Verification of Service Composition and Compensation by Using Process Algebra

S. Ripon, F. Sultana, and F. Rahman

***Abstract*—Web service technology provides a platform that facilitates the development of distributed services. In order to support business to business interactions within the distributed environment a crying need is to aggregate web services and assemble them is a goal oriented infrastructure. With the emergence of web services, the coordination and interaction involved between multiple business partners are conducted by using the web services. Faults can arise at any stage of business transaction and handling such faults where multiple partners are involved is both crucial and difficult. Process algebras can be used to model concurrent and distributed interactive systems. Compensating CSP is a language defined to model business transactions within the framework of CSP process algebra. It has the facility to model faults within a transaction as compensations. However, the language lacks automated tool support to verify the service composition. Finite state Process (FSP), on the other hand, is designed to model the composition of web services and importantly, it has an automated tool support for verification of composition of services, however there is no construct for compensation. By combining the benefits the both cCSP and FSP, this paper illustrates a mechanism to model and verify the composition of services and compensation in FSP by following the mechanism adopted in cCSP. The verification of composition properties is performed by LTSA tool.**

***Index Terms*—Compensation, web services, cCSP, FSP.**

I. Introduction

Business transactions involve multiple partners, and need coordination and interaction with each other. Many business companies or enterprises publish their applications' functionalities on the web by using web service. Web services are defined as self-contained, modular units of application logic, which provide business functionality to other applications through an Internet connection. Each service provider is a self-contained software system having its own threads of control. Web services technology facilitates the development of distributed services. There are various standard protocols namely WSDL [1], UDDI [2], SOAP [3] that are defined to describe, look for and access the available services. In this technological era, business applications like web services allow greater efficiency and availability for businesses. A web service alone has a limited functionality which may not be sufficient to respond to the user's request. Whereas a composition of several web services can achieve a specific goal. From a user perspective, the composition might be considered as a simple web service, even though it is composed (choreographed) of several web services. In an essence, the aggregation is a collaboration of many Web service providers.

Coordination among the web services is particularly crucial as it describes the logic that makes a set of different services as a whole system. Hence these coordination models and languages need through formal study along with proving their various properties. It has been suggested [4], [5] that process algebra can be used to formally define the coordination of web services. Process algebras make it easy to specify the message exchange between web services.

Business transactions have to deal with faults that can arise at any stage of transaction. Transactions that require long period of time to complete is known as Long Running Transaction (LRT) [6] and separate measures are required to handle faults in LRT. Compensation is an error recovery mechanism which is particularly used in LRTs. Compensation mechanism has to be incorporated within a transaction so that faults can be handled automatically in LRTs. Various works suggested to model business transactions by using process algebras [7]-[10]. However, very few of them included compensation mechanism within the language.

Compensating CSP (cCSP) [11]-[13] is a language defined to model business transactions within the framework of standard CSP [14] process algebra. In cCSP transactions are defined as processes. Besides, the language has constructs to define orchestration of compensation to handle faults when required. However, the language lacks automated tool support to verify the service composition. Finite state Process (FSP) [15], on the other hand, is designed to model the composition of web services and importantly, it has an automated tool support to verify the composition of services, however there is no construct in the language for compensation. The objective of this paper is to combine the benefits of both cCSP and FSP. We define a mechanism to model the composition of services along with corresponding compensation in FSP by adopting the technique proposed in cCSP. In particular, the management of compensation is incorporated into FSP encoding of the service composition. Verification of service composition, compensation and various correctness properties is carried out in LTSA tool [16].

The rest of the paper is organized as follows. A brief overview of cCSP is illustrated in Section II. In the following section we give an example of web service composition and draw the schematic diagram of the service composition. We also draw message sequence chart of service composition along with compensation which clearly shows the compensation handling mechanism that is to be followed in this paper. In Section IV, we show how the web services are encoded into FSP. The verification of service composition,

Manuscript received October 25, 2016; revised December 17, 2016.

S. Ripon, F. Sultana, and F. Rahman are with the Department of Computer Science and Engineering, East West University, Bangladesh (e-mail: dshr@ewubd.edu).

doi: 10.18178/jacn.2016.4.4.230

compensation and various properties are also shown here. The following section illustrates a brief comparison between cCSP constructs and their corresponding representation in FSP. Finally, we conclude the paper by summarizing our work and future plan in Section VI.

II. COMPENSATING CSP

Compensating CSP (cCSP) is a language defined to model long running business transactions within the framework of standard CSP [csp] process algebra. Processes in cCSP are modeled in terms of the atomic events they can engage in. The processes are categorized into standard and compensable processes.

