing express delivery method, which specifies that

Listing 1

```
<BF>
 <name>Delivery</name>
  <Means>
   <DeliveryMeans>Air</DeliveryMeans>
  </Means>
  <Method>
   <DeliveryMethod>Complete</DeliveryMethod>
   <TransportCompany>UPS</TransportCompany>...
  </Method>
  <Goods><!-- References Goods Business Objects -->
  </Goods>
  <Change>
   <permitted>true</permitted>
   <element>location</element>
   <numberoftimes>2</numberoftimes>
   <prize monetary="$">150</prize>
   <paymentmeans>invoice</paymentmeans>
  </Change>...
</BF>
```

Listing 2

```
<BF>
 <name>Payment</name>
 <BusinessProtocol>
  <participant>
   <name>SteelWorks</name>
   <role>Supplier</role>
   <activities>
    <activity>
     <name>Create&Send Invoice</name>
      <messages>
       <messageOutgoing>InvoiceMessage</messageOutgoing>...
      </messages>
     </activity>...
    </activities>
   </participant>...
   <sequence>
    <activity>Create&Send Invoice</activity>
    <activity>Receive&Check Remittance Advice</activity>
    <selective>
     <sequence>
      <activity>Accept Remittance Advice</activity>
      <activity>Process Remittance Advice</activity>
     </sequence>
     <activity>Decline Remittance Advice</activity>
    </selective>
   </sequence>...
  </BusinessProtocol>...
</BF>
```

Listing 3

```
<activities>
 <activity>
  <name>Create&Send Invoice</name>
  <messages>
   <messageOutgoing>
    <name>Invoice Notification</name>
    <acknowledgeable>true</acknowledgeable>
    <tta type="maxdurationinhours">2hours</tta>
    <signed>
     <encryptionalgorithm keylength ="1024">
      DES</encryptionalgorithm>
     <hashalgorithm keylength="256">
      SHA</hashalgorithm>...
   </messageOutgoing>...
  </messages>
 </activity>
</activities>
```

Fig. 6. Listings of BTML snippets

if the goods are ready for transport the delivery should not take more than two days (which requires delivery by air). The specification in BTML can be found in Listing 1 of Fig. 6. Listing 1 also specifies that the delivery location is changeable at most twice at a cost of 150 $ per time and that the fees will be added to the original invoice. Listing 2 of the same figure specifies part of a simple payment protocol seen from the vantage point of the supplier. Finally, Listing 3 adds QoS properties to the elements of the business transaction. In particular, we may wish to indicate that the InvoiceNotification process (PIP 3C in Fig. 2) requires that the time to acknowledge an Invoice Notification message send from the Create&Send Invoice activity of the Supplier to the Receive Invoice activity of the Customer is no longer then 2 hours. This property can be specified using the Responsiveness primitive in the General QoS field in Fig. 3. The QoS Responsiveness primitive has an operator called setAcknowledgeable that specifies whether a particular message should or should not be acknowledgeable and also the time frame for this to happen. We may also wish to add other QoS constraints on messages or message parts. For example we may wish to specify further security primitives indicating whether or not a message or message part should be non-reputable or signed with a particular hash and encryption function. All of this can be specified in BTML as shown in Listing 3 of Fig 6.

Another important aspect of the business reference architecture is that it can blend business with technical protocols. For instance, the business application that we sketched in the previous does not have any concrete way to handle the actual payment so that it can transfer funds using a financial service provider.

This situation also holds for the RosettaNet PIPs. To remedy this situation we also extended the payment part of the business transaction described in the previous with a technical protocol, such as SET, that guarantees secure payments [10].

