A Financial Compensation based Transaction Management Model for Service-Oriented Business Collaborations

Zheng Yang*†, Jinhui Yao*†, Shiping Chen*†, David Levy*, Rafael Calvo*
*School of Electrical and Information Engineering, University of Sydney, Australia
†Information Engineering Laboratory, CSIRO ICT Centre, Australia

Abstract—The Internet has been encouraging and enabling business collaborations via online transactions over the Web. However, managing transactions in a long-running business process across domains still remains a challenge. In this paper, we propose a novel financial-compensation-based transaction management model, fcBTxM, to address this challenge. Unlike classical transaction management, fcBTxM does not attempt to recover data consistency via rollback when a failure occurs. Instead, our model always tries to forward-roll a business process via financial compensation. We use a state machine to capture and describe the states of our transaction model and their relationships. We also develop a set of technologies and protocols for enabling the new transaction management. A real business collaboration example is provided to demonstrate the concept, and preliminary testing results are provided to evaluate our technologies.

Index Terms—Business Transaction Management, Contract Compensation, e-Contract, Accountability, Service Compliance

I. INTRODUCTION

Classical data transaction management techniques have been widely adopted in commercial products. However extending the techniques in SOA environment to manage complex real-world business collaborations is still an open issue, due to:

- A service-oriented business process may involve multiple independent business entities (called participant) [18], who are collaborators and may be competitor as well. As a result, it is not feasible to put locks on to individual database for a transaction in such a long-running process.
- Behind each participant service may be complicated and heterogeneous IT infrastructures with different security and business policies [19]. Therefore, it is very difficult, if not impossible, to ensure the ACID of classical transactions within a global transaction across all partners.
- Even although each participant can recover their business data from a failed transaction, the classical transaction management technology provide no way to guarantee a recovery of the financial loss caused by the transaction failure. Besides, in real business practice, the collaborators usually expect to continue and complete a transaction, rather than rollback, even though partially with some failures, as long as with reasonable financial compensations.

Therefore, there is a need for a transaction management model to overcome the above mentioned challenges and coordinate the real-world business transactions.

In this paper, we propose a novel financial compensation based transaction management model, fcBTxM, to address this challenge. Differing from other transaction models, fcBTxM always tries to forward-roll a business process via financial compensation, instead of compensating data via roll-back, when a failure occurs. We present a state machine to capture and describe the states of our model and the relationships between the states. We define the protocols for enabling the new transaction management. We use a real business collaboration example to demonstrate the concept, followed by a system design to specify the implementation details.

II. RELATED WORK

Research over computer assisted transaction management is conducted in the community from various perspectives. Early research such as [10], [7], [8] and [9] addressed the transactional need raised from database community and gained significant successes. However, these work cannot be applied to SOA environment because of its unique characteristics such as cross-organizational and long-live [17].

In recent years a number of researchers try to solve the issue of transaction coordination via relaxation of one or more of the ACID attributes. i.e. [5] uses definable constraints to relax atomicity and avoid unnecessary recovery. An adaptive relaxation approach towards ACID was proposed in [11] by adding a ‘pre-commit’ phase and the decision is finally given by user intervention. S. Choi et.al. in [4], proposed an approach by analyzing dependency relationship among possible states of a transaction to maintain consistency but realize relaxation on isolation. That concept was further detailed and strengthened in [1] and a scheduler protocol was also provided to serialize the participant services in a transaction. WS-Business Activity [15], WS-Coordination [14] and DTP [23] are industrial standards but they do not have mechanism to handle financial-based compensation. In the meanwhile, there are some other approaches tend to do forward recovery to

DOI 10.1109/ICWS.2012.117
heighten the level of fault-tolerance thereby to improve overall throughput of transactions. For example, a forward recovery strategy for process fragments was suggested in [6]. However, all these approaches only focus on data compensation or data dependency when a transaction fails. As such, they have no capability of in managing real world business transactions, of which the fact is that when a business activity (BA) is done, it cannot be reversed without cost.

