UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

38050 Povo – Trento (Italy), Via Sommarive 14
http://www.dit.unitn.it

WEB SERVICES AND BUSINESS TRANSACTIONS

Michael P. Papazoglou

2003

Technical Report # DIT-03-014


World Wide Web: Internet and Web Information Systems, 6, 49–91, 2003

Web Services and Business Transactions *

MICHAEL P. PAPAZOGLOU mikep@uvt.nl
Infolab, Tilburg University, PO Box 90153, 5000 LE, Tilburg, Netherlands

Abstract

Process oriented workflow systems and e-business applications require transactional support in order to orchestrate loosely coupled services into cohesive units of work and guarantee consistent and reliable execution. In this paper we introduce a multi-level transaction model that provides the necessary independence for the participating resource managers, e.g., local database and workflow servers, of organisations engaging in business transactions that are composed of interacting web services. We also present a taxonomy of e-business transaction features such as unconventional atomicity criteria, the need for support for business conversations and the need for distinguishing between three basic business transaction phases. In addition, we review current research and standard activities and outline the main ingredients of a business transaction framework necessary for building flexible e-business applications.

Keywords: web service, workflow system, business process orchestration, business transaction, business protocol, business conversation

1. Introduction

Electronic-business applications are based on a sequence of business messages that are exchanged between enterprises, their content, precise sequence and timing, for the purpose of carrying a business activity or collaboration such as securities trade settlement. These types of business functions are commonly known as business processes and could be both internal and external to organisations.

A *business process* is a set of logically related tasks performed to achieve a well defined business outcome. We may view an automated business process as a precisely choreographed sequence of activities systematically directed towards performing a certain business task to completion. For example, processing a credit claim, hiring a new employee, ordering goods from a supplier, creating a marketing plan, processing and paying an insurance claim, and so on, are all examples of typical business processes. An *activity* is an element that performs a specific function within a process. Activities can be as simple as sending or receiving a message, or as complex as coordinating the execution of other processes and activities. A business process may encompass complex activities some of which run on back end systems such as, for example, a credit check, automated billing, a purchase order, stock updates and shipping, or even such frivolous activities as sending a document, and filling a form. Typically, a workflow application is associated with business

* Part of this work has been funded by the Telematica Institute under the project PIEC.

processes and their activities. This is irrespectively whether the issue is the management of the logistics chain, the marketplace or the exchange of documents between partners.

With e-business applications business processes need to cross-organisational boundaries, i.e., they occur across organisations or between organisational subunits. In such situations business processes can drive their collaboration to achieve a shared business task by enabling highly fluid networks of collaborating web services. Accordingly, in a web service environment the business process workflow is made up of activities that are implemented as a set of scoped operations that may span a single or possibly multiple web services. These web service centric activities follow routes with checkpoints represented by business conditions and rules. *Business rules* are compact statements about an aspect of the business that can be expressed within application [37]. Business rules can represent among other things typical business situations such as escalation ("send this document to a supervisor for approval"), managing exceptions ("this loan is more than $500K, send it to the CFO"), or progression assurance ("make sure that we deal with this within 30 min or as specified in the customer's service-level agreement").

1.1. Cross-enterprise business processes and web services

One key requirement in making cross-enterprise business process automation happen is the ability to describe the collaboration aspects of the business processes, such as commitments and exchange of monetary resources, in a standard form that can be consumed by tools for business process implementation and monitoring. Enterprise workflow and business process management systems today support the definition, execution and monitoring of long running processes that coordinate the activities of multiple business applications. However, the loosely coupled, distributed nature of the Web prevents a central workflow authority (or a centralized implementation of middleware technology) from exhaustively and fully coordinating and monitoring the activities of the enterprise applications that expose the web services participating in message exchanges [1]. The reasons are as follows:

- Traditional workflow models rely on message-based computing methods that are tightly coupled to protocols for e-business or application integration. These protocols assume a tightly linked and controllable environment, which is not the nature of the Web.
- Traditional workflow models do not separate internal implementation from external protocol description.
- Participants in traditional workflow are, normally, known in advance. When moving to the Web environment, it is highly probable that web services will be dynamically chosen to fulfil certain roles.

When processes span business boundaries, loose coupling based on precise external protocols is required because the parties involved do not share application and workflow implementation technologies, and will not allow external control over the use of their backend applications [35]. Such business interaction protocols are by necessity message-centric: they specify the flow of messages representing business activities among trading partners (without requiring any specific implementation mechanism).

With e-business applications trading partners must run their own, private business processes (workflows), which need to be orchestrated and coordinated to ensure that the outcome of the collaborative business process is reliable and does not fail or hang waiting for another processes or system to respond to a request. This requires that processes be able to communicate asynchronously when appropriate and is opposed to business process integration schemes based on synchronous technologies such as CORBA or DCOM. Such asynchronous behaviour is exhibited by web services.

Web services technology promises to facilitate this undertaking by replacing proprietary interfaces and data formats with a standard web-messaging infrastructure. Web messaging is sufficient for some simple application integration needs; however, it does not adequately support the complete automation of critical business processes.

The first generation of web services technology has largely focussed on the web-messaging foundation supported by SOAP [33] and WSDL (Web Services Definition Language). WSDL [8] is used to describe a web service in terms of its *ports* (addresses implementing this service), *port types* (the abstract definition of operations and exchanges of messages), and *bindings* (the concrete definition of which packaging and transportation protocols, such as SOAP, are used to inter-connect two conversing end points). Although this foundation has the ability to specify critical information and requirements relating to the business process context, it does not support business processes that cross organisational boundaries.

A business process choreography consists of the aggregation of web services by encoding business rules or patterns governing service interactions and has the ability to reuse the created aggregations [20]. Business logic can be seen as the ingredient that sequences, coordinates, and manages interactions among web services. To program a complex cross-enterprise workflow task or business transaction, for example, it is possible to logically chain discrete web service activities into inter-enterprise business processes.

The problem of choreographing web services is tackled by a trio of standards that have been recently proposed to handle this next step in the evolution of Web services technology. The standards that support business process orchestration are: Business Process Execution Language for Web Services (BPEL4WS or BPEL for short) [9], WS-Coordination (WS-C) [6] and WS-Transaction (WS-T) [7]. BPEL4WS is a workflow-like definition language that describes sophisticated business processes that can orchestrate web services. WS-Coordination and WS-Transaction complement BPEL4WS to provide mechanisms for defining specific standard protocols for use by transaction processing systems, workflow systems, or other applications that wish to coordinate multiple web services. These three specifications work in tandem to address the business workflow issues implicated in connecting and executing a number of web services that may run on disparate platforms across organisations involved in e-business scenarios.

1.2. The need for a business transaction framework

In the world of web services, a business process specifies the potential execution order of operations originating from a collection of web services, the shared data passed between

these services; the trading partners that are involved in the joint process and their roles with respect to the process; joint exception handling conditions for the collection of web services; and other factors that may influence how web services or organisations participate in a process [21]. This allows, in particular, specifying long-running transactions between web services in order to increase the consistency and reliability of business processes that are composed out of web services.

Business process orchestration requirements involve asynchronous interactions, flow coordination, business transaction activity and management. These are common to all e-business applications that need to coordinate multiple web services into a multi-step business transaction. Thus the web services environment requires that several web service operations have transactional properties and be treated as a single logical unit of work performed as part of a business transaction. 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 parts are delivered to their final destination, which could days or weeks after the placement of the order.

A *business transaction* is a consistent change in the state of the business that is driven by a well-defined business function. Usually, a business process is composed of several business transactions. In a web service environment business transactions essentially signify transactional web service interactions between organisations 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, coordinating and managing marketing strategies, determining new product offerings, granting/extending credit, managing market risk and so on. Such complex business transactions are usually driven by interdependent workflows, which must interlock at points to achieve a mutually desired outcome. This synchronization is one part of a wider business coordination protocol that defines the public, agreed interactions between business parties.

When a business function is invoked through a web service as part of a larger business process, the overall transactional behaviour associated with that business process depends on the transactional capabilities of the web service. Moreover, a business process may involve one or more business functions (invoked through one or more web services). To support the desired transactional behaviour, coordination across all or some of these participants may be required. Such coordination must respect the loose coupling and the autonomy of web services and must be imposed by a business coordination protocol. Without a failure-resilient coordination protocol business interactions become cluttered with multiple out-of-band messages for error recovery, which throws responsibility onto the internal business processes in respect of durable recording of the state of progress. Therefore, the motivation is to create a business transaction framework (BTF) that orchestrates loosely coupled web services into a single business transaction by offering transactional support in terms of coordinating distributed autonomous business functionality and guaranteeing co-

ordinated, predictable outcomes for the trading partners participating in a shared business process.

BTF should rely on a web services orchestration infrastructure that offers the ability to create complex processes by defining different activities that invoke web service invocations operations that are part of dynamic service compositions, manipulate and coordinate data flows, use exception and error handling mechanisms, or terminate a process. Such advanced functionality is typically offered by the new BPEL standard. The BTF should be layered on top of the process orchestration infrastructure and should provide the following essential components:

1. Support for a *business transaction model* for expressing long-lived e-business transactions, conventional short duration transactions, exception handling mechanisms and compensating activities as well as business centred atomicity criteria.
2. Support for *coordination protocols*. The BTF should be able to provide protocols that leverage business related transactional mechanisms for coordinating the operations of web services across distributed applications. The framework should enable an application to create the context needed to propagate an activity to other services and to register for coordination protocols. The BTF should also enable existing workflow and transaction processing systems of organisations engaging in business transactions to coordinate their activities and interoperate by hiding their proprietary protocols and heterogeneity.
3. Support for *Business Protocols*. The BTF should be able to capture the information and exchange requirements between trading partners (business conversations), identifying the timing, sequence and purpose of each business collaboration and information exchange. In other words, the business protocol defines the ordering in which a particular partner sends messages to and expects messages from its partners based on an actual business context and captures all behavioural aspects that have cross-enterprise business significance. Each participant can then understand and plan for conformance to the business protocol without engaging in the process of human agreement that adds so much to the difficulty of establishing cross-enterprise automated business processes. An example of a business protocol is the RosettaNet Partner Interface Processes (PIPs).

A business protocol may exhibit the flowing characteristics [9]:

(a) It invariably includes data-dependent behaviour. For example, a supply-chain protocol depends on data such as the number of line items in an order, the total value of an order, or a deliver-by deadline. Defining business intent in these cases requires the use of conditional and time-out constructs.
(b) It is able to specify exceptional conditions and their consequences, including recovery sequences.
(c) It relies on the support of long-running interactions (business conversations) that include multiple, often nested units of work, each with its own data requirements. Business protocols frequently require cross-partner coordination of the outcome (success or failure) of these units of work at various levels of granularity.

We collectively refer to the business transaction model and coordination protocol of a BTF as its *Business Transaction and Coordination Protocol* (BTCP).

This paper introduces a Business Transaction Framework based on networks of collaborating web services and outlines its requirements, essential characteristics and building blocks. The paper also introduces a set of functional criteria for the BTF and measures standard initiatives such as the Business Process Execution Language for Web Services (BPEL), WS-Transaction, WS-Coordination, the Business Transaction Protocol (BTP), and the ebXML Business Process Specification Schema (BPSS) against them.

In what follows we shall provide an overview of essential BT characteristics and functional criteria that we consider as core requirements for a BTF. Subsequently, we describe the BPEL standard web services orchestration infrastructure, which is the foundation for on which the BTF building blocks rely, and explain how it supports business transaction and coordination mechanisms as well as business protocols. Finally, we concentrate on the essential components of a BTF, namely its Business Transaction and Coordination Protocol and Business Protocol, and explain how current standard and research activities on business transactions and business architectures contribute towards achieving them. We also explain how these activities measure against the stated BTF functional criteria.

2. Business transaction model

Conventional transactions operate beyond corporate firewalls while transaction monitors have total control over the resources that participate in a transaction. Every participant in a transaction either commits or fully rolls back rolls back in unison. This ACID (atomic, consistent, durable, isolated) transaction model involves transactions that are tightly coupled and occur between trusted systems over short periods of time. Although extremely reliable, classic ACID transactions are no suitable for loosely coupled environments such as web service based business transactions.

In a web services environment transactions are complex, involve multiple parties, span many organisations, and can have long duration. More specifically, business transactions are automated long-running propositions involving negotiations, commitments, contracts, shipping and logistics, tracking, varied payment instruments, and exception handling. Performance of these business related tasks requires the infusion of transactional properties onto the web services paradigm. Business transactions exhibit the following characteristics:

1. They normally represent a function that is critical to the business, e.g., supply-chain management.
2. They can involve more than two parties (organisations) and multiple resources operated independently by each party, such as business applications, databases and ERP systems.
3. They define communications protocol bindings that target the domain of web services, while preserving the ability to carry business transaction messages also over other communication protocols. Protocol message structure and content constraints are schematised in XML, and message content is encoded in XML instances.
4. They should be based on a formal trading partner agreement, expressed in terms of business protocols such as RosettaNet PIPs or ebXML Collaboration Protocol Agreements

(CPAs) [12]. An agreement represents a "contract" between two or more partners to carry out specific functions in a public business process.