7 Related Work

Automated business transactions are a new category of research, wider than historical data-centric local, distributed of federated transactions. This *third generation* of transaction management builds out from core transactional technology, particularly the concept of a open nested transactions and multi-phase distributed outcomes (*two-phase commit* in conventional database/messaging transactions). Research in this paper was inspired by the work found in [11], which motivates the need for using transactions that mimic real business exchanges and presents an overview of several technologies and protocols that may support a business transaction framework. Research in the business transactions area is also related to the creation of meta-models for Web service transaction models. In [12], a meta-modeling approach to transaction management is proposed; that approach however focuses on the modeling and representation of transaction models driven purely from database technology perspective without taking into account business and workflow requirements. To support our implementation efforts, the work found in [13] is used, where the authors propose to combine multiple transaction models as WS-C coordination types into BPEL specifications that can support transactional workflows. The work reported in this paper can also benefit from other ongoing research in the SOC domain. Of particular interest is the work on SLAs reported in [14]. Here, the authors define a template-based approach that enables automated service provisioning. This provisioning can be guided by the WS-Agreement [8] protocol. Finally, the work reported in [15] is quite relevant as it describes many non-functional properties applicable for Web services that can also benefit business transactions.

8 Summary

In the previous we have described a business transaction model, business transaction specification language and associated reference architecture. Key characteristics of this model is that it sharply distinguishes between a business related and a systems related view of transactions. At the business-level, the transactions of our model are weaved around commonly standard business functions that apply to a variety of application scenarios and can represent business exchanges, the sequencing and timing, business agreements stipulated in SLAs, liabilities and dispute resolution policies, and blends these transactions with QoS criteria. Business transactions in the systems-level retain the driving ambition of consistency and provide support for conventional ACID as well as open-nested long-running transactions. Implementation of the systems-level services is currently provided by Web services standards like BPEL, WS-Coordination, and WS-Transaction.

The potential benefits of this approach arise largely from its ability to standardize common business functions, better align business processes with business objectives and provide information to enable monitoring and troubleshooting of problems and delays. Business decisions can be made at every step of the business transaction to align it with business objectives and to alleviate undesirable conditions. For example, in case of a purchase order cancellation due to a faulty part, an order transaction can automatically reserve a suitable replacement product and notify the billing and inventory processes of the changes. When all interactions between the various business processes have been completed and the new adjusted schedule is available, the purchase order Web service notifies the customer sending her an updated invoice.

References

1. Cabrera, L.F., et al.: Web Services Coordination (2005)
2. Cabrera, L.F., et al.: Web Services Atomic Transaction (2005)
3. Cabrera, L.F., et al.: Web Services Business Activity Framework (2005)
4. Bunting, D., et al.: Web Services Composite Application Framework (2003)
5. RosettaNet: Standards required to support xml-based b2b integration (2001) http://xml.coverpages.org/rosettanetStandardsForIntegration.pdf.
6. Khalaf, R.: From rosettanet pips to bpel processes: A three level approach for business protocols. In BPM, Proceedings. Volume 3649 of LNCS (2005) 364–373
7. Bajaj, S., et al.: Web Services Policy 1.2 - Framework (WS-Policy). W3C (2006) http://www.w3.org/Submission/WS-Policy/.
8. Andrieux, A., et al.: Web Services Agreement Specification (WS-Agreement). TR, Grid Resource Allocation Agreement Protocol (GRAAP) WG (2005)
9. Merkow, M.S., et al.: Building SET Applications for Secure Transactions. Wiley & Sons, USA (1998)
10. Kratz, B.: Emulating SET in BTML. ITRS 30, Infolab, Tilburg University (2006)
11. Papazoglou, M.: Web services and business transactions. World Wilde Web: Internet and Web Information Systems **6**(1) (2003) 49–91
12. Hrastnik, P., Winiwarter, W.: Using advanced transaction meta-models for creating transaction-aware web service environments. International Journal of Web Information Systems **1**(2) (2005) 89–99
13. Tai, S., et al.: Transaction policies for service-oriented computing. Data & Knowledge Engineering **51** (2004) 59–79
14. Ludwig, H., et al.: Template based automated service provisioning supporting the agreement driven service life-cycle. In ICSOC 2005, Proceedings. Volume 3826 of LNCS (2005) 283–295
15. O'Sullivan, J., et al.: Formal description of non-functional service descriptions. TR, QUT (2005) http://www.bpm.fit.qut.edu.au/about/docs/non-functional.jsp.