M. Papazoglou in [17] described the need of a business transaction framework and its requirement of transactional support. A conceptual business transaction model is introduced which supports the definition, execution and monitoring of long running processes that coordinate the activities of multiple business applications. In [18] a business transaction model is further proposed together with a Business Transaction Model Language (BTML) which is used to present business transactions. ATC (Abstract Transaction Construct) is given in [20], which is a contract-driven business transaction framework. The paper focuses on reusing existing transaction models in composition phase to facilitate business transaction management. Though none of the above addressed the criticality of financial compensation in coordinating real-world business transactions, they did inspire our work on fcBTxM.

A critical challenge in modelling fcBTxM is the definition of e-Contracts. Extensive studies on e-contracting have been done in the recent years. ECA (Event-Condition-Action), a three-layer architecture for e-contract enforcement designed in [3], shares a similar idea of our model however our focus are different. ebXML CPPA(Collaboration Protocol Profile and Agreement) [22] is a standard definitions provided by OASIS for the sets of information used in business collaborations. It defines the collaboration itself while lacking ability to define automatable contracts. WS-CCDL is proposed in [13] as a language to define e-Contracts, which is extended in this paper to define contracts for our model. A sample implementation of WS-CCDL is presented in [2].

III. A Motivation Example

In manufacturing industries, especially large and expensive equipment manufacturing, the final assemblers do not reserve parts but procure them from suppliers, after an order received from client. The production methodology is called the “Toyota Production System”, which can significantly improve the equipment providers’ cash flow, but requires strong supplier quality assurance [16]. In this section, we use a part of medical scanner production process as a motivating scenario for our research. For simplification, we suppose there are only two critical parts, i.e. lens and controller unit, which need to be purchased from the suppliers to assemble a complete medical scanner. The business process is briefly illustrated in Figure 1.

A client (C) requests a medical scanner from producer P. After getting the order, P starts the production process. The critical parts - lens and controller unit - are requested in parallel from two different suppliers - Lens Provider (S1) and CU Provider (S2) respectively. S1 and S2 then ship the requested parts to P. After both parts are available, P starts production of the scanner. The completed scanner will finally be delivered to C.

There are some significant characteristics of this process, which are: (a) the transaction takes a long time from order to delivery; (b) a failure of a single activity in the process may lead to the failure of the whole process; (c) activities are carried out in multiple organisations; (d) the failure of the process can cause significant financial loss to all or some of the participants.

These characteristics make this process a typical example for exception-handling in business transaction management. Suppose S1 cannot supply lens in time, the final delivery of the scanner has to be delayed. This leads to a liquidated damage payable as per P’s contract with the client. Or even worse - the transaction may need to be cancelled.

In this particular case, P definitely bears a financial loss. P consequently wants to get compensation from S1 since the fault of S1 brings in the failure; however S1 may not be able to afford the penalty or does not want to pay it.

It is clear from the example that there is a need for a comprehensive technical model to facilitate the management of this kind of failure-sensitive business transactions. However, current transaction management technologies do not have build-in mechanism to handle the financial compensations given in the example, which are not only relates to data reversion in the participants’ databases. In rest of this paper, we present a financial-compensation-based transaction management model to address the above issues.

IV. A new Transaction Model for Business Collaborations

Before presenting our solution, we model the entities, operations and their relationship in the problem domain. There are three types of roles in our system, the role set (R) is defined as

$$R = (P, M, T_{TP})$$

Where P refers the participants in the transactions, M refers to the party that manages the transaction and T_{TP} is the trusted third party to provide financial services to handle compensations, e.g. setting up an escrow account to keep some deposit.

A business process, as the one in our motivating example, involves multiple (web) services provided by different service providers (also called participants). In our transaction model,
we regard each service provider as a service entity (SE), which is the atomic business unit of a business process. In a transaction, a service entity can be a service provider as well as a service consumer. Thus P refers to the total set of service entities involved, \( P = \{ SE \} \).

Each service entity (SE) may provide multiple services to their clients, that is

\[
SE_i = \{ S_{SE_1}^i, S_{SE_2}^i, \ldots, S_{SE_n}^i \} \quad (2)
\]

The interactions, that is, the provision of the services within a business process are carried out in the form of transactions. We define the simplest form of the transaction as atomic transaction(\( T \)), which only involves the provision of one service. It links the provision of the service to the associated contract that has been agreed upon between the two participants.

\[
T^* = (S_j^{SE_1}, C_k^{SE_1}) \quad (3)
\]

In the transactions, the services associated will conduct the business activities (BA) to fulfill their responsibility in the whole business process. For simplicity, here we assume each service is responsible of conducting one business activity, \( S_j^{SE_1} \Rightarrow BA_j^{SE_1} \).