A standard process does not have any compensation. The basic unit of the standard processes is an atomic event (A). The other operators are the sequential ($P; Q$), and the parallel composition ($P \,||\, Q$), the choice operator ($P \,\square\, Q$), the interrupt handler($P \rhd Q$), the empty process SKIP, raising an interrupt THROW, and yielding to an interrupt YIELD. A process that is ready to terminate is also willing to yield to an interrupt. In a parallel composition, throwing an interrupt by one process synchronizes with yielding in another process.

Compensation is part of a compensable process that is used to compensate a failed transaction. In a sequential composition, the compensation is defined in such a way that the compensations of the completed tasks will be accumulated in reverse to the order of their original composition, whereas compensations parallel processes will be placed in parallel. We use notations $P, Q, ..$ to identify standard processes, and $PP, QQ, ..$ to identify compensable processes. The basic way of constructing a compensable process is through a compensation pair ($P \div Q$), which is constructed from two standard processes, where P is called the forward behavior that executes during normal execution, and Q is the associated compensation that is designed to compensate the effect of P when needed.

$SKIPP, THROWW$, and $YIELDD$ are the compensable counterpart of the corresponding standard processes and they are defined by pairing an empty compensation with them, e.g., $SKIPP = SKIP \div SKIP$. cCSP language syntax is summarized in Table I.

TABLE I: CCSP SYNTAX

Standard Processes		Compensable Processes	
$P, Q := A$	(atomic event)	$PP, QQ := P \div Q$	(compensation pair)
$\|P ; Q$	(sequential composition)	$\|PP ; QQ$	
$\|P \,\square\, Q$	(choice)	$\|PP \,\square\, QQ$	
$\|P \,\|\|\, Q$	(parallel composition)	$\|PP \,\|\|\, QQ$	
|SKIP	(normal termination)	$\|SKIPP$	
|THROW	(throw an interrupt)	$\|THROWW$	
|YIELD	(yield to an interrupt)	$\|YIELDD$	
$\|P \rhd Q$	(interrupt handler)		
|[PP]	(transaction block)		

III. SERVICE COMPOSITION

We model a car broker web service as a case study. A car broker web service support customer negotiating car purchasing and arranges loan for the same. The broker service utilizes two web services: Supplier to search suitable quote based on customer demand and LoanStar, a lender service that arranges loan for customer to buy car. Each web service is independent and can be combined with any other web services. Our emphasis is on the coordination of services and management of compensation mechanism. The compensations have to be orchestrated in such a way that whenever there occurs an interrupt appropriate compensation process will be executed in proper order. Each web service is modeled as an independent process. Separate processes are defined to model the corresponding compensations. Finally, the processes representing all the services are composed in parallel.

Buyer: First, Buyer gives an order to Broker to find a quote of a car. Buyer then receives a suitable quote from Broker. The Buyer can either accept or reject the quote. If the quote is satisfactory, Buyer either sends a confirmation message to Broker or reject the quote by throwing a message.

Broker: After receiving the order from the Buyer the Broker requests the Supplier for available quotes. The buyer then select a quote from the received quotes and then Broker simultaneously sends quote to Buyer, gives an order to the Supplier and requests for loan to the `LoanStar` assuming that buyer will accept the final order. Broker accepts all the positive acknowledgements from Buyer, Supplier and `LoanStar` if Buyer accepts the quote, Supplier is able to provide the requested model and `LoanStar` approved the loan then we can say that the order is complete.

Supplier: Supplier is a service that receives request for quotation from Broker in accordance to the order of a particular car model by Buyer. Getting the request for quotation, Supplier collects quotes from all of its associated partners and passes the accumulated quotes to the Broker. Supplier receives a final order from Broker while Broker selects a suitable quote for Buyer. If the Supplier is able to manage the desired car model ready to supply, it acknowledges Broker by a positive reply.

LoanStar: LoanStar is assumed as a lender web service that offers loans to online Buyers. After a detailed assessment of the loan, LoanStar can either approve the loan or reject it. If the assessment outcome is positive loan request is granted and LoanStar sends a positive acknowledgement to Broker. Architectural view of the example scenario is depicted in Fig. 1.