Electronic-business applications require transactional support beyond classical ACID and extended transactions. Strict ACIDity and isolation is not appropriate to a loosely coupled world of autonomous trading partners, where security and inventory control issues prevent hard locking of local databases. The major issue is the isolation of a database transaction. This property requires resource locking that is impractical in the business world. It would also preclude users from participating in a business process, which is a strong requirement for e-business applications. Sometimes in a loosely coupled or long running activity it may be desirable to cancel a work unit without affecting the remainder. This is very similar to the properties of nested transactions where the work performed within the scope of a nested transaction is provisional and sub-transaction failure does not affect the enclosing transaction.

Due to their long-lived nature and multi-level collaborations, BTs require support for a variety of unconventional behavioural features that are summarised as follows:

1. *Generic characteristics*:
   (a) who is involved in the transaction;
   (b) what is being transacted;
   (c) the destination of payment and delivery;
   (d) the transaction time frame; e.g., a timeout value that defines the maximum amount of time during which the transaction should be active;
   (e) permissible operations.
2. *Distinguishing characteristics*:
   (a) links to other transactions;
   (b) receipts and acknowledgments;
   (c) identification of money transferred outside national boundaries.
3. *Advanced characteristics*:
   (a) the ability to support reversible (*compensatible*) and repaired (*contingency*) transactions;
   (b) the ability to reconcile and link transactions with other transactions;
   (c) the ability to specify contractual agreements, liabilities and dispute resolution policies;
   (d) the ability to support secure transactions that guarantee integrity of information, confidentiality, privacy and non-repudiation;
   (e) the ability for transactions to be monitored, audited/logged and recovered.

Business transactions usually operate on document-based information objects such as *documents* and *forms.* A document is traditionally associated with items such as purchase orders, catalogues (documents that describe products and service content to purchasing organisations), bids and proposals. A form is traditionally associated with items such as invoices, purchase orders and travel requests. Both these media are arranged according

to some predefined structure. Forms-based objects are closely aligned with simple business transactions that have a numerical or computational/transformational nature while document-based objects are usually associated with agreements, contracts or bids. This allows business transactions to interchange everything from product information and pricing proposals to financial and legal statements. It also allows coordinating autonomous parties whose relationships are governed by contracts rather than by the dictates of a central design authority. That is, the ability to allow transactions to run across arbitrary transaction-aware web services.

2.1. Business transaction types

When a business function is invoked through a web service as part of a larger business process, the overall transactional behaviour associated with that business process depends on the transactional capabilities of the web service. Rather than having to compose ever more complex end-to-end offerings, application developers choose those elements that are most appropriate, combining the transactional and non-transactional web-service fragments into a cohesive BT-service whole. At run-time the BTF will manage the flow of control and data between the business processes and will establish transaction boundaries around them as defined in the transaction script.

A BTF may utilize workflow technology to bundle web services together and provide the sequence of business activities, arrangement for the delivery of work to the appropriate organisational resources; tracking of the status of business activities; coordination of the flow of information of (inter and intra-) organisational activities and the possibility to decide among alternative execution paths [25]. In [26,39] we proposed two kinds of business transactions for e-business applications (on which we expand in this paper):

- *Atomic business transactions* (or service-atoms): these are small scale interactions made up of services that all agree to enforce a common outcome (*commit* or *abort*) of the entire transaction. The atomic transaction follows the ACID properties and guarantees that all participants will see the same outcome (atomic). In case of a success all services make the results of their operation durable (commit). In case of a failure all services undo (*compensate*, *roll-back*) operations that they invoked during the transaction. The atomic transaction could be nested (close nesting model) and gives an all or nothing guarantee to the group of atomic activities that are executed as part of the transaction.
- *Long-running business transactions* (or BTs for short)[1]: these are aggregations of several atomic transactions and exhibit the characteristics and behaviour of open nested transactions [13]. BTs are non-atomic in the sense that they allow the *selective confirm (commit) or cancel (rollback) of participants* (even if they are capable of committing). The atomic transactions forming a particular BT do not necessarily have a common outcome. Under application control (business logic), some of these may be performed (confirmed), while others may fail or raise exceptions such as time outs or failure. Failed BTs are cancelled with their operations are rolled back.

To understand the nature of atomic transactions we introduce a simple example. Assume that a client application (initiator in Figure 1) decides to invoke one or more

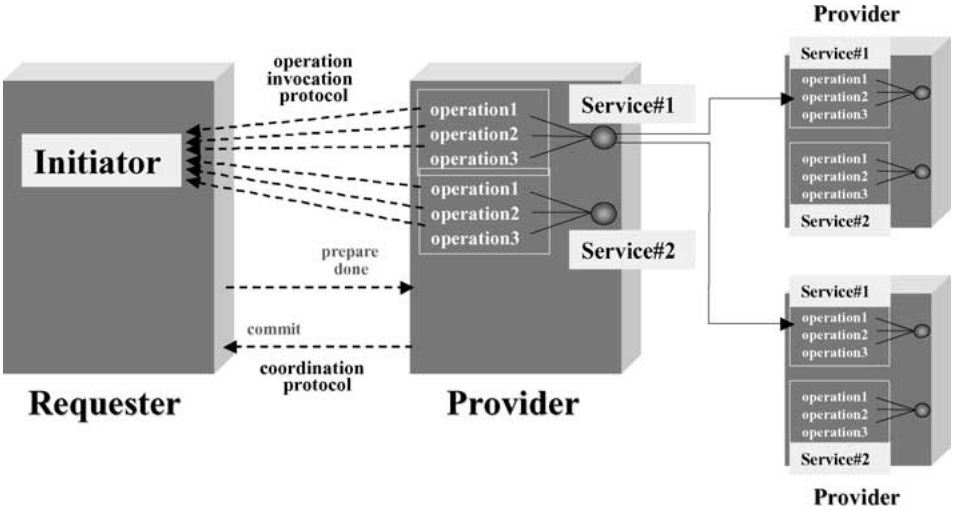

Figure 1. Actors in an atomic transaction and their invocations.

operations from a particular service (provider in Figure 1) such as orderFulfillment, or associatedShipping. It is highly likely for the client application to expect these operations to succeed or fail as a unit. We can thus view the set of operations used by the client in each web service (requester in Figure 1) as constituting an atomic unit of work (atomic transaction). An atomic transaction is the simplest unit of work and behaves like existing X/Open XA-compliant, two-phase commit transactions. An atomic transaction follows the traditional ACID properties and must either fully commit or fully rollback. Within an atomic transaction, the operations exposed by a single transactional web service and the internal processes of the service, e.g., support processes, would usually make up a single atomic transaction. Atomic web services frequently appear when the business model represents one of the core business processes, e.g., order entry, of an enterprise. Non-atomic web service activity implementations support the service atoms and are frequently found within ancillary processes, e.g., travel expense accounts.

Atomic transactions use a two-phase commit protocol (with presumed abort) whereby a coordinating process is required to manage the coordination protocol messages, e.g., prepare, commit, cancel, that are sent to the participating services within a given atomic transaction (see Figure 1). This coordinating process might be implemented within the application itself, or more likely, it will be a specialised web service [38]. Once the actual work involving the consumed web services in an atomic transaction has finished, the client application (requester) can begin the two-phase commit coordination of those web services. The client (initiator) is expected to *control all aspects of the two-phase commit protocol*, i.e., prepare phase and confirm phase. The rationale behind allowing the client to govern when prepare and confirm calls are made on the atomic transaction is to permit maximum flexibility within the protocol. Allowing the client to decide upon timings implicitly permits reservation-style business processes to be carried out with ease. For instance, this can apply to a hotel or aircraft reservation system, where the prepare phase

of the two-phase commit protocol reserves a room or seat, and the confirm phase actually buys the reserved hotel room or plane seat.

The function of a BT is to aggregate atomic transactions with conventional business logic functions into a cohesive BT. These transactions are also known in the literature as open nested transactions or Sagas [13,16]. This model supports transaction interleaving with arbitrary nesting. In other words, a transaction may be composed of other transactions. A BT is a set of service atoms that can be manipulated by the BT's initiator (typically a client application). A client application can dictate whether a service atom within the BT succeeds or fails, even if the service is capable of succeeding. To exemplify this consider the following case [38]. In an e-business application one service atom arranges for the purchase of a valuable object, while a second arranges for its insurance, and a third one for its transportation. If the client application is not risk-averse, then even if the insurance atom votes to cancel, then the client might still confirm the BT and get the item shipped uninsured. Most likely, however, the client would simply like to retry to obtain insurance for the item. In this case the web services infrastructure comes into play, and the client would opt for another insurer via the UDDI service infrastructure. Once the client discovers a new insurer, it can try again to complete the BT with all the necessary atoms voting to confirm on the basis of the particular coordination protocol used.

One of the major problems with web services is that their underlying implementations are located remotely and hosted by third parties (providers in Figure 1), thus, there is an increased risk of failure during their application. To cater for this threat a BT may decide to selectively cancel atomic transactions and create new ones during its lifetime. Thus, the membership of BT is established dynamically by the business logic part of a client application. For instance, a service may withdraw its participants because of a timeout. A transaction composition service (invoked by the client application) may take interim "polls" to discover whether service atoms are ready or have been cancelled. This transaction composition service is also able to decide that all service atoms will finally be prepared in one stretch.

BTs have several distinguishing characteristics when compared with traditional database transactions. Firstly, they extend the scope of traditional transaction processing as they may encompass classical transactions, which they combine with non-transactional processes (BTs). Secondly, they group both classical atomic as well as non-atomic computations together into the BT unit of work that reflects the semantics and behaviour of their underlying business task. Thirdly, unconventional types of atomicity govern them.

This split-level approach structures BTs in a flexible way. Using a combination of atomic transactions that deal with smaller interactions, and BTs that deal with larger business transactions, provides a great deal of flexibility in creating an overall transaction scheme that is both robust and flexible [38].

2.2. Modes of business transaction recovery

The business transaction model we described in Section 2.1 relaxes the classical isolation requirement of ACID transactions. There are two possible modes of recovery for BTs: *backward recovery* and *forward recovery*.

Backward recovery is used in case that the transaction aborts. With backward recovery the encompassing business process will return to the consistent state that existed before the execution of the aborted transaction including the contexts of its children, if any. To achieve this outcome the transaction initiates some compensating activity that will cancel the effects of the failed transaction. Compensation is the logic for reverting the effects of a failed transaction. Atomic transactions provide automatic backward recovery. Such backward recovery relates only to activities that are conducted within the context of the atomic transactions. This is not true of BTS. A BT may not be able to guarantee complete backward recovery in all cases as it may include activities with non-reversible side effects. For instance, a cancelled order entry BT may perform backward recovery by returning goods to the supplier but will still incur shipping and restocking costs. For this reason, the BT must define the appropriate business logic to perform backward recovery [1].

Forward recovery is used with BTs only and is a guarantee that in the event of system failure, the transaction state can be restored to a consistent state and its execution can continue reliably past the point of failure. With forward recovery the business process instance is allowed to continue its execution taking into account that the transaction failed. Compensating activities are used wherever the execution of an activity cannot be rolled back (such as when an order is shipped). Atomic transactions in the context of a BT are guaranteed forward recovery upon successful completion (in case that their encompassing BT fails) and will perform backward recovery in case of their own failure.

2.3. Atomicity criteria

BTs are governed by unconventional types of atomicity much in the spirit of business applications that use them. It is convenient to classify atomicity along the following broad categories: *system-level atomicity*, *operational-level atomicity* and *business interaction-level atomicity*. The different types of atomicity criteria and their inter-relationships are depicted in Figure 2.

1. *System-level atomicity*. This category contains a single default atomicity criterion that is enforced at the system-level and is applicable to all operations of web services that have been declared of transactional nature.
   - *Service request atomicity*. A single operation on a transaction-aware web service occurs completely or not at all. This is a capability that the end-point publishes to its users. The end-point may implement the functionality it offers by an internal transaction on its infrastructure, or some other mechanism. Service request atomicity simply implies that each service provider offers services and operations (and guarantees) that these will complete as an atomic piece of work.
2. *Business interaction-level atomicity*. This category encompasses the atomicity primitives that are required when organisations "discuss" and negotiate trading relationships and terms before conducting any real business. Business interactions are normally based on standard business protocols and involve conversation sequences and document exchanges.

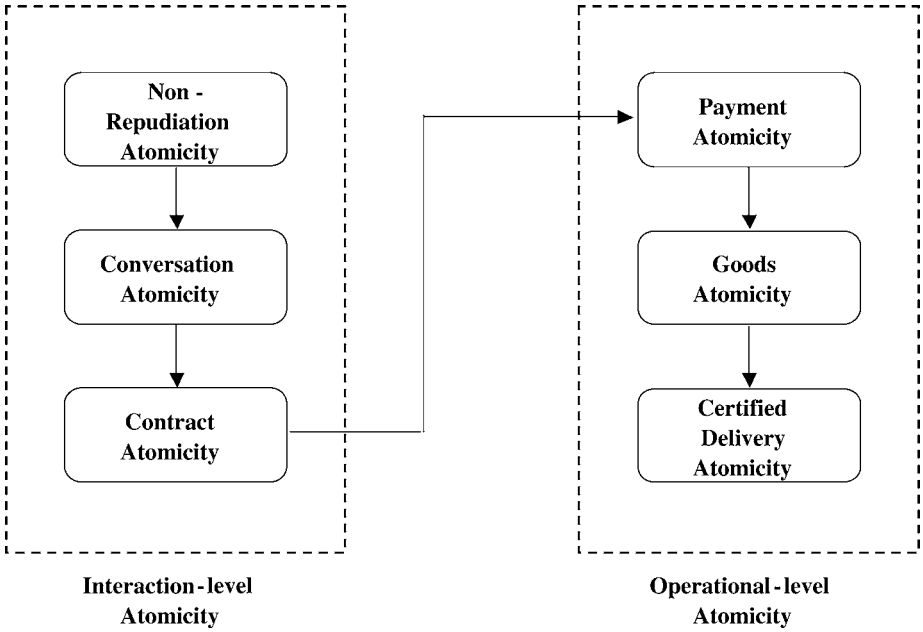

Figure 2. Classification of BT atomicity types.