As we suggested, regarding business transaction management, data is not the only thing to be focused on. In fact, we argue that financial-based compensation is the ultimate way to solve business disputations. Business activities are usually non-reversible and transaction participants lose money when a transaction fails. The contract \( C_k^{SE_1} \) here specifies the agreed compensation policies between participants. If a participant causes a vital failure, the transaction manager \( M \) will hold it responsible to compensate the substantial loss of the others, e.g. inform the \( T_{TP} \) to deduct the agreed amount in the escrow account. Details about the contract are elaborated in section 4.

Like in our scanner production process, a group of atomic transactions may be contained within a larger and more complex transaction - a composite transaction(\( T^* \)). A composite transaction may involve other atomic and composite transactions. It is defined as

\[
T^{(1)} = (T^*, T^{(1)}, V) \quad (4)
\]

where \( V \) refers to the dependency (Dep) relations between different transactions contained within, it is defined implicitly or explicitly in the business process logic or contracts

\[
V = (T_i, T_j, Dep_{ikj}) \quad (5)
\]

where:

\[
Dep = (none)\text{optional}dependent\quad (6)
\]

A none dependence between two transactions \( T_i \) and \( T_j \) (\( T_i \rightarrow_{none} T_j \)) implies the status of transaction \( T_i \) has no effect on \( T_j \). An optional dependence \( T_i \rightarrow_{optional} T_j \) implies the completion of \( T_j \) relies on the completion of \( T_i \) yet \( T_j \) has other means available for a completion if \( T_i \) fails. And lastly, a dependent relation \( T_i \rightarrow_{dependent} T_j \) implies the completion of \( T_j \) critically relies on the completion of \( T_i \) and the failure of \( T_i \) inevitably leads to the failure of \( T_j \).

With this construct, we here define the different operations of a transaction.

**Definition 1. Commit Atomic Transaction**

An atomic transaction \( T^*_i \) can be committed, denoted as commit(\( T^*_i \)), if the business activity (BA) is completed and the associated service and contract is fulfilled (if any)

\[
\exists T^* = (S_j^{SE_1}, C_k^{SE_1}) \quad SE_i, provide (S_j^{SE_1}) \Rightarrow SE_i, fullfill(C_k^{SE_1}) \Rightarrow M.commit(T^*_i) \Rightarrow T^*_i \in Cmt \quad \square
\]

where "\( \Rightarrow \)" symbol refers to material implication and \( Cmt \) is the set of committed transactions.

**Definition 2. Commit Composite Transaction**

A composite transaction \( T^{(1)} \) can be committed, denoted as commit(\( T^{(1)} \)), when all the transactions contained within are committed. That is

\[
\forall T \subseteq T^{(1)}, T \in Cmt \Rightarrow M.commit(T^{(1)}) \Rightarrow T^{(1)} \in Cmt \quad \square
\]

When a transaction fails, it needs to be aborted and the related compensations specified in the contracts are to be exercised, so as to remedy the impact of the failure. This is a more practical mechanism to ‘roll back’ a failed transaction. Here we define the Abort operation in a process.

**Definition 3. Roll-forward Atomic Transaction**

A composite transaction can be rolled forward, denoted as roll_F(\( T^* \)), if an uncritically depended atomic transaction \( T \) fails

\[
\exists T^* = (S_j^{SE_1}, C_k^{SE_1}) \quad SE_i, fail (S_j^{SE_1}) \Rightarrow SE_i, fail (C_k^{SE_1}) \Rightarrow M.fail(T^*_i) \Rightarrow T^*_i \in Fld \Rightarrow M.exercise(C_k^{SE_1}, T_{TP}) \quad \square
\]

Given \( T^* \in T^{(1)} \), if \( T^* \rightarrow_{nonopt} T^{(1)} \), M.roll_F(\( T^* \))

where \( Fld \) is the set of failed transactions, and \( T_{TP} \) is the trusted third party that provides financial service for this transaction. Rolling forward refers to the continuation of the composite transaction (if any) after the aborting the failed transaction and exercising the contracts.