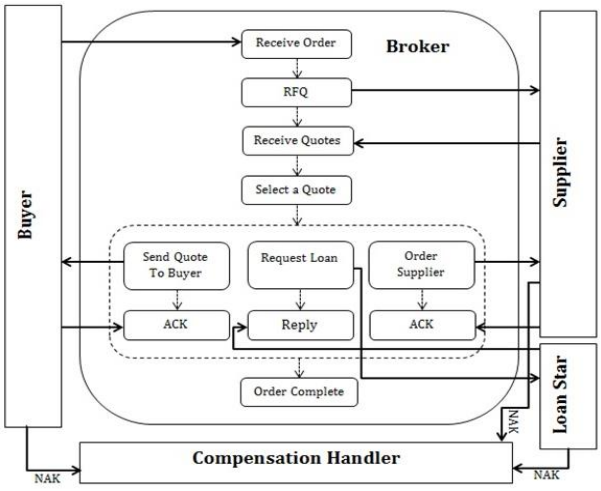
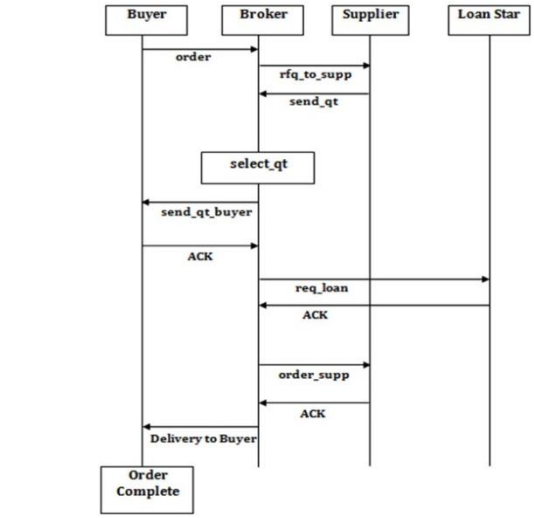


Fig. 1. Architectural view of Car Broker web service.

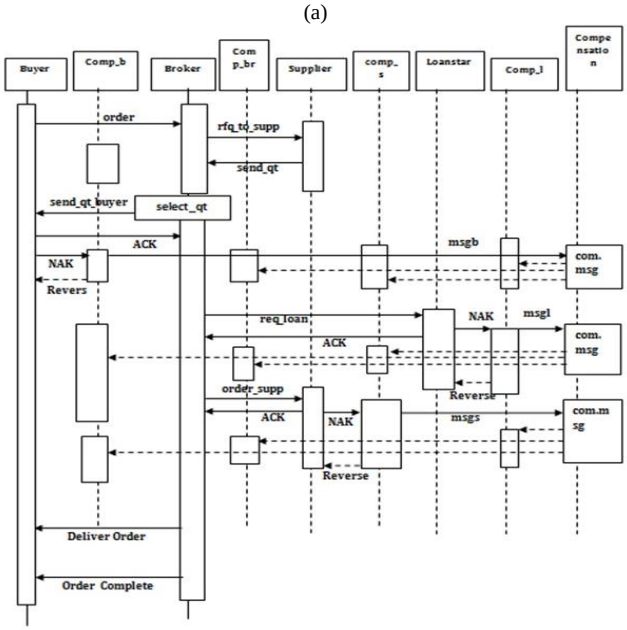
If any negative acknowledgement is thrown by any service it is considered as a fault or error of the system for which

service cannot be continued anymore. That's why we have to handle errors by compensating the services.

Using the compensation mechanism all services can reach into a state that can be considered as an equivalent to their initial state from where they have been interrupted. In our model while a negative acknowledgement is thrown by a service this message is received by the compensation process of this service. The reverse actions are performed to compensate the service from where the interruption occurs. Simultaneously the compensation process throws an interrupt to the main compensation handling process and it then throws a combination of messages that will be received by the compensation processes of the respective services attached to the system and runs the compensating actions in parallel. The message sequence chart (MSC) of the composite services with and without compensation handler are shown in Fig. 2(a) and 2(b).



(a)



(b)

Fig. 2. Message sequence chart of overall composition.

IV. FSP Representation

FSP stands for Finite State Processes. Finite State Processes is an algebraic notation to describe process models. The constructed FSP can be used to model the exact transition of workflow processes through a modeling tool such as the Labeled Transition System Analyzer (LTSA), which provides compilation of an FSP into a Labeled

Transition System. Models are described using state machines, known as Labeled Transition Systems LTS. These are described textually as finite state processes (FSP) and displayed and analyzed by the LTSA analysis tool. The tool gives an opportunity to test the workflows before implementing the model. LTS is the graphical form and FSP is the algebraic form. [5]. FSP consists of Action Prefix, Process Definition, Choice, Indexed Processes and Actions, Guarded Actions, properties, Constant and Range Declarations, Variable Declaration, Process Alphabets and so on.