- *Non-repudiation atomicity*. Establishing non-repudiation in BTs involves a process of digitally signing the content of the transaction at an application level. The transaction is then not only verified and authorized based on the identity of the user who executed it, but is saved in signed form. The history of saved, signed, transactions is protected against modification by the nature of the public key digital signature — a property not shared by other types of electronic signatures. This transaction history can be analysed in the event of a dispute. While there is not anything absolute about implementing a non-repudiation transaction system, applying digital signatures to transactions in this way puts the high-value service provider in the best legal position in dealing with disputes[2].
- *Conversation atomicity*. This is a non-repudiation atomic primitive that allows a pair of collaborating services to correlate sequences of requests within a logical unit of work. The pair of services uses architected conversation messages and headers to begin the conversation and end the conversation. These determine if the conversation ended successfully or if one or both participants want the conversation to rollback. The semantics of rollback is that each participant (service) can undo the operations it has performed within the conversation and expect its counterpart service to revert back to a consistent state. Furthermore, a service can rely on the transactional behaviour of its counterpart services to ensure state consistency in the presence of failures, such as logical error conditions, e.g., shipping is impossible, or system-level failures, e.g., crash of a process or a network failure. Conversation atomicity requires that more complex processing and compensation models are employed and that service

providers can build on their internal transaction and business logic environments to provide this increased flexibility.

- *Contract atomicity*. BTs are generally governed by contracts and update accounts. These are normally based on electronic business protocols that include the exchange of financial information services and the exchange of bills and invoices. Electronic contracts define both the legal terms and conditions and technical specifications that trading partners must implement to put an electronic trading relationship into effect. Thus, contract atomic BTS are also conversation atomic. If a contract atomic BT succeeds it may be designated as legally binding between two (or more) trading partners, or otherwise govern their collaborative activity. If it fails it is null and void, and each partner must relinquish any mutual claim established by the transaction. This can be thought of as "rolling back" the BT upon failure.

3. *Operational-level atomicity*. BTs usually involve the actual delivery of purchased items (tangible or non-tangible). The operational-level atomicity primitives introduced in the following govern such transactions. The notion of operational-level atomicity introduced in this paper refines similar principles reported in [36] and applies them to transaction-aware web services.

   - *Payment atomicity*. Payment-atomic protocols affect the transfer of funds from one party to another. Payment atomicity is the basic level of atomicity that each operational-level BT should satisfy. For many e-business applications payment-atomic protocols must also be contract-atomic. However, in other cases the link between contract and payment atomicity might be tenuous. It really depends on the application semantics whether a payment atomic transaction is also contract atomic. When a payment atomic transaction is also contract atomic, a reference should exist in the body of the payment atomic transaction to its contract-id.
   - *Goods atomicity*. Goods atomicity protocols are payment-atomic, and also affect an exact transfer of goods for money. Goods atomicity implies that the (tangible or non-tangible) goods will be received only if payment has been made. The idea is that the event of delivering the goods is part of the BT in which the goods are paid for. In many situations it is not possible to entirely roll back goods atomic transactions. Therefore, there is no automatic roll back for these transactions, instead special business logic needs to be developed in the surrounding application to handle goods atomic BTs on a case by case basis.
   - *Certified delivery atomicity*. A goods-atomic transaction guarantees delivery to the customer, but an additional requirement is that the right goods are delivered. Thus, the certified delivery atomicity primitive is a payment- and goods-atomic protocol that allows both transacting parties to prove exactly which goods were delivered as part of the BT in which the goods were paid for.

Figure 3 shows a high-level business process (specified in a free XML syntax for simplicity) that enables a buyer to issue a purchase order and obtain a response from a seller that acknowledges the purchase order and returns a an offer to the buyer. This business process contains a non-repudiation atomic transaction in its body that operates on purchase order request and confirmation documents.

```
<BusinessProcessSpec xmlns="http://www.BTF.org/BusinessProcess"

  <BusinessDocument name="Puchase Order Request"
     nameID="Pip3A4PurchaseOrderRequest"
     specificationLocation="PurchaseOrderRequest.xsd">

     <Documentation>The document is an XSD file that specifies the rules for
      creating the XML document for the business action of requesting a purchase
      order. The RosettaNet PIP business protocol is followed.
     </Documentation>
  </BusinessDocument>

  <BusinessDocument name="Puchase Order Confirmation"
     nameID="Pip3A4PurchaseOrderConfirmation"
     specificationLocation="PurchaseOrderConfirmation.xsd">
     <Documentation>The document is an XSD file that specifies the rules for
      creating the XML document for the business action of making a purchase
      order confirmation.
     </Documentation>
  </BusinessDocument>

  <BusinessTransaction name="Request Purchase Order"
     nameID="RequestPurchaseOrder_BT">
    <RequestingBusinessActivity name="Purchase Order Request"
       nameID="PurchaseOrderRequestAction"
        <btf:NonRepudiationAtomic>
          timeToAcknowledgeReceipt="...">
            .....
        </btf:NonRepudiationAtomic>
    </RequestingBusinessActivity>

    <RespondingBusinessActivity name="Purchase Order Confirmation"
       nameID="PurchaseOrderConfirmationAction"
       <btf:NonRepudiationAtomic>
          timeToAcknowledgeReceipt="...">
            .....
       </btf:NonRepudiationAtomic>
    </RespondingBusinessActivity>
  </BusinessTransaction>

 </BusinessProcessSpec>
```

Figure 3. Example of a simple business transaction.

The above atomicity primitives are described in terms of XML constructs. The atomicity primitives are programmable constructs that are used as the basis for developing business applications that rely on transaction-aware web services. These primitives allow expressing business application semantics at a higher-level than that of the current standards and can thus be programmed to serve the needs of a large variety of business applications.

2.4. Business transaction states

Once a BT is instantiated it obtains a unique identifier. This identifier is held in the transaction context of the this transaction [1]. The *transaction context* is unique to a transaction

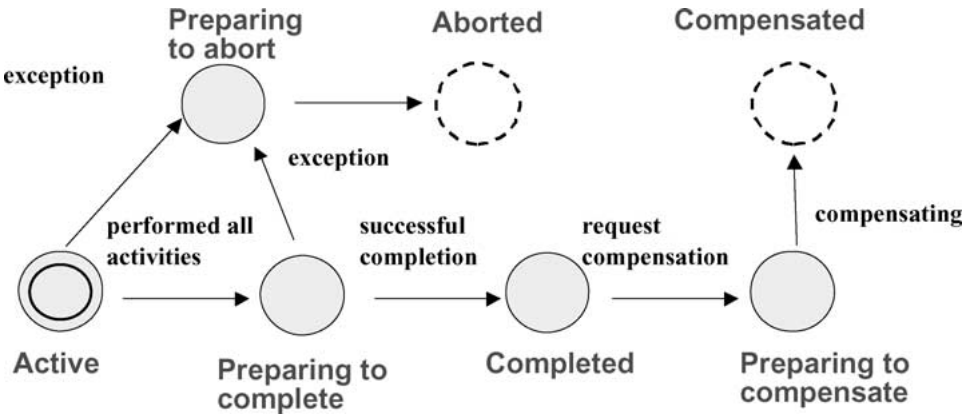

Figure 4. Transition diagram for BT instance states.

instance and holds information that describes a transaction such as parent–child relationships, type of transaction, and all the activities executed in that context, including child contexts that are part of the transaction. If a transaction instance is part of a nested transaction, the instance will be set in the context of the parent transaction. This allows referencing the activities within a transaction even after completion of the transaction, in particular in order to compensate for the transaction.

Each transaction[3] instance transitions through a number of states [1]. These are depicted in Figure 4 and described in what follows:

Active: the transaction is active and performs the activities stated in its transaction context.

Preparing to complete: the transaction has performed all the activities in its activity set and is now preparing to complete. This may involve additional work such as making data changes persistent, performing two-phase commit (for atomic transactions only) and coordinating nested BT completion (viz. context completion).

Completed: the transaction has performed all the work required in order to complete successfully.

Preparing to abort: the transaction has failed to complete successfully and is now preparing to abort. This may involve additional work such as reverting data by running compensating transactions, communicating the outcome to transaction participants (for atomic transactions only) and coordinating nested BT abortion.

Aborted: the transaction has failed to complete successfully and has performed all the work required to abort.

Preparing to compensate: the transaction is now executing activities within its own compensation activity set.

Compensated: the transaction has performed successfully all the work within its own compensation activity set.

A transaction instance always starts in the active state and transitions to the preparing to complete and completion state if all its work is performed successfully. A transaction

will transition to the preparing to abort and aborted if an exception is raised, e.g., time-out, or fault occurs, or one of its parents raises an exception, or the transaction cannot complete successfully. The transaction then performs all the work required, in particular the activities that were related to the exception and need completion. For example, in BPML a transaction event handler can perform any set of activities that is required to assure complete backward recovery of the transaction [1].

It is possible to compensate for a transaction once it is in the complete state (see Figure 4). The initial attempt to compensate a BT will transition it to the compensating state. The transaction will then perform all of the activities in its compensation activity set before moving into the compensated state. Note that only the aborted and compensated states are terminal states (see Figure 4).

2.5. Business transaction phases

As we already explained earlier, one of the major problems with web service based transactions is locking. Participants in a BT are in general unable or unwilling to lock any underlying resources exclusively or indefinitely. In a web services application because participants are outside the firewall and because the transaction process can last too long, it is undesirable to lock the underlying resources. Locking of these resources introduces possible problems such as denial of service attacks and the loss of business that results from such attacks. Therefore, for each party intending to use BT-based messages to implement its business processes it would be useful to distinguish between the following transaction phases:

1. The *pre-transaction* (or application communication protocol) phase: during this phase meaningful business terms such as order information, prices, delivery conditions and so on, are exchanged between trading partners. This phase is the prelude to performing the actual business functions. Automating the exchange of critical information prior to the actual business transaction decreases the impact of out-of-date data, reduces the potential for cancellations, and improves the odds of successfully completing transactions. The application communication protocol may use business conversations to state the intentions of interacting applications prior to committing themselves to a business transaction. Prior to performing a business transaction interacting systems need to exchange application-specific messages to affix tentative commitments to the terms of an interaction, e.g., price, quantity, delivery time, etc. To exemplify this, consider a manufacturing company that orders the parts and services it needs via a web services infrastructure. This manufacturer may have relationships with several parts suppliers and several providers of services such as shipping and insurance. All of the communications between these organisations is via XML messages. Thus, within this example, the manufacturer's and a chosen supplier's applications, could exchange messages detailing what the goods are, their quantity, price and so on. After these terms are accepted the applications will start exchanging messages implementing the main business transaction. This suggests that the parts of the business application (in both interacting systems) that handle these sets of application related messages need to be distinguished

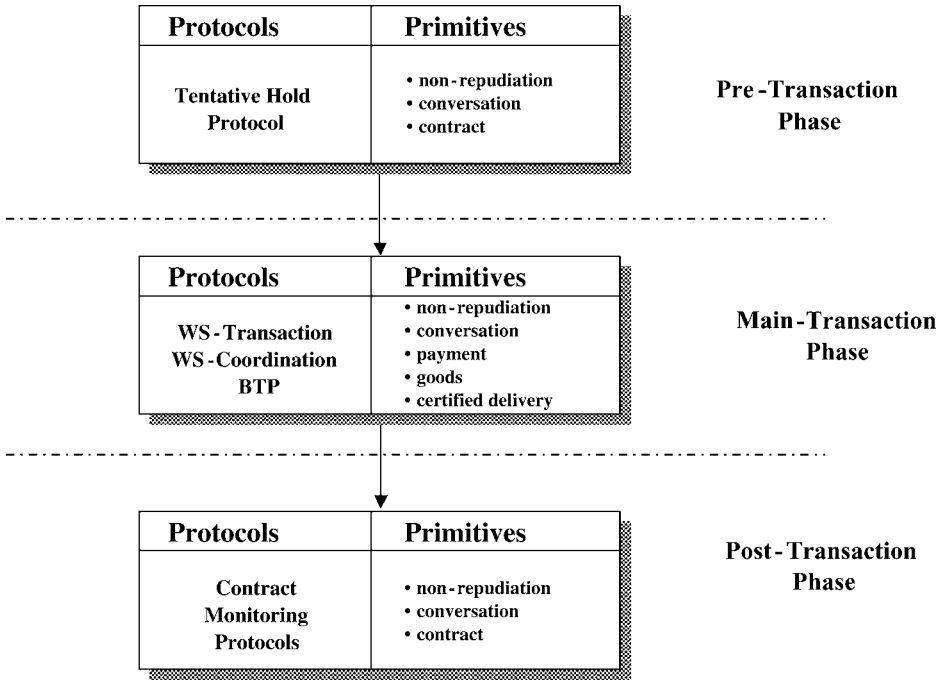

Figure 5. BT phases.

from the main part of a transaction. In general, BTs may involve fairly complex conversation sequences between services that can be performed during or even after the execution of a transaction (main and post-transaction phase).
To program activities that need to take place during the pre-transaction phase the non-repudiation, conversation and contract atomicity primitives can be employed (see Figure 5).