**Definition 4. Abort Composite Transaction**

A composite transaction \( T^{(1)} \) needs to be aborted, denoted as abort(\( T^{(1)} \)), when one or more transactions it critically depends on has/have been failed.

\[
\exists T_i \in T^{(1)}, T_i \rightarrow_{opt} T^{(1)} \land T_i \in Fld \Rightarrow M.abort(T^{(1)}) \Rightarrow \forall T \in T^{(1)} \land T \notin Fld \land T \notin Cmt, M.abort(T) \quad \square
\]

With the business transaction model defined above, we can capture and define all the transaction states and their transformation in our transaction model using a state machine as shown in Figure 2.

We argued that in real world, many business activities cannot be simply rolled back once they are carried out, without financial lose. For example, delivered goods cannot be returned to the supplier if it has never been delivered, since there is cost associated in the activity. As such, we say

425
the composite transaction is partially committed after every 
\( \text{commit}(T^*) \) or \( \text{roll}(T^*) \). If an atomic transaction \( (T^*) \)
committed, it means the corresponding business activity is
done successfully, the composite transaction \( (T^{(1)}) \)
will continue to the next business activity and so on till all business
activities in the business process are completed, where \( (T^{(1)}) \)
becomes ‘Completed’. If a \( (T^*) \) is failed, \( (T^{(1)}) \) will
be brought into an intermediate ‘failed’ state. Then system would
start the penalty process and propose a compensation scheme
according to the contracts. After exercising the contracts, it is
possible the transaction can be continued, i.e., a late delivery,
which is fairly common in real world businesses and does not
lead to a vital failure of the business transaction. Once again,
an automatic analysis or vote will be proceeded to determine
whether to continue. The business process continues, i.e. roll
forward the \( (T^{(1)}) \), if most parties see the failure tolerable; and
abort otherwise.

V. E-CONTRACT

Most real world serious business transactions are executed
under legal contracts, which define the legal rights and obligations
of the participants and are agreed and signed by the business partners. As can be seen from our previous modelling,
e-Contract is a natural component of our transaction model,
which specifies the business rules/agreements to be used by a
business transaction management system to determine whether
a business activity in a composite transaction is successful.
e-Contract can be represented using languages/standards: XML,
EDI, WS-CCDL[13], etc. as long as the transaction manager
implements the parsing logic required to understand it.

A contract is consisted of the \textbf{Transacting Parties, Contributions, Liabilities and Agreements}. Transacting Parties
defines the transaction participants involved in the contract,
i.e. service provider and service consumer; The goals of the
business activity are given in \textbf{Contributions} which will be used
for judging the result; \textbf{Liabilities} tells the transaction manager
the amount of deposit needs to be requested and to whom the
deposit need to be disbursed in case the business activity fails
to meet the criteria defined in \textbf{Contributions}; The acceptance
of the contract is given in \textbf{Agreements} in form of electronic
signatures by transacting parties or reference to paper contract
signed.

We extend our previous work [13] and gives the schema
of the e-Contract using XML in Figure 3. A sample contract
between \( S_1 \) and \( P \) in the motivating example is given in Figure
4 to demonstrate our transaction model working together with
such an e-Contract, as a proof of concept.

VI. SYSTEM DESIGN AND CASE STUDY

A. System Architecture and Functional Components

The high level architecture of our system is given in Figure
5 and the functional components include:

\textbf{Business Process(BP) / Service Provider(SP)}

BP defines business activities and coordinates SPs to
achieve the desired goal of \( T^* \). BP can be a normal computer
application, or a BPEL[12] process, which is the de-facto
standard for the orchestration of web services. BP can be
executed standalone, or accompanied with our model.

SP is invoked by BP to provide actual services. SP can
be a standalone Web Service, another BPEL process, or an
application/interface provided by a business entity.

**Financial Compensation based Business Transaction Agent (fcBTxA)**
Each SP and BP is associated with a fcBTxA, which provides interfaces that can be called by its host, which is hidden from the underlying logic of the model and can focus on the application specific logic.