In our system there are four major processes which have their own compensation and each of them has their safety properties to ensure a good composition.

```
/*   Buyer  */
BUYER =
(order->rcv_qt->reply->(send_b_ack->BUYER|send_b_n
ak->thrwb->END)).
COMP_B = (thrwb->cancel_rcv_qt->cancel_order->END).
MSGB = (thrwb->msgb->END).
/*  Broker  */
BRK_PHASE1 =
(rcv_order->rfq_to_supp->rcv_qt_supp->select_qt->E
ND).
REQ1 = (select_qt->send_qt_buyer->reply->END).
REQ2 = (select_qt->order_supp->reply->END).
REQ3 = (select_qt->req_loan->reply->END).
RCV1 = (reply->rcv_buyerack->END).
RCV2 = (reply->rcv_suppack->END).
RCV3 = (reply->rcv_loanack->END).

||REQ = (REQ1||REQ2||REQ3).
||RCV = (RCV1||RCV2||RCV3).
||BRK_PHASE2 = (REQ||RCV).
||BROKER = (BRK_PHASE1||BRK_PHASE2).
||COMP_BRK = (BRK_PHASE2_COMP||BRK_PHASE1_COMP).

CMP_REQ1 = (thrwbrk->wdrw_buyer_qt->reqwdrwn->END).
CMP_REQ2 = (thrwbrk->wdrw_s_order->reqwdrwn->END).
CMP_REQ3 = (thrwbrk->wdrw_l_req->reqwdrwn->END).
||BRK_PHASE2_COMP = (CMP_REQ1||CMP_REQ2||CMP_REQ3).

BRK_PHASE1_COMP =
(reqwdrwn->cancel_qt_select->cancel_supp_qt_rcv->
 cancel_rfq_to_supp->cancel_buyer_order->END).
/**    Supplier  */
SUPPLIER =
(rcv_rfq->send_qt->rcv_brk_order->reply->(send_s_a
ck->SUPPLIER|send_s_nak->thrws->END)).
COMP_S =
(thrws->cancel_brk_order->cancel_qt->cancel_rfq->E
ND).
MSGS = (thrws->msgs->END).
/** Lender  */
LOANSTAR =
(rcv_req->reply->(send_l_ack->LOANSTAR|send_l_nak-
>thrwl->END)).
COMP_L = (thrwl->cancel_loan_req->END).
MSGL = (thrwl->msgl->END).
```

Fig. 3. FSP encoding of buyer, broker, supplier and LoanStar.

Buyer: The process starts the service by giving an `order` for a car to Broker. When Buyer receives a quote from Broker `rcv_qt` it checks whether the quote is suitable or not. If the quote is suitable then it either sends an acknowledgement `send_b_ack` to Broker or denies the quote by throwing a negative acknowledgement `send_b_nak`. Buyer's

compensation process is composed of two processes. One process consists of compensating actions to compensate Buyer actions and another contains a message which alerts the main compensation process that an interrupt is thrown from Buyer. The compensation process COMP_B consists of compensation actions to be performed by Buyer before sending a negative acknowledgement. The action thrwb is the synchronizing action for those processes who tries to execute it. When COMP_B runs, at the same time another process MSGB also run in parallel. MSGB throws a message msgb which is received by the Main Compensation Process. msgb indicates that Buyer process is not running anymore; so getting this message Main Compensation Process will take necessary steps to compensate others. The FSP encoding of the service composition is illustrated in Fig. 3.

Broker: Broker interacts with other three partner services. The process works in two phases. Phase one consists of several sequential actions and Phase two is the concurrent execution of three parallel processes. Phase one starts by receiving an order from Buyer labeled as rcv_order. According to Buyer's order Broker request for the quotes to the Supplier. The actions rfq_to_supp. rcv_qt_supp represents that Broker receives the quotes from Supplier. Broker finds the best possible quote for the requested car model by Buyer represented by the action select_qt. After selecting the quote in Phase two, Broker simultaneously send quote to Buyer, order to Supplier and request loan to LoanStar by using the process REQ. The order will be completed without any error if and only if Buyer, Supplier and LoanStar send positive acknowledgements to Broker. These acknowledgements received in the process RCV and the respective receiving messages are rcv_buyerack, rcv_suppack, rcv_loanack. Broker Phase two is the composition of these two processes REQ and RCV and the composition is titled as BRK_PHASE2. Finally, the Broker process is the composition of two phases; BRK_PHASE1, the sequential part of Broker and BRK_PHASE2, the parallel part of Broker.