2. The *main-transaction* phase: during this phase the main business processes and transactions are executed. This phase relies on that part of the BTF that supports the definitions of atomic and payment-related business transactions and provides the protocols and infrastructure that coordinates the execution of distributed operations in a web services environment. This infrastructure assists e-business applications in executing business transactions and getting the actual work of the application done. Key elements of this phase include:

   - the ability to support the development of distributed applications that automate and manage the coordination of multi-business transactional and non-transactional interactions;
   - extensions to two phase commit protocol (e.g., Transaction Internet Protocol) that focus on the semantics of business message exchanges; and
   - fault-handling and recovery capabilities.

To program activities that need to take place during the main-transaction phase the non-repudiation, conversation, payment, goods and certified delivery atomicity primitives can be employed (see Figure 5).

3. The *post-transaction* phase: this phase is responsible for observing the agreements and terms stipulated during the execution of a BT. This element corresponds, broadly speaking, to contract fulfilment phase, which follows the main transaction phase, whereby certain future behaviour is stipulated for the transacting parties, the obligations they assume, which are time-bounded, and provisions are made in case an obligation is not fulfilled. Such electronic trading agreements can be expressed in terms of the Trading Partner Agreement Markup Language (tapML) [10], which is an XML-grammar for expressing electronic contracts. The foundation of tpaML is the Trading Partner Agreement (TPA). The TPA is an electronic contract that uses XML to stipulate the general terms and conditions, participant roles, e.g., buyers and sellers, communication and security protocols, and a business protocol (such as valid actions and sequencing rules). A TPA thus defines how trading partners will interact at the transport, document exchange, and business protocol levels.
   The main issues of interest for monitoring the performance of contracts that have been stipulated during a business exchange, include [11]:

   (a) establishing the terms and conditions that each party is obliged (or permitted, or prohibited, or empowered and so on) to observe at a given point in time;
   (b) determining whether each party complies with the behaviour stipulated for it by the agreement; and
   (c) where a party deviates from prescribed behaviour-intentionally or due to force majeure-to determine what, if any, remedial mechanisms are applicable that might return the business exchange to a normal course.

   To program activities that need to take place during the post-transaction phase the non-repudiation, conversation, and contract atomicity primitives can be employed (see Figure 5).

3. Business process orchestration

When looking at web services, it is important to differentiate between the baseline specifications of SOAP, UDDI and WSDL that provide the infrastructure that supports the publish, find and bind operations in the service-oriented architecture and higher-level specifications. Baseline specifications enable users to connect different services across organisational boundaries in a platform and language independent manner. However, these standards do not allow defining the business process orchestration semantics of web services. For this purpose higher-level specifications are required to provide the functionality that supports and leverages web services and enables specifications for business process modelling and automation.

Figure 6 shows a sample business orchestration sequence linking the business activities we defined in Figure 3 from a buyer to seller. Here we assume that activities within the

```
<BusinessProcessOrchestration name="Request Purchase Order"
  nameID="RequestPurchaseOrder_BC" initiatingPartner="BuyerId">
   <serviceLinkType name="BuyerSellerLink"
          xmlns ="http://schemas.xmlsoap.or/ws/..">
       <role name="Buyer" portType name="BuyerPortType"/>
       <role name="Seller" portType name="SellerPortType"/>
   /serviceLinkType>

  <Start toBusinessState="Request Purchase Order" />
    <sequence>
       nameID="RequestPurchaseOrder_BTA"
       businessTransaction="Request Purchase Order"
       businessTransactionIDRef="RequestPurchaseOrder_BT"
       fromRole="Buyer" fromRoleIDRef="BuyerId"
       toRole="Seller" toRoleIDRef="SellerId"
       timeToPerform="... "
    </sequence>


  </BusinessProcessOrchestration >
```

Figure 6. Example of a simple business orchestration specification.

business orchestration are conducted in a sequential manner. Again we use a free syntax based on XML for reasons of simplicity.

BPEL has recently emerged as the standard to define and manage business process activities and business interaction protocols comprising collaborating web services. This XML-based flow language defines how business processes interact. Enterprises can describe complex processes that include multiple organisations — such as order processing, lead management, and claims handling — and execute the same business processes in systems from other vendors. BPEL combines the former IBM WSFL and Microsoft XLANG efforts.

Business processes specified in BPEL are fully executable portable scripts that can be interpreted by business process engines in BPEL-conformant environments. At the core of the BPEL process model is the notion of peer-to-peer interaction between services described in WSDL. Both the interacting process as well as its counterparts are modelled in the form of WSDL services. BPEL provides a mechanism for creating implementation and platform independent compositions of services weaved strictly from the abstract interfaces provided in the WSDL definitions. Actual implementations of the services themselves may be dynamically bound to the partners of a BPEL composition, without affecting the composition's definition. A BPEL business process can interoperate with the web services of its counterparts, irrespectively whether these are implemented on the basis of BPEL.

A BPEL process is a flow-chart-like expression of an algorithm defining process steps and entry-points into the process that is layered on top of WSDL [20]. BPEL uses WSDL to specify activities that should take place in a business process and describes the web services provided by the business process. The role of BPEL is to define a new web service by composing a set of existing services. The entry-points correspond to external clients

invoking either input-only or input-output (request-response) operations on the interface of the composite service. We distinguish four main sections in BPEL: the message flow, the control flow, the data flow and the process orchestration sections.

Each step in the process is called an activity. Each activity is implemented as an interaction with a web service provided either by a particular provider or by one of its business partners. The *message flow* section of BPEL is handled by primitive activities that include invoking an operation on some web service (⟨invoke⟩), waiting for a process operation to be invoked by some external client (⟨receive⟩), and generating the response of an input-output operation (⟨reply⟩).

Primitive activities can be combined into more complex algorithms using the structured activities provided by the language. These are used for control flow purposes. The *control flow* section of BPEL is a hybrid model half-way between block structured and state transition control flow definitions. The control flow part of BPEL includes the ability to define an ordered sequence of steps (⟨sequence⟩), to have branching (⟨switch⟩), to define a loop (⟨while⟩), and to execute one of several alternative paths (⟨pick⟩). It main structured activity is the ⟨flow⟩ statement that allows defining sets of activities (including other flow activities) that are connected via ⟨links⟩. Thus a flow activity in BPEL may create a set of concurrent activities directly nested within it. It also enables expressing synchronisation dependencies between activities that are nested directly or indirectly within it. Within activities executing in parallel, execution order constraints can be specified. All structured activities can be recursively combined. The ⟨link⟩ construct is used to express these synchronisation dependencies providing among other things the potential for parallel execution of parts of the flow. The links are defined inside the flow and are used to connect exactly one source activity to exactly one target activity. A ⟨link⟩ may be associated with a transition condition, which is a Boolean expression using values in the different data containers in a process.

The *data flow* section of BPEL requires that information is passed between the different activities in an implicit way through the sharing of globally visible data containers. Data containers specify the business context of a particular process. These are collections of WSDL messages, which represent data that is important for the correct execution of the business process, e.g., for routing decisions to be made or for constructing messages that need to be sent to partners. Containers may exchange specific data elements via the use of ⟨assign⟩ statements. The ⟨assign⟩ statement allows not only data manipulation, but also dynamic binding to different service implementations. Currently, there is no capability provided by BPEL to specify and execute transformations on the data.

As a language for composing web services, BPEL processes interact by making invocations to other services and receiving invocations from clients. The prior is done using the ⟨invoke⟩ activity, and the latter using the ⟨receive⟩ and ⟨reply⟩ activities. BPEL calls these other services partners. A partner is a web service that the process invokes and/or any client that invokes the process.

The *process orchestration* section of BPEL uses service links to establish peer-to-peer partner relationships. Partners are connected to a process in a bilateral manner using a service link type. Service-links define the shape of a relationship with a partner by specifying the WSDL ⟨message⟩ and ⟨port-types⟩ constructs used in the interactions in both

directions. A service link type is a third party declaration of a relationship between two or more services, comprising a set of roles, where each role indicates a list of port types. A BPEL partner is then defined to play a role from a given service link type.

Clients of the process are treated as partners, because a process may need to invoke operations on clients (for example, in asynchronous interactions), and because the service offered by the process may be used (wholly or in parts) by more than one client [20]. For example, a process representing a loan servicing system may offer a single web service, but only parts of it are accessible to the customer applying for the loan, while other parts are accessible for the customer service representative, and the entire service is accessible to the loan underwriters. The approach of using partners to model clients allows the process to indicate that certain operations may only be invoked by certain clients.

Overall there are many distinct similarities between BPEL and the BPML initiative proposed by BEA, SAP and Sun Microsystems. We summarise this discussion by examining how the BPEL specification addresses transactional properties, and business protocols that are part of BTF.

- Business transaction and coordination protocol support: Once the business process and connections with business customers, partners, and internal entities is defined using BPEL, the next step is to provide a mechanism to coordinate and integrate the distinct web service activities occurring within that business process so that a dependable outcome can ensue. In a process specification there may be several distinct activities that must be successfully completed, some of which may run simultaneously and the process may need to make sure that they are all completed in a transactional manner. BPEL itself does not provide transactional and coordination mechanisms, however, it is complemented by WS-Coordination [6] and WS-Transaction [7] that provide such mechanisms.

  BPEL supports WS-Coordination and WS-Transaction by providing compensation mechanisms that allow the process designer to implement compensating actions for certain irreversible activities. For example, during the execution of a travel reservation process (that includes trip, hotel and car reservation) the designer may specify compensating activities to cancel parts of or the entire reservation. BPEL also include fault handlers that catch exceptions that occur within the scope of grouped (nested) activities that share common characteristics.

  Activities performed within an atomic scope either all complete successfully or are all compensated. Long running transactions in BPEL are centred on scopes and scopes can be nested. While BPEL long-running transactions currently assume that a nested scope is contained in a single process and is hosted by a single BPEL engine, the agreement protocol in WS-Transaction does not assume this [21]. Thus, an expected extension of BPEL is the support of long-running transactions that are distributed across processes and even BPEL engines.
- Business protocol support: BPEL does not only specify executable processes it can also be used for specifying business protocols such as RosettaNet PIPs. In BPEL, the language for specifying business protocols is a subset of the language used for specifying executable business processes.

A business protocol may be perceived in BPEL as an abstract view on a non-executable private process. For example, in a supply-chain protocol, the buyer and the seller are two distinct roles, each with its own abstract process. Their relationship is typically modelled as a service link. Abstract processes use all the concepts of BPEL but approach data handling in a way that reflects the level of abstraction required to describe public aspects of the business protocol. Specifically, abstract processes handle only protocol-relevant data [9]. BPEL provides a way to identify protocol-relevant data as message properties.

4. Business transaction framework

In what follows we shall present and analyse several protocols that contribute towards the requirements and functionality of the BTF described in Section 1.2. In particular, we shall examine to which extend business transaction and coordination (BTCP) and business protocols support the BT phases. As currently there are virtually no research results regarding the post-transaction phase we will chiefly focus our attention on the pre- and main transaction phases.

Several approaches have been in the context of Internet and Web transactions that address some of the requirements of the BTF. These include among other correlation mechanisms for managing multiple service conversations [31], protocols for conversational transactions [24], WSDL extensions to describe implicit transactional semantics found in business applications [22] and extensions of the two-phase commit protocol with a transactional messaging infrastructure [34].

One of the earliest attempts to define an Internet-based transaction protocol that simplifies the distributed applications programming is the Transaction Internet Protocol (TIP) [14]. The TIP deals with the deficiencies of conventional two-phase commit (2PC) protocols. These only offer transaction semantics to a limited set of applications, operating within a special purpose (complex and homogeneous) infrastructure and use a particular set of inter-communication protocols. Conventional 2PC protocols employ a one-pipe model whereby the transaction protocol flows over the same conduit (pipe) as the as the application to application communications. This renders them complex and hard to implement, thereby inhibiting the development of distributed business applications. TIP is a simple two-phase commit protocol that removes the restrictions of conventional 2PC protocols by providing ubiquitous distributed transaction support in a heterogeneous and cross-domain environment. This is made possible by employing a *two-pipe* model separating the transaction protocol from the application communications protocol.

Although the TIP offers flexibility for 2PC protocol-based short-lived transactions, it falls short in the case of long-lived business transactions. Business transactions consist of a number of component transactions with largely different response times, thus blocking resources controlled by short-lived transactions for unacceptably long periods of time, rendering them unable to process new service requests. This is an undesirable option from an autonomous service provider's point of view.

Despite its limitations, the TIP has made an important contribution: it recognises the need to separate application semantics from the main transaction protocol. In our view this ability is a paramount property for developing flexible e-business applications.

4.1. Application communication protocols

These protocols are the main component in support of the pre-transaction phase. Two important protocols fall within the premises of this category: the W3C Tentative Hold Protocol and the ebXML Collaboration Protocol Agreement (CPA). As the CPA is traditionally associated with business protocols we shall examine it in conjunction with this part of the BTF (see Section 4.3).

4.1.1. The Tentative Hold Protocol. The objective of Tentative Hold Protocol (THP) is an effort to facilitate automated coordination of multi-business transactions [30]. Tentative Hold Protocol is an open, loosely coupled messaging-based protocol for the exchange of information between trading partners prior to the actual transaction itself. This framework includes a standard way for trading partners to exchange tentative commitments to the terms of an interaction, e.g., price, quantity, delivery time, and so on, and update each other when the situation changes. These THP exchanges increase the chances of successful completion of the business interaction regardless of whether the interaction is coordinated using custom applications or standard BTF services such as compensating transactions.