**Financial Compensation based Business Transaction Manager (fcBTxM)**
The Financial Compensation based Business Transaction Manager (fcBTxM) manages \( T^\ast \) and its subsidiary \( T^\ast s \). fcBTxM gives transactional support to BPs in coordinating SPs/BAs. It is ensured by fcBTxM that all results of \( T^\ast s \) in \( T^\ast \) are traceable and convincing. fcBTxM also drives recovery activities including contract enforcement (financial compensation), fault-tolerance decisions where necessary, i.e., after a \( T^\ast \) fails.

**3rd Party Trusted Organization (3TO)**
An independent trusted third party receives and disburses the deposit from business transaction participants (SP). It can be a government organization or a licensed escrow company. The disbursement by the third party is dependent on the commands from fcBTxM, according to the fulfilment of contractually-agreed conditions by the transacting parties.

**B. Interfaces**
We classify the interfaces of the system into two categories: Model APIs and Component APIs. Model APIs, which start with \( tx\_ \) and need to be implemented by BP/SP, specify the interfaces between BP/SP and fcBTxA and further provide BP/SP with the interactions with fcBTxM and 3TO. Component APIs, with \( xa\_ \) as prefixes, are designed for the interaction among fcBTxA, fcBTxM and 3TO. These APIs are hidden from BP/SP and implemented within the model. The implementation should be comparatively generic. Model APIs and Component APIs are listed in table I.

**C. Key points for fcBTxM**
Apart from the logic for the interfaces listed above, the following functionalities should be implemented in fcBTxM to support our contract exercising and forward rolling concept.

- **Log Evidence**
The results of the \( T^\ast \)'s need to be evidenced in fcBTxM for possible disputation settling. The evidence can be stored in fcBTxM or an accountability service. We have provided a comprehensive solution for evidence logging in [21].

- **Managing Compensation**
With the obligations retrieved from the contracts, fcBTxM should be able to work out the compensation scheme and hand it to 3TO for actual deposit disbursement. If the result of a \( T^\ast \) meets the requirement agreed, the deposit should be refunded to the SP; However, if a SP fails to meet the expectation, the deposit should be disbursed according to the agreed contracts.

**D. Put It All Together - System Protocol**
As the key concept of our model, we propose the ‘deposit-before-service’ principle. fcBTxM requests deposit from each SP before it starts the corresponding BA. After the completion or failure of the BA, fcBTxM drives the process to refund the deposit or the exercise of the contract. If failure of the \( T^\ast \) is not vital to \( T^\ast \), it would be rolled forward and the next \( T^\ast \) would be started.

The interaction of the functional components in the system can be illustrated as below. Only for illustration purpose (not part of the protocol), the interactions are prefixed with the components which initiate the APIs.

**E. Case Study**
Upon illustration of the protocol, in the rest of this section
### Category | API Name | Arguments Explanation | Description
---|---|---|---
**Model API** | tx\_Start\((Contracts,T^1\text{ID})\) | All \(T^1\) for each one for a BA. It carries all contract related information | Start the \(T^1\) and submit the involved contracts
| tx\_BAReady\((BA\_ID)\) | The ID of the current BA | Inform BP the BA is ready to start
| tx\_AtomicCommit\((BA\_ID, BA\_Result)\) | \(BA\_Result\) contains the result of the current BA | Commit the \(T^*\) after the BA's completion, if it is successful; The result is then logged as evidence.
| tx\_Complete\((T^1\text{ID})\) | The ID of the \(T^1\) | Called when no BA left undone in the BP to complete \(T^1\)
| tx\_Abort\((T^1\text{ID})\) | | Abort \(T^1\) after a vital failure