THP defines an architecture that allows tentative, non-blocking holds or reservations to be requested for a business resource, e.g., the items for sale from an online retailer. In this example of online ordering, these lightweight reservations are placed prior to the sale, allowing multiple clients to place holds on the same item (thus non-blocking). Whenever one client completes the purchase of the item the other clients receive notifications that their holds are no longer valid. This provides flexibility for both ends of the transaction. The clients have the ability to request tentative reservations on the resources that they want to acquire as a single coordinated purchase, verifying price and availability before completing the transactions. If one item becomes unavailable (purchased by some other client), this client has the ability to replace it with another item without losing any holds on other items already reserved. The vendors grant non-blocking reservations on their products, retaining control of their resources, while allowing many potential customers greater flexibility in coordinating their purchases. THP, thus, enables customers to link to a wide variety of suppliers and effectively arbitrage orders to gain best pricing and delivery.

There is a THP coordinator on both the client and resource owner side, responsible for communicating various messages such as hold requests, cancellations, and so on. Additionally, a resource owner provides a business rules engine with which the resource side THP coordinator communicates in order to handle business rule specific actions. This gives the resource owner the possibility of providing targeted customer service with the granting of holds, specifying greater or lesser hold expirations for a given hold request, as well as the potential for notifying valued clients when some resource is being reserved by another client — allowing the preferred client the opportunity to lock in their purchase first.

Tentative Hold Protocol and compensating transactions. The management of cross-organisation business transactions often involves the use of compensating transactions. Under normal circumstances individual steps in a business transaction are completed immediately. However, if one of the business transaction steps fails, the other steps in the transaction are undone using a compensating transaction. A new transaction is issued to compensate for the failed transaction, e.g., an order cancellation transaction would compensate for the order placement transaction. Compensating transactions allow for eventual consistency (rather than immediate consistency, as provided by the two-phase commit protocol) between parties involved in the transaction. Consistency is ensured only after any required compensations have completed.

Not all business transactions may be cleanly and completely reversible. Consider, for example, a situation where an item was purchased across international borders. Such a purchase could conceivably involve fluctuating currency exchange rates. A compensating transaction for a cancelled sale may not result in transferring the same amount of money should the exchange rate fluctuate between the original transaction and the compensating transaction.

Compensating transactions need to be kept to a minimum in e-business applications and this can be achieved by using the THP.

Adding a tentative hold phase for business application elements provides the following benefits [30]:

- Minimises compensations. Consider, for instance, a client application which attempts to coordinate the purchase of an item X from company A and item Y from company B. Without THP, the client application would most likely place an order for item X from company A, and then try to purchase item Y. If the application is unable to purchase item Y, then it would compensate for its earlier action in its business logic by cancelling the order with company A. However, if tentative holds were used, the client application would place tentative holds on both the items thus ensuring their availability, and thereafter complete both purchases. In this case, the application is far less likely to cancel, having determined that both X and Y are available at an acceptable price before placing any orders.
- Reduces lag between original transaction and compensation. From the previous example, it is clear that the lag time between a purchase and a compensating transaction would be reduced as the client application acquires tentative commitments from all potential business partners before it issues any actual business transactions with any one of them.

Tentative Hold Protocol and Two Phase Commit. The standard method for achieving the ACID properties in short-running transactions involving multiple (possibly distributed) resources is the two-phase commit protocol. This protocol ensures agreement by all parties regarding the outcome of the work (all-or-nothing). Two-phase commit protocols are best suited for a single-domain distributed applications. Since two-phase commit assumes the participating resources to be protected (locked), possibly for long times, it is not suitable for long running B2B transactions, where resources are managed by systems that belong to separate organisations. However, 2PC can be used for small scale interactions, i.e., atomic transactions, where resource locks can be tolerated.

Introducing a tentative hold phase prior to the 2PC protocol — for small scale interactions, i.e., atomic transactions, where resource locks can be tolerated — provides the following benefits [30]:

- Shortens the required 2PC lock duration by minimizing the time spent in the prepare phase since it allows applications to obtain tentative commits and make all decisions before they enter the prepare phase of a 2PC commit transaction.
- Minimises the likelihood of rollbacks since it allows the customers and vendors to exchange information on the terms they could commit to, e.g., price and quantity, and keep each other up-to-date about any change in status through proactive notifications.

4.2. Pure business transaction and coordination protocols

To address the shortcomings of the TIP and at the same time provide support for business web service enabled transactions, IBM and Microsoft have introduced the WS-Transaction and WS-Coordination specifications while OASIS has introduced the concept of the Business Transaction Protocol (BTP) [23]. These parallel initiatives were designed to provide a basic framework for providing transactional coordination of participants of services offered by multiple autonomous organisations that use WSDL enabled web services to exchange data and orchestrate processes. We call these protocols "pure" BTCPs as their constructs do not support the requirements of the pre- or post-transaction phase.

As at the time of this writing the WS-Transaction and WS-Coordination are still under development, we shall first introduce them briefly and then concentrate on the BTP.

4.2.1. The WS-Coordination and WS-Transaction specifications. The WS-Coordination and WS-Transaction specification complement BPEL in that they provide a web services-based approach to improving the dependability of automated long-running business transactions in an extensible, interoperable way. WS-Coordination provides a framework for coordinating the actions of distributed applications via context sharing. WS-Transaction provides standards for atomic transactions as well as long-running transactions.

The WS-Coordination specification. The WS-Coordination specification describes an extensible framework for providing protocols that coordinate the actions of distributed applications. Such coordination protocols are used to support a number of applications, including those that need to reach consistent agreement on the outcome of distributed transactions. The WS-Coordination framework supports the notion of pluggable coordination models, similar to the work on frameworks for extended distributed object transactions (the OMG/J2EE Activity service). Each specific coordination model is represented as a coordination type supporting a set of *coordination protocols*; a coordination protocol is the set of well-defined messages that are exchanged between the participants [20]. Sample protocols include completion protocols, the two-phase commit protocol, synchronization protocols, or outcome notification protocols.

WS-Coordination provides developers with a way to manage the operations related to a business activity. A business process may involve a number of web services working together to provide a common solution. Each service needs to be able to coordinate its activities with those of the other services for the process to succeed. WS-Coordination sequences operations in a process that spans interoperable web services to reach an agreement on the overall outcome of the business process. The specification supplies standard mechanisms to create an activity via an *activation service* and register via a *registration service* with transaction protocols that coordinate the execution of distributed operations in a web services environment. The WS-coordination provides a definition of the structure of activity context and the requirements for propagating this context between cooperating services via its *coordination context*.

A coordination context is created via the activation service when a client specifying a desired coordination type sends a request message to begin an activity. The coordination context includes information about the type of the activity, i.e., atomic or business activity, a global identifier, expiration data, a port reference for the registration service, and protocol-specific extensions. The port reference is a WSDL definition type that is used to identify an individual port. The port type reference consists of the URI of the target port as well as contextual information that may include service-specific instance data.

Whenever the client initiates a business invocation on some web service, the coordination context is implicitly propagated along with the application message. The service that is being invoked can then find out about the registration service's port reference to register for the coordination protocol that it wishes to participate in.

The WS-Transaction specification. The WS-T specification leverages WS-Coordination by extending it to define specific protocols for transaction processing. More specifically, it describes the coordination types that are used with the extensible coordination framework outlined in the WS-Coordination specification. It defines two coordination types: *atomic transaction* and *business activity*. These are similar in nature to the atomic and business transactions we defined in Section 2.1. Atomic transactions are suggested for transactions that are short-lived atomic units of work within a trust domain, while business activities are suggested for transactions that are long-lived units of work comprising activities of potentially different trust domains.

Atomic transactions compare to the traditional distributed transaction model, where, for example, resource participants register for the two-phase commit protocol. The coordination type correspondingly comprises protocols common to atomic transactions, including the two-phase commit protocol.

The business activity coordination type supports transactional coordination of potentially long-lived activities. These differ from atomic transactions in that they take much longer time to complete and do not require resources to be held. To minimise the latency of access by other potential users of the resources used by an activity, the results of interim operations need to be realised prior to completing the overall activity. They also require that business logic be applied to handle exceptions. Participants are viewed as business tasks (scopes) that are children of the business activity for which they register. Participants may decide to leave a business activity (for example, to delegate process-

ing to other services), or, a participant may declare its outcome before being solicited to do so.

Developers can use either or both of these coordination types when building applications that require consistent agreement on the outcome of distributed activities. The WS-Transaction specification monitors the success of specific, coordinated activities in a business process. WS-Transaction uses the structure that WS-Coordination provides to make sure the participating web services end the business process with a shared understanding of its outcome. For example, a travel reservation process contains various activities that have to complete successfully but might run simultaneously, such as airline ticketing, car rental, and hotel room booking. The combination of WS-Transaction and WS-Coordination makes sure that these tasks succeed or fail as a unit.

4.2.2. The OASIS Business Transaction Protocol. The Business Transaction Protocol, BTP, is a Committee Specification of the Organisation for the Advancement of Structured Information Standards (OASIS) [23]. BTP is a core-level transaction protocol that contains a set of specific messages that get exchanged between systems supporting a business application in order to achieve interoperability between web service enabled transactions. The BTP should be implemented on the sites of all trading partners.

There are two noteworthy requirements underpinning the development of the BTP [38]:

- Support discontinuous service, where web services are anticipated to suffer outages during their lifetimes, and ensure that any work being undertaken survives those outages. This means that we must allow for failures — e.g., failures due to business rules such as trying to purchase an out of stock item, and system/network failures — and allow participants in a transaction plenty of leeway to leave that transaction.
- Interoperation of multiple communication protocols using XML. Whatever the wire-level transport, messages always arrive at BTP-aware services in XML. This obviously suggests that web services are in the frame here since they only accept XML messages.

BTP is based on two-phase commit for small-scale (short duration) interactions known as *atoms*, which can be aggregated into larger non-ACID transactions known as *cohesions* [38]. BTP atoms posses full ACID properties and are coordinated by a standard two-phase commit protocol. They differ from the atomic transactions introduced in Section 2 in that they are not nested. The BTP cohesion is a flexible transaction that aggregates atoms and is used to relax atomicity by allowing for the selection of work to be confirmed or cancelled based on higher level business rules. In that respect cohesions have the same characteristics as the BTs we introduced in Section 2. The strength of BTP lies in the fact that it permits the composition of atomic units of work (atoms) into cohesive business transactions (cohesions) that allow application intervention into the selection of these atoms that will be confirmed and of those which will be cancelled. Cohesions themselves can also be aggregated.

Business transaction actors. To coordinate interaction the BTP defines the roles that trading partner applications may perform in a BTP-based interaction. It also introduces the messages, e.g., enrol, resign, prepare, etc, that pass between such actors and the obliga-

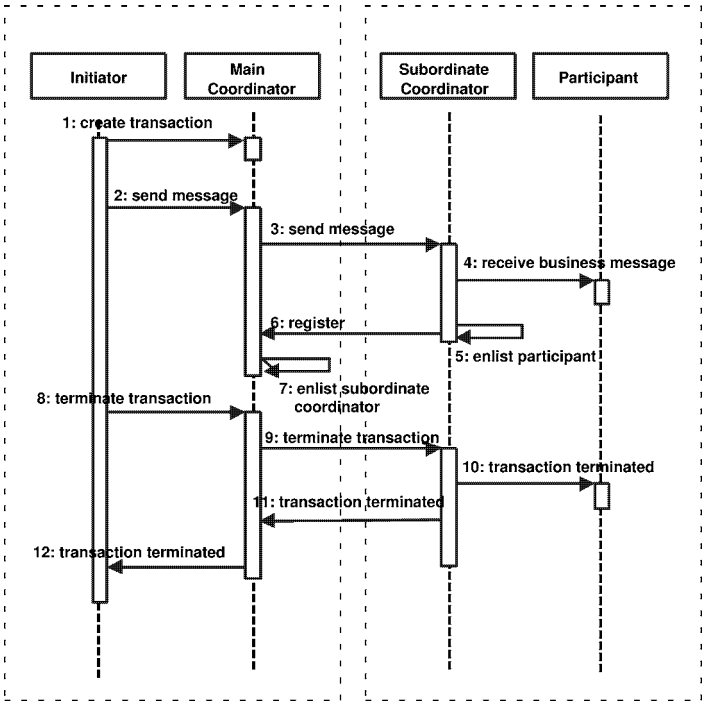

Figure 7. Business transaction actors and life-cycle.

tions upon and commitments made by actors-in-roles. There are a variety of roles used in the BTP specification, see Figure 7. These are as follows:

1. A transaction is always initiated by an application of a trading partner (*initiator*). The initiator sends application messages to a web service in order to invoke its operations.
2. A *transaction coordinator* is a software component that implements the BTP and can decide about the outcome of a single atomic transaction. It enlists and de-lists participants in a transaction and participates in the transaction execution protocol. A coordinator instructs participants to prepare, vote, and/or confirm. A transaction participant returns a successful message to a prepare instruction if it is capable of confirming or cancelling the set of operations it participates in an atomic transaction, i.e., first phase of a 2PC protocol. The cancel message is essentially the rollback operation of an atomic transaction. The confirm message instructs participants to make their current set of operations permanent, i.e., commit. The coordinator makes its decisions based on input from BT participants and the BT initiator. As BTs are nested there might be multiple coordinators in a BT. These can play the role of a main or subordinate coordinator. There is only one main coordinator in a BT. The main coordinator is normally the one that receives the createTransaction request from the initiator of the transaction. However, in the hub-and-spoke topology, normally employed by e-marketplaces, messages are exchanged via an intermediary, the hub. In this scenario the transaction coordinator

in the hub is always the main coordinator. The main coordinator drives the execution/termination protocol for that BT. The subordinate coordinators cooperate with the main coordinator for terminating a transaction with success, failure or timeout.

3. The applications of trading partners that take part in a transaction are called *participants*. These are capable of executing prepare, cancel and/or confirm operations issued by a coordinator. A participant is capable of sending vote and prepare messages to a coordinator. A participant sends a vote message to a coordinator usually in response to a prepare message. A participant can vote to cancel, ready, ready with inability to cancel after timeout, or ready with cancel after timeout. An enrol message is sent from participant to a coordinator when a participant has a set of operations that it wants to participate in a service atom. A resign message is sent from a participant to a coordinator to indicate that the service should no longer be part of this atomic transaction. These signals are used by the application(s) to determine whether to confirm or cancel the results of application operations. A participant must have a BTP address, to which BTP messages from the coordinator are delivered. A participant is responsible for executing the BT termination protocol. For example, if a confirm command is sent, then the results of all operations of the services contained within the participant must be made permanent. However, if a cancel command is sent then the actions of the services are undone.

Since multiple application components and resources participate in a BT, it is necessary for the transaction coordinator to establish and maintain the state of the BT as it occurs. This is achieved by using a transaction context, which is a data structure propagated onto messages passed between the BT initiator and the services within a business transaction. The transaction context specifies the type of a transaction atomic transaction or cohesion and identifies the initiator as a *superior* — containing both addressing information and the identification of the relevant state information. The context also indicates whether this superior will behave atomically or cohesively with respect to its inferiors[4]. Participants in a BT become context-aware by the transaction via receiving a message that carries the transaction context. Initially only the initiator is context-aware. As the initiator sends messages to other participants they become context-aware. In turn, context-aware participants can propagate the transaction context to others by sending appropriate messages to them.

The propagation of the business transaction from one party to another, to establish the *superior*:*inferior* relationships involves the transmission of the transaction context. This is shown in Figure 7. This is in association with one or more application messages between the trading parties. When a message reaches a trading partner's server, the (subordinate) coordinator at the trading partner server intercepts that message and extracts the transaction context from it, see Figure 7. If the transaction is unknown to this coordinator, it makes a note of the transaction by storing the transaction context. The message is then delivered to the recipient trading partner's application for further processing. After delivering the message, the (subordinate) coordinator enlists the recipient trading partner's application as a *participant* in the transaction, see Figure 7. At this moment, this coordinator becomes a *subordinate coordinator* for this transaction. It then notifies the *main coordinator* of its involvement in the transaction by sending a *register* request. The main coordinator then adds this coordinator as a subordinate coordinator for this transaction.

The initiator is the only participant that is allowed to terminate the transaction. In order to terminate the transaction, the initiator sends a terminate request to the main coordinator. Then the main coordinator together with all the subordinate coordinators jointly executes the termination protocol. A transaction can be terminated with success or with error. Transaction termination with error triggers the appropriate compensating transaction at the participating trading partners' server.

The BTP participants enable the consistent reversal of the effects of atomic transactions. BTP participants may use recorded before- or after-images, or compensation operations to provide the "roll-forward, roll-back" capacity, which enables their subordination to the overall outcome of an atomic business transaction.

The cohesion protocol. BTP is designed to allow coordination of application work between multiple participants owned or controlled by autonomous organisations. BTP uses a two-phase outcome coordination protocol to ensure the overall application achieves a consistent result. BTP permits the consistent outcome to be defined *a priori* — all the work is confirmed or none is — (an atomic business transaction or atom) or for application intervention into the selection of the work to be confirmed (a cohesive business transaction or cohesion) [23]. The distinction between atomic and cohesive behaviour is whether the superior will choose or allow some inferiors to cancel while others confirm - this is not allowed for atomic behaviour, in which all must confirm or all must cancel, but is for cohesive. The method of reversal for an atom is specific to the participating service. It could be a classic roll-back of state or a compensating transaction. The BTP only demands that the ACID properties are maintained.

Once the unit of work within a transaction is about to terminate, a two-phase commit algorithm is executed to make certain that a consistent set of results is made durable across each of the components involved in the transaction (see Figure 8). The two-phase commit algorithm comprises the following steps [23]:

- The transaction coordinator decides to finish the transaction in which case it needs to determine whether the BT participants are able either to confirm or cancel their respective operations by sending them prepare to commit messages — that is whether they are prepared to make their work durable. When a prepare message is issued against an atom it instructs the atom to prepare its associated participating services when the transaction is to terminate. When it is issued against a cohesion it instructs the cohesive transaction to prepare one or more of its participating services at transaction termination time.
- The participants report their ability to confirm-or-cancel (their preparedness) to the coordinating entity. After receiving these reports, the coordinating entity:

    1. Determines which component should be instructed to confirm and which should be instructed to cancel. The confirm statement is essentially a commit statement for an atom. It instructs an atom to confirm all of its participating services. It also confirms all participant services that voted to confirm in the case of a cohesive transaction. The cancel statement instructs all participating services in an atom, or the services specified in the parameters of a cohesion to cancel.

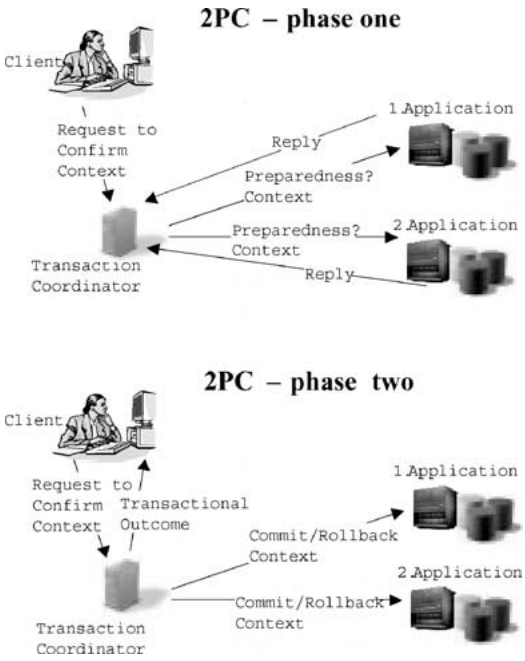

Figure 8. The relaxed two-phase commit protocol for cohesions.

2. Informs each component whether it should confirm or cancel by sending a message to its BTP element.
3. The coordinating entity returns a set of results to its superior entity (the client application). The client application may decide to confirm the cohesion even in the case that some of the atoms have chosen to cancel rather than confirm.

The two-phases of the BTP protocol ensure that either the entire attempted transaction is abandoned or a consistent set of participants is confirmed. In contrast to this, in most standard 2PC-based systems the coordinating entity automatically commits (confirms) if all the participants vote ready. The BTP deliberately hands the decision up to the initiator (client) application. This allows the initiator to fully control both phases of the two-phase commit protocol and make complex decisions about the outcome of the atoms and cohesions. These decisions are based on business rules and other (application-related) service execution outcomes.

In order to guarantee consensus, the standard 2PC-protocol is necessarily a blocking protocol. After the prepare phase response, each participant that returned a commit response must remain blocked until it has received the coordinators phase two message telling it what to do. Until they receive this message, any resources used by the participant are unavailable for use by other atomic transactions, since to do so may result in non-ACID behaviour. In contrast to this, the BTP 2PC variant is a consensus protocol between parties and does not require two-phase locking (within the participants).

The following code fragment taken from [17] illustrates how a cohesion might look to a client transaction API in Java (initiator). Note an API like this is not part of BTP specification, the code is given for illustrative purposes only.

```
void cohesionComposer() // an application method {
  Atom orderGoods = new Atom();
  Atom shippingViaGoodsSupplier = new Atom();
  Atom shippingFromAnotherSource = newAtom();
  // application work
  Quote quoteForGoods =
     orderGoods.sendApplicationMessage ("quoteForGoods", arg, arg ...);
  Quote quoteForShippingViaGoodsSupplier =
     orderGoods.sendApplicationMessage ("quoteForShipping", arg, arg ...);
  Quote quoteForShippingFromAnotherSource =
     orderGoods.sendApplicationMessage ("quoteForShipping", arg, arg ...);
  // ensure that the quotes are guaranteed (may be folded into app messages)
  orderGoods.prepare();                // no exception, so it is ready
  shippingViaGoodsSupplier.prepare();  // ditto
  shippingFromAnotherSource.prepare(); // ditto
  orderGoods.confirm();
  QuotesOutcome quotesOutcome = this.decideQuotesOutcome
                              (quoteForShippingViaGoodsSupplier,
                               quoteForShippingFromAnotherSource);
  quotesOutcome.selected().confirm();
  quotesOutcome.rejected().cancel();
}
```

4.2.3. Business Transaction and Coordination Protocols and the BTF. The protocols outlined in this section help implement the elements of the BTF and characteristics of business transactions as described in Sections 1.2 and 2.

- *Business transaction and coordination protocol support.* Both the WS-Coordination and WS-Transaction specifications and the BTP are built of top of web services environment to provide a general framework for how business processes can be implemented. They enable persistent and reliable processing of distributed activities and provide support for a broad range of transactional and non-transactional processes. They are also extensible and customisable so that they can meet evolving business requirements.
  No support for atomicity types is provided other than service request atomicity. Although neither the WS-Coordination and WS-Transaction specifications nor the BTP support the unconventional atomicity criteria and transaction phases we described in Section 2.3, they can both be extended to accommodate such functionality.
- *Business protocol support.* The WS-Coordination and WS-Transaction do not provide direct support for business protocols, however, when they are combined with BPEL they offer the basis for specifying a variety protocols.
  BTP is "agnostic" regarding its underlying e-business protocol stack so it can be easily implemented in conjunction with other standards, such as ebXML. For example, a header can be added to the ebXML message envelope to carry the transaction context defined by BTP.

4.3. Business protocols and transactional conversations

A business protocol is associated with a business processes and governs the exchange of business information and messages between trading partners across differing middleware platforms and organisations. It specifies the structure and semantics of business messages, how to process the messages, and how to route them to appropriate recipients. A business protocol may also specify the characteristics of messages related to persistence and reliability. A business protocol is bound to a business conversation definition and a delivery channel for a trading partner. The business protocol should also be able to specify collaboration or contractual agreements that exist between two trading partners and contain configuration information required for the partners to interoperate. A business protocol is indirectly bound to such a collaboration agreement through its associated conversation definition and associated trading partner delivery channel [27].

4.3.1. Business standards and protocols. Business standards and protocols typically manage the structure for defining form, fit and function of any product or service, regardless of the industry. In addition, they specify the structure and format and semantics of the business content of a message as well as the message exchanges (also referred to as *business conversation dialogue*) between trading partners. They use specialized business and technical dictionaries to provide commonly used domain-specific terminology and accepted values. To develop meaningful business standards and protocols, vendors have begun to work together, facilitated by standards bodies, and in unison with business experts to define global XML standards like ebXML (`ebXML.org`), RosettaNet (`www.rosettanet.org`), cXML (`cxml.org`) and others that enable the development of process-centric e-business applications.

In conjunction with these activities many industry standards groups are working on domain-specific XML and ebXML-based business protocols and standards in the domains of the Automotive Industry Action Group, the Open Travel Alliance (`www.opentravel.org`), the Association for Retail Technology Standards (`www.uccnet.org`), Health Care and Medical Equipment, and the ACORD standards group for the insurance industry [27].

RosettaNet. RosettaNet (`www.rosettanet.org`) is an independent, non-profit consortium of major IT, electronic component and semiconductor manufacturing companies dedicated to the collaborative development and rapid deployment of industry wide, open e-business process standards. These standards form a common e-business language, aligning processes between supply chain partners on global high-technology trading network. To supports its mission RosettaNet provides specifications for the RosettaNet Implementation Framework (RNIF), the RosettaNet Partner Interface Processes (PIPs), and business and technical dictionaries.

PIPs define business processes between trading partners. PIPs fit into seven groups of core business processes that represent the backbone of the trading network. Each group is broken down into segments — cross-enterprise processes involving more than one type of trading partner. PIPs are specialized system-to-system XML-based dialogues. Each PIP

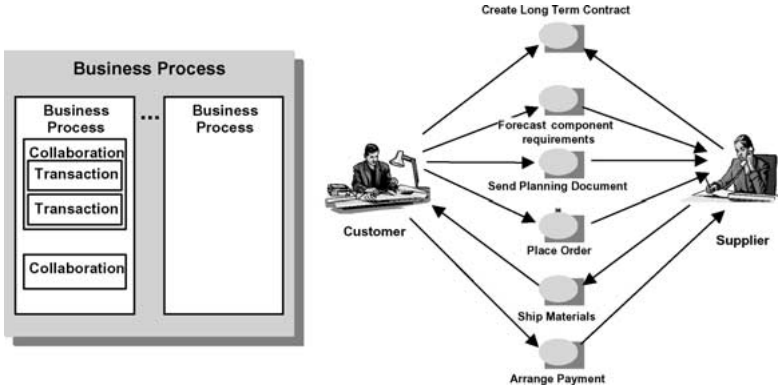

Figure 9. ebXML business processes and transactions.

specification includes a business document with the vocabulary, and a business process with the choreography of the message dialogue. Typical PIPs include order management, inventory management, marketing information management, service and support and manufacturing. The RNIF acts as the grammar and provides common exchange protocols while PIPs form the dialogue. The RNIF together with the dictionary, which provides a common set of properties for business transactions and products, form the basis for the implementation of RosettaNet PIPs.

ebXML. The vision of ebXML is to create a single global electronic marketplace where enterprises of any size and in any geographical location can meet and conduct business with each other through the exchange of XML based messages. To facilitate this, ebXML provides an infrastructure for data communication interoperability, a semantic framework for commercial interoperability, and a mechanism that allows enterprises to find, establish a relationship, and conduct business with each other. Once all of this is accomplished, enterprises can then exchange information and products/services according to these agreements.