**Component API** | xa\_Start\((Contracts,T^1\text{ID})\) | Amount is contained in the contract | Start \(T^1\) in fcBTxM and submit the contracts
| xa\_RequestDeposit\((BA\_ID,Contract)\) | Request deposit for the current BA; Only 3TO holds the deposits
| xa\_SubmitDeposit\((BA\_ID,Contract)\) | Submit deposit to 3TO before starting a BA
| xa\_BAReady\((BA\_ID)\) | Inform BP via fcBtxA the BA is ready to start
| xa\_Complete\((T^1\text{ID})\) | Complete \(T^1\)
| xa\_Abort\((T^1\text{ID})\) | Abort \(T^1\)
| xa\_RollForward\((BA\_ID, BA\_Result)\) | Start the financial compensation process and attempt to continue the \(T^1\)
| xa\_DepositReceived\((BA\_ID, Contract, Amount)\) | Confirm the deposit has been received
| xa\_DisburseDeposit\((BA\_ID, Beneficiary, Amount)\) | \(Beneficiary\) is the final recipients of the deposit for current BA | Disburse the deposit, to either the service provider, or other participants, according to the result and the contract

### Table I
**INTERFACES**

![Sequence chart](image)

Figure 6. Sequence chart: \(T^1\) completed successfully
we study two typical cases of the execution of our model to demonstrate how the model coordinates the BAs/SPs in our motivation example.

- **$T^1$ completed successfully**
  As the flow chart shown in Figure 6, BP is initiated by a production request and starts $T^1$. Both $S_1$ and $S_2$ submit bond to 3TO and then are requested to deliver their own products as independent BAs. Upon the completions of the BAs, BP commits both $T^1$s. After receiving the results $fcBTxM$ request 3TO to refund the deposits back to both $S_1$ and $S_2$. BP runs to the end and completes the $T^1$ in $fcBTxM$.

- **$T^*$ failed but $T^1$ rolled forward**
  The difference, compared to the first case, is that $S_2$ fails to deliver CU while $S_1$ does its activity as expected. BP captures the failure of $S_2$ and uses a backup provider afterwards [12]. BP sends the result to $fcBTxM$ to exercise the corresponding contract for the failed BA but rolls forward the $T^1$. This flow is shown in Figure 7.

Here we only show the process of putting deposits for $S_1$ and $S_2$. In real business, BP should also put deposits.

VII. PERFORMANCE EVALUATION

We have implemented an experimental business process in Amazon EC2 - a computing resource provisioning service that charges the user according to the CPU usage. As shown in the sequence chart, we deployed 5 services nodes on 5 standard computing instances in EC2 - these are virtual machines with computing power equivalent to 1GHz CPU and 1.7GB memory. Our implementation used Apache Tomcat 5.5 as our Servlet container, and Axis2 1.5 as our web service engine. We used BPEL to orchestrate the service nodes to form our medical scanner production process. Apache ODE has been used in the transaction manager to conduct the business process defined in BPEL.

The interaction protocols with the transaction manager ($fcBTxM$) and 3rd party trusted organization (3TO) are incorporated into the original business processes deployed at the five service nodes. We conducted testing to evaluate the latency introduced by incorporating the transaction manager and 3TO. Figure 8 shows the overall latency to finish the business process with and without transaction manager and 3TO. For the process with transaction manager and 3TO incorporated (the series marked with ‘circles’), the latency introduced compared to the one without (the series marked with ‘squares’) grows as the request message becomes larger. In percentage terms, on average we observed a 27% increase in the overall latency. Certainly, this overhead is noticeable if
not significant. However, the latencies in our testing are all due to the message exchanging only, there is no local computation involved. In real practise, there will be other service or non-service based local activities, such as the manufacturing of the products which may take hours or even days. Considering these, it is reasonable to conclude that the extra latencies introduced by incorporating $feBTxM$ and 3TO are acceptable and cost-effective to coordinate the collaboration.

VIII. Summary

In this paper we addressed a problem from real business practise and proposed a novel transaction model, to raise a new perspective for business transaction management. Differing from existing transaction models, our model does not do data compensation when a failure occurs. Instead, the model focuses on e-Contracts and financial compensation since this approach fits the real world business transactions better. The model also ensures the benefits and accountabilities of the transacting parties by enforcing a ‘deposit-before-service’ protocol, making the exercisability of the contracts guaranteed. We gave a formal modelling, using state machine to capture all transactional states and their relationships/transitions, and proposed a systematic design of the model. A reasonable evaluation has also been done in a real cloud (Amazon) to prove the model is implementable and not adding much additional cost to the infrastructure. We believe our model can be potentially developed to an eBusiness platform, helping business partners from various perspectives.

IX. References