The Business Process Specification Schema (BPSS) [5] is a relatively simple schema that describes public processes and aims to support the specification of business transactions and their choreography into business collaborations.

BPSS supports a long-running business transaction model based on proven e-business transaction patterns used by previous standards such as RosettaNet. Business transactions within the BPSS are applied to the semantic business level with a simplistic protocol defined for the interaction between two parties (requesting and responding) and determination of success or failure of the transaction. An *ebXML business transaction* represents business document flows between requesting and responding partners[5]. In any ebXML business transaction there always is a requesting business document, and optionally, a responding business document. Each business transaction request or response may require that a receipt acknowledgement be returned to the sender. For contract-forming transactions such as purchase order requests an acceptance acknowledgement may need to be

returned to the requester. Time constraints can be applied to the return of responses and acknowledgements. If an ebXML business transaction fails on either side, the other side is notified so that both sides can carry out any actions necessary to process the failure in their internal systems. Each business transaction can be implemented using one of many available standard UN/CEFACT Modelling Methodology (UMM) patterns, which determine the actual exchange of business documents and business signals between trading partners to achieve the required electronic business transaction.

An ebXML business transaction can be viewed as a type declaration, while business transaction activities (which reference a unique business transaction type) are the usage of this transaction within a particular choreography.

A BPSS *business collaboration* is essentially the specification of business transaction activities between the two partners, their associated document flow, and the choreography of these business transaction activities. BPSS describes public processes as collaborations between roles, with each role abstractly representing a trading partner. There are two types of collaborations: binary collaborations and multi-party collaborations. Multi-party collaborations are decomposed to binary collaborations. The business collaboration specifies all the business messages that are exchanged between two trading partners, their content, and their precise sequence and timing. This part of the "agreement" provides a shared understanding of the interaction. The way the contract is implemented is private but the expectations are public. An ebXML collaboration is conducted by two parties, each using a human or an automated business logic system that interprets the documents transmitted and decides how to (or whether to) respond. All collaborations are composed of combinations of *atomic transactions*, each between two parties. Multi-party arrangements must be decomposed into bilateral transactions. The sequencing rules contained in a collaboration definition are not between messages but between business transaction activities.

4.3.2. Business Conversations. In business applications it is expected that web services can communicate through conversation sequences. A conversation sequence is a long-running sequence of interactions, e.g., documents exchanges, between two or more interacting services [15]. Since the notion of conversation is fundamental to web services, the exportation of transactional properties should fit within the context of conversations, giving rise to transactional conversations [15]. For example, a master purchasing agreement, which permits the placing of orders for components by known buying organisations, allows a buyer and a seller to create and subsequently exchange meaningful information about the creation and processing of an order. Such agreements, commonly knows as TPAS (see Section 2.5), stem from business negotiations and are specific to a particular trading community, e.g., a vertical e-marketplace such as semiconductors, chemicals, travel industry, etc. Agreements usually result in the specification of shared or canonical data formats and of the messages that carry those formats, and their permitted sequences, all of which are needed for an automated implementation of an agreement.

A TPA relates to a conversation between the trading partners. This conversation is a unit of business under the TPA and there may be many conversations running concurrently between the partners. A conversation is represented as a set of related business transactions each of which are represented by an interchange of messages between the partners.

These messages may be interchanged synchronously or asynchronously and it is up to each trading partner's BTF infrastructure to ensure that the message exchanges adhere to the rules of the TPA. Each party is responsible for maintaining the correlation between messages within a conversation — in effect the conversation state. A party is also responsible for correctly invoking its own business process implementations upon receipt of a request message and generating an appropriate response that is returned to the other party.

One important element of business conversations is the ability to demarcate the parts of a conversation that are transactional. This can be achieved by using the conversation atomicity principle to demarcate the parts of the conversation that are characterised as transactional. At one extreme the whole conversation sequence may be considered transactional. However, as this is not practical, it is more likely that a conversation sequence may have multiple parts that we can view as transactional.

The objective of transactional conversations is to introduce a standard way of describing BT-based conversations and thereby support interactions between services. A conversation is a short- or long-lived series of business messages exchanged between trading partners. A conversation definition typically includes a unique conversation name and version, specifies the roles that the trading partners play in the conversation; it is linked to a specific business protocol (e.g., RosettaNet PIPs) and TPA; and typically references a business process management collaborative workflow template for each role.

Figure 10 depicts a hypothetical conversation between two trading parties which engage in a bidding procedure to settle on mutually acceptable prices. This conversation is assumed to take place during the BTF pre-transaction phase. The figure also shows that a successful conversation may lead to the formulation of a contractual agreement between the trading parties. The message choreography for a conversation is described in these collaborative workflow templates. Simple business conversations and roles can be defined and monitored using tools such as the BEA WebLogic suite [3].

More complex conversation sequences can also be defined on the basis of core constructs such as the ones we illustrated in Figure 10. For example, in [18] the authors propose that web-service conversations be nested and include conversation policies. As there is normally only a single state and transition to represent all messages related to multi-threaded negotiations, the authors of this publication also suggest the use of pre- and post-conditions to explicitly represent separate transitions for different messages.

Trading partner agreements and business conversations are supported by business protocols such the ebXML. An ebXML business collaboration specifies as part of a Collaboration Protocol Agreement (CPA) an "agreement" between two or more trading business partners [12]. The CPA is connected to business process specifications and their related business transactions (both expressed in BPSS). The ebXML working group has adopted the term CPA to mean the same as a TPA. A CPA is created between two trading partners who wish to interoperate. Accordingly, the CPA requires knowledge of each partner's technical capabilities. These capabilities include communication protocols (HTTP, SMTP, FTP etc.) the partners support, messaging service messaging protocols and requirements, security requirements that they place upon the message exchanges, interface requirements and links to the business processes they support. These technical capabilities are described along with contact information and business classification information in a Collaboration

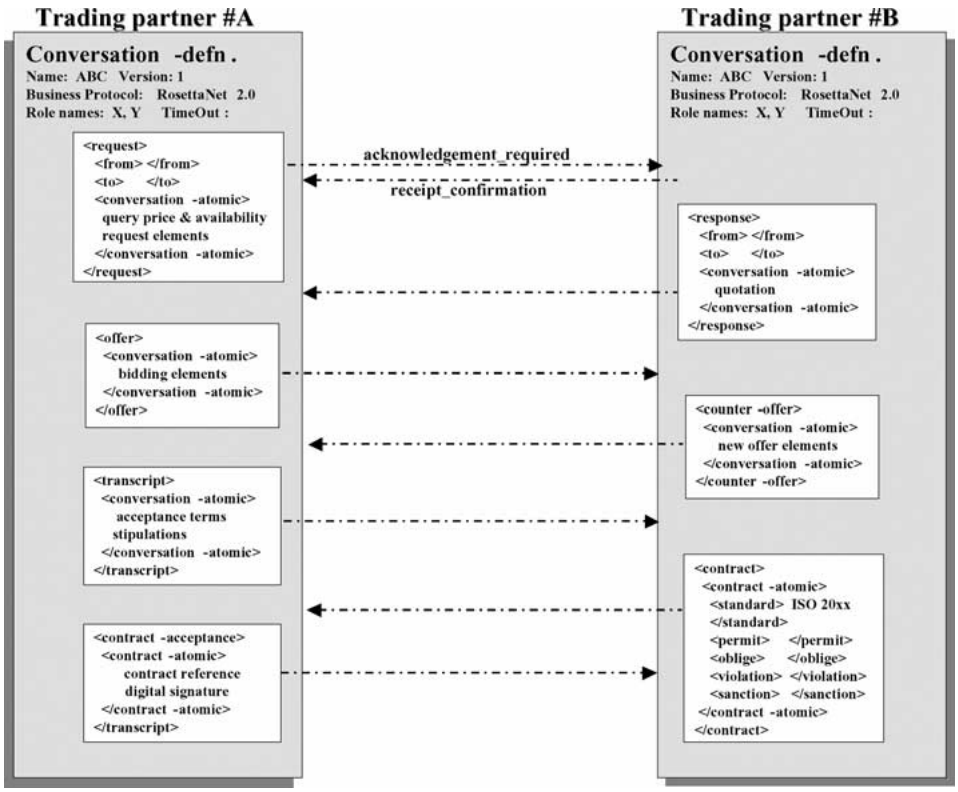

Figure 10. A hypothetical business conversation between trading partners.

Protocol Profile (CPP). Each party has a CPP that is a public description of their capabilities. The CPA is then an intersection of the two party's CPPs with the results of negotiation variable parameters in the profiles. The CPA includes an agreed common set of technical capabilities together with a definition of what business processes are to be performed between the two parties.

In Figure 11, party A and party B use their CPPs to jointly construct a single copy of a CPA by calculating the intersection of the information in their CPPs. The resulting CPA defines how the two parties will behave in performing their business collaboration. Thus, a CPA can contain the following key information [12,19]:

- *Overall Properties*. This section contains information relating to the overall CPA, e.g., the duration of the agreement.
- *Identification*. This section identifies the parties to the agreement. It includes nominated contacts with their addresses together with other party identifiers.
- *Communication Properties*. This section identifies what communication protocols the parties support. It also includes where necessary parameters such as timeouts needed to ensure interoperability between the parties.

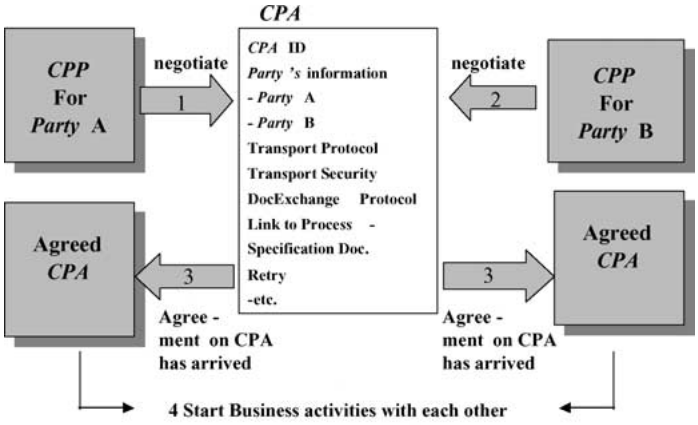

Figure 11. A CPA between trading partners.

- *Document Exchange Properties*. This section establishes the messaging protocol to be used in exchanging business documents. It includes a description of the set of messages that can be exchanged, their encoding, and any parameters associated with reliable delivery.
- *Security Properties*. This section includes information needed to ensure a secure interchange of information between the parties. It includes parameters for non-repudiation such as certificates, protocols, hash functions and signature algorithms. It also includes include digital certificates and encryption algorithms that can be used for security purposes.
- *Roles*. This section associates a party with a role defined in the CPP. In general, this description is defined in terms of roles such as "buyer" and "seller". The CPP identifies which role or roles a party is capable of playing in each collaboration protocol referenced by the CPP.
- *Business Transactions*. This section defines the business transactions or services that the parties agree to interchange. It describes the interfaces between the parties and the business application functions that actually perform the business transactions.
- *Comments*. This section is a free format section that can be used for additional information such as reference to any legal documents relating to the partner agreement.

Figure 12 shows how collaboration agreements (CPAs in BPSS), business conversations and trading partner representations are connected with each other. This figure shows that a collaboration agreement is governed by a business protocol that specifies the interactions between partner roles and the sequencing rules that determine the order of requests. The business protocol layer is the interface between the collaboration agreement actions and the business application functionality. When a given collaboration agreement can be reused for different pairs of parties, a template can be specified in terms of role parameters rather than specific party names. Roles are defined generic terms such as buyer and seller. The document exchange layer defines the properties exchanged by the trading parties [10]. Any document formats agreed to by the two parties may be used. The document exchange

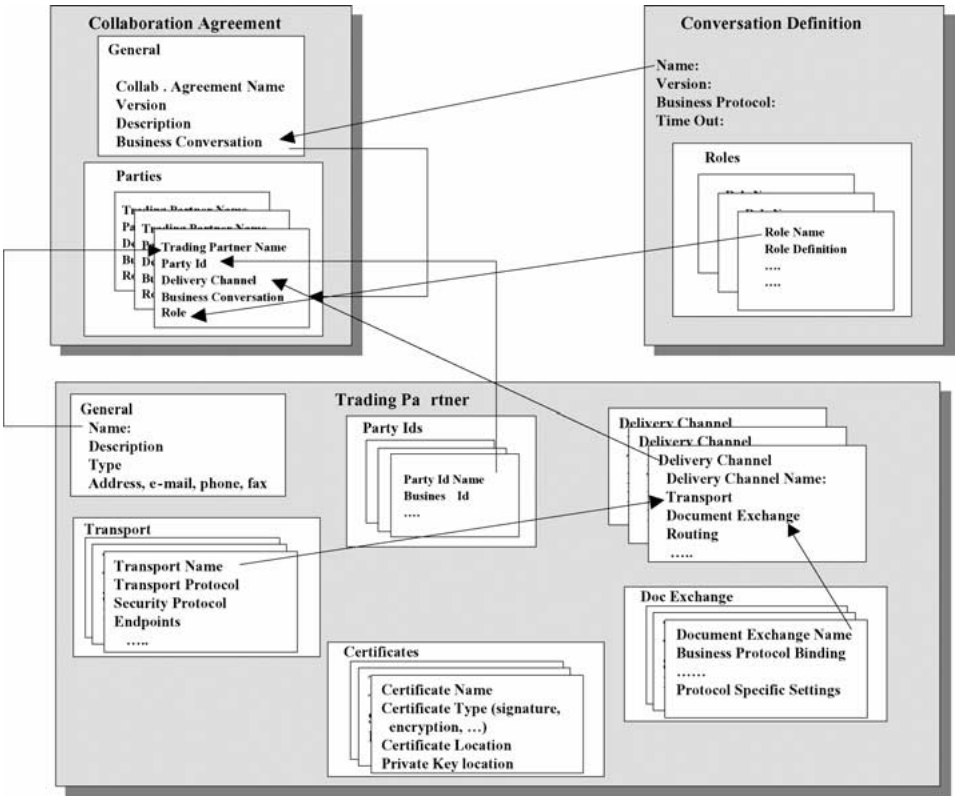

Figure 12. Connecting collaboration agreements, business conversations and trading partners.

layer accepts a document from the business protocol layer. It can encrypt, add a digital signature and pass the document to the transport layer for transmission to the other party. The transport layer is responsible for message delivery using selected communication and security, e.g., authentication and encryption, protocols. The communication properties define the system-to-system communication used between interacting parties. A delivery channel consists of one transport definition and one document exchange definition. It corresponds to a single communication and document-processing path. The collaboration agreement may include multiple transport and document exchange definitions, which can be grouped into delivery channels with different characteristics.

In a web services environment, trading partner agreements and business dialogues can also be supported by the Web Services Conversation Language (WSCL) [2]. The goal of WSCL is to provide the support infrastructure to allow for the definition of abstract interfaces of web services that provide business level conversations of business processes. WSCL specifies the XML documents being exchanged and the allowed sequencing of these document exchanges. WSCL may be used in conjunction with service description languages, such as WSDL to provide protocol binding information for abstract interfaces, or to specify the abstract interfaces supported by a concrete service. WSCL does not spec-

ify how the conversation participants will handle and produce the documents received and sent. The definition of a business conversation is thus service independent, and can be used by any number of services with completely different implementations. In this way a conversation developer, e.g., a vertical standards body, can create a WSCL description of the underlying elements of some conversation and ca publish in a UDDI directory. A service provider who wants to create a service that supports this conversation description elements could create and document service end points that support the message sequencing specified in the WSCL document [4].

4.3.3. Business protocols and the BTF. The ebXML BPSS clearly provides the necessary infrastructure and functionality to support the pre- and post-transaction phases of the BTP as well the ability to express desired business protocol functionality and business conversation interactions. However, it needs closer investigation to determine possibilities of utilization and leveragability between the ebXML business transactions and the BTF.

The following is a brief comparison of properties of business transactions defined in ebXML and business transaction frameowrks, such as the BTP, and is partly based on an analysis presented in [28] and [29].

- Business transaction and coordination protocol support:
  - ebXML business transactions are pre-defined reusable interactions that include one or two business documents exchange, and one or more signals that indicate the state changes in the transaction. ebXML transactions are not nested and there is no support for specifying compensating transactions. The ebXML business transactions are atomic in nature and thus comparable to the atomic transactions. No support for atomicity types is provided other than service request and non-repudiation atomicity. The BPSS provides a form of non-repudiation atomicity for message exchanges. For each request or response, it can be stipulated that the sender must save a copy of the message contents [29]. Additionally, it can be stipulated that a digitally signed receipt acknowledgement must be returned to the sender, who saves it. This provides a high degree of non-repudiation for ebXML transactions.
  - ebXML business transactions may be based on UMM transaction patterns. A transaction pattern is immaterial for BTF since coordination is governed by protocol not by a pattern of message exchanges.
  - The semantics of an ebXML transaction is enforced by the Business Service Interface (BSI). The BSI is any type of software application interface that implements business collaborations and manages the steps in a business transaction. With the BTF a service participating in a transaction enforces the semantics of functionality, while the protocol itself supports the coordination and recovery of the transaction.
  - There is no support for 2PC in ebXML transactions while the minimal requirement for BTF coordination purposes is the support of an extended 2PC protocol.
- Business protocol support. As already explained, BPSS process model can be referenced in an ebXML collaboration protocol agreement (CPA). This provides details on which roles the trading partner support in a specified process model in the context of some business agreement.

Where BTP could be applicable is where a "tighter," lower level coordination protocol (supporting a 2PC model for coordination of requestor and responder) is required for the transaction semantics between BPSS roles. More likely is that BTF could be leveraged to support multi-party collaborations when the ebXML BPSS looks into supporting this. The flexible coordination capabilities that the BTF offers could be used to prepare multiple BPSS binary collaborations and then decide to confirm only a subset — thus allowing ebXML to support not only atomic transactions for binary and multi-party collaborations but also BTF-like business transactions for multi-party collaborations.

5. Concluding remarks

Web services and businesses-to-business collaborative systems are becoming the predominant methods of creating business applications. Process oriented workflow systems and e-business applications require transactional support in order to orchestrate loosely coupled services into cohesive units of work and guarantee consistent and reliable execution.

In this paper we addressed the issues relating to the use of business transactions in web service based applications, introduced two forms of business transactions and presented a taxonomy of e-business transaction features such as unconventional atomicity criteria, the need to support business conversations and the need to clearly discern between three business transaction phases: the business applications communication protocol element, the main-transaction element and the post-transaction element. We also presented the elements of a flexible and extensible framework (Business Transaction Framework) that is necessary for building robust and extendible e-business applications. Subsequently, we introduced standard web service based initiatives — such as the BPEL, WS-Coordination, WS-Transaction and the Business Transaction Protocol (BTP) — and business protocol support activities, such as the Business Process Specification Schema (BPSS) of ebXML that enables the description of the public interface of e-business processes. Finally, we compared the functional characteristics of these standards and activities against those of the proposed business transaction framework.

Notes

1. These are also known as cohesions under the OASIS BTP specification [23], open transactions in the Business Process modelling Language (BPML) specification [1] and business activities in the WS-Transaction specification [7].
2. Although strictly speaking non-repudiation is a system-level primitive it makes more sense to consider it as part of the business interaction-level category.
3. We use the term transaction in this subsection to mean both atomic transaction or BT, unless otherwise stated.
4. The BTP distinguishes between superiors that treat their inferiors in an atomic or cohesive fashion. The former are called (atom) coordinators, while the latter are called (cohesion) composers. We use the term coordinator in the broad sense to encompass both types of superior behaviour.
5. ebXML business transactions should not be confused with BTF business transactions as the former are only simple atomic transactions.

References

[1] A. Arkin, "Business Process Modelling Language (BPML) specification," BPMI, http://www.bpmi.org/index.esp, June 2002.

[2] A. Banerji et al., "Web Services Conversation Language (WSCL) 1.0," W3C Note, March 2002, www.w3.ord/TR/wsc110/

[3] BEA Systems, "Introducing B2B integration," http://edocs.bea.com/wli/docs70/b2bintro/over.htm, 2002.

[4] D. Beringer, H. Kuno, and M. Lemon, "Using WSCL in a UDDI registry," Hewlett-Packard, May 2001.

[5] BPSS — "Business Process Specification Schema," ebXML Business Process Project Team, May 11, 2001, www.ebxml.org/specdrafts/cc_and_bp_document_overview_ver_1.01.pdf

[6] F. Cabrera et al., "Web Services Coordination (WS-Coordination)," August 2002, http://www.ibm.com/developerworks/library/ws-coor/

[7] F. Cabrera et al., "Web Services Transaction (WS-Transaction)," August 2002, http://www.ibm.com/developerworks/library/ws-transpec/

[8] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web Services Description Language (WSDL) 1.1," W3C, Note 15, 2001, www.w3.org/TR/wsdl

[9] F. Curbera, Y. Goland, J. Klein, F. Leyman, D. Roller, S. Thatte, and S. Weerawarana, "Business Process Execution Language for Web Services (BPEL4WS) 1.0," August 2002, http://www.ibm.com/developerworks/library/ws-bpel

[10] A. Dan et al., "Business-to-business integration with tpaML and a business-to-business framework," *IBM Systems Journal* 40(1), 2001.

[11] A. Drosopoulu and T. S. E. Maibaum, "Towards electronic contract performance," in *Proceedings of the 12th International Conference and Workshop on Database and Expert Systems Applications*, IEEE Computer Society Press, 2001.

[12] ebXML Collaboration Protocol Profile and Agreement Technical Committee, "Collaboration-protocol profile and agreement specification version 2.0," September 2002, www.oasis-open.org/committees/ebxml-cppa/documents/ebCPP-2.0.pdf

[13] A. Elmargarmid (Ed.), *Database Transaction Models for Advanced Applications*, Morgan Kaufmann, 1992.

[14] K. Evans, J. Klein, and J. Lyo, "Transaction Internet Protocol — requirements and supplemental information," 1998, www.landfield.com./rfcs/rfc2372.html

[15] S. Frolund and K. Govindarajan, "Transactional conversations," in *W3C Web Services Workshop*, July 2001, http://www.w3.orgt/2001/03/WSWS-popa

[16] J. Gray and A. Rueter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann, 1993.

[17] A. Green and P. Furniss, *Scope and Requirements, Actors and Terminology*, Choreology Ltd., May 2001.

[18] J. Hanson, P. Nandi, and D. Levine, "Conversation-enabled Web services for agents and e-business," in *International Conference on Internet Computing*, 2002.

[19] J. Ibbotson, "ebXML trading-partners specification," in *XML Europe 2001*, Brussels, May 2001.

[20] R. Khalaf, S. Tai, and S. Weerawarana, "Web services, the next step: A framework for robust service composition," *CACM*, Special Issue on Service-Oriented Computing, M.P. Papazoglou and D. Georgakopoulos, Eds., October 2003.

[21] F. Leymann and D. Roller, "A quick overview of BPEL4WS," IBM DeveloperWorks, August 2002, http://www-106.ibm.com/developerworks/

[22] T. Mikalsen, S. Tai, and I. Ravellou, "Transactional attitudes: Reliable composition of autonomous Web services," in *Workshop on Dependable Middleware Based Systems*, March 2002.

[23] OASIS Committee Specification, "Business Transaction Protocol," version 1.0, May 2002.

[24] J. Ouyang, A. Sahai, and V. Machiraju, "An approach to optimistic commit and transparent compensation for e-service transactions," HPL-2001-34, HP Laboratories Palo Alto, February 2001.

[25] M. P. Papazoglou, A. Delis, A. Bouguettaya, and M. Haghjoo, "Class library support for workflow environments and applications," *IEEE Transactions on Computer Systems* 46(6), June 1997.

[26] M. P. Papazoglou, A. Tsalgatidou, and J. Yang, *The Role of eServices and Transactions for Integrated Value Chains*, IDEA Publishers, 2001.

[27] M. P. Papazoglou and P. M. A. Ribbers, *Foundations of e-Business: Organisational and Technical Infrastructure*, Wiley, 2003.
[28] M. Potts and S. Temel, "Business transactions in workflow and business process management," OASIS Business Transactions Technical Committee Workflow Subcommittee, December 2001.
[29] D. O'Riordan, "Business process standards for Web services," in *Web Services Business Strategies and Architectures*, P. Fletcher and M. Waterhouse, Eds., Expert Press, 2002.
[30] J. Roberts and S. Krisnamurthy, "Tentative hold protocol," in *W3C Workshop on Web Services*, November 2001, `http://www.w3.org/TR/tenthold-1`
[31] A. Sahai, J. Ouyang, V. Machiraju, and K. Wurster, "End-to-end e-service transaction and conversation–management through distributed correlation," HPL-2000-145, HP Laboratories Palo Alto, September 2000.
[32] J. Snell, "Introducing the web services flow language," IBM Developer Works, June 2001, `http://www-106.ibm.com/developersworks/library/`
[33] "Simple Object Access Protocol (SOAP) 1.1," W3C Note, May 2000.
[34] S. Tai, T. Mikalsen, I. Rouvellou, and S. Sutton, "Dependency spheres: A global transaction context for distributed objects and messages," in *5th International Enterprise Distributed Object Computing Conference (EDOC)*, September 2001.
[35] S. Thatte, "XLANG — Web services for business process design," Microsoft Corporation, 2001, `http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm`
[36] J. D. Tygar, "Atomicity in electronic commerce," *ACM-Mixed Media*, April 1998.
[37] B. von Halle, *Business Rules Applied*, Wiley, 2002.
[38] J. Webber et al., "Making web services work," *Application Development Advisor*, November/December 2001, 68–71.
[39] J. Yang and M. Papazoglou, "Interoperation support for electronic business," *Communications of the ACM* 43(6), June 2000, 39–47.