Broker's Compensation Process COMP_BRK consists two phases. COMP_BRK is composed of two separate processes, BRK_PHASE2_COMP and BRK_PHASE1_COMP. BRK_PHASE2_COMP is the compensation process of Broker's parallel part. After getting an interrupt from Main Compensation Process Broker's phase two compensation process withdraws all placed request to its partner processes with the actions wdrw_buyer_qt, wdrw_s_order and wdrw_l_req. These actions are composed in parallel with the separate respective processes CMP_REQ1, CMP_REQ2 and CMP_REQ3 in BRK_PHASE2_COMP process. reqwdrwn indicates that all requests are successfully withdrawn. Broker's Phase One Compensation Process BRK_PHASE1_COMP starts after successfully withdrawing all requests placed by Broker to its partner processes. Process BRK_PHASE1_COMP consists a sequence of actions that cancels every actions performed by Broker after receiving an order from Buyer till selecting a quote among various quotes sent by supplier.

Supplier: Supplier receives a request for quotes from the Broker by rcv_rfq. According to the request, Supplier sends accumulated quotes to the Broker by send_qt. After selecting the appropriate quote Broker sends an order for car and that is received by the action labeled rcv_brk_order at the Suppliers end. If the Supplier able to deliver the order it confirms Broker by sending an acknowledgement send_s_ack otherwise it rejects the order and sends a negative acknowledgement send_s_nak. Supplier's Original Compensation process is composed of two processes. One is Supplier's compensation process and another is a messaging system alerts the Main Compensation Process that a negative acknowledgement is thrown from Supplier. COMP_S, the compensation process of Supplier reverse the actions which are already done by Supplier before sending a negative acknowledgement. This process is synchronized through a shared action thrws while Supplier needs to be compensated. When COMP_S runs, at the same time another process MSGS also run in parallel. MSGS throws a message msgs which is received by the Main Compensation Process. This message indicates that Supplier process is not able to run anymore and the compensation process of supplier is running; after getting this message Main Compensation Process will take necessary steps to compensate others.

LoanStar: After selecting the quote Broker sends a request to its business partner LoanStar to arrange a loan for Buyer. This request is received in LoanStar by the action rcv_req. Loanstar confirms the approval of the loan by sending an acknowledgement send_l_ack. Loanstar rejects the loan request using send_l_nak if it is not able to arrange the loan for the Buyer. Process COMP_L and MSGL are used to compensate LoanStar's activities. While COMP_L runs, simultaneously a process MSGL runs. COMP_L compensates LoanStar's actions those took place before sending a negative acknowledgement. A shared action thrwl is used to synchronize with those processes that are willing to execute the compensation process of LoanStar. MSGL throws a message msgl which is received by the Main Compensation Process. msgl indicates that LoanStar rejects the loan request. So Main Compensation Process has to take necessary steps to compensate others.

```
CMAIN = (msgb->COMP_EXCPT_BUYER|msgs->
COMP_EXCPT_SUPP|msgl->COMP_EXCPT_LOAN),
COMP_EXCPT_BUYER = FROM_BUYER;END,
COMP_EXCPT_SUPP = FROM_SUPP;END,
COMP_EXCPT_LOAN = FROM_LOAN;END.

BUYERMSG_TO_COMP_BRK = (thrwbrk->END).
BUYERMSG_TO_COMP_S = (thrws->END).
BUYERMSG_TO_COMP_L = (thrwl->END).
||FROM_BUYER = (BUYERMSG_TO_COMP_BRK||
BUYERMSG_TO_COMP_S||BUYERMSG_TO_COMP_L).

SUPPMSG_TO_COMP_B = (thrwb->END).
SUPPMSG_TO_COMP_BRK = (thrwbrk->END).
SUPPMSG_TO_COMP_L = (thrwl->END).
||FROM_SUPP = (SUPPMSG_TO_COMP_B||
SUPPMSG_TO_COMP_BRK||SUPPMSG_TO_COMP_L).

LOANMSG_TO_COMP_B = (thrwb->END).
LOANMSG_TO_COMP_BRK = (thrwbrk->END).
LOANMSG_TO_COMP_S = (thrws->END).
||FROM_LOAN = (LOANMSG_TO_COMP_B||
LOANMSG_TO_COMP_BRK||LOANMSG_TO_COMP_S).
```

Fig. 4. FSP encoding of compensation process.

Compensation Process: When a negative acknowledgement is given by any partner processes of Broker, a message is received by the Main Compensation Process. From the message, the main compensation process identifies from which process the interrupt is thrown. On the basis of the sent message a combination of messages is generated by the Main Compensation Process to compensate other processes except the process from where the message was received. The FSP encoding is shown in Fig. 4 and the transition diagram of the compensation process is shown in Fig. 5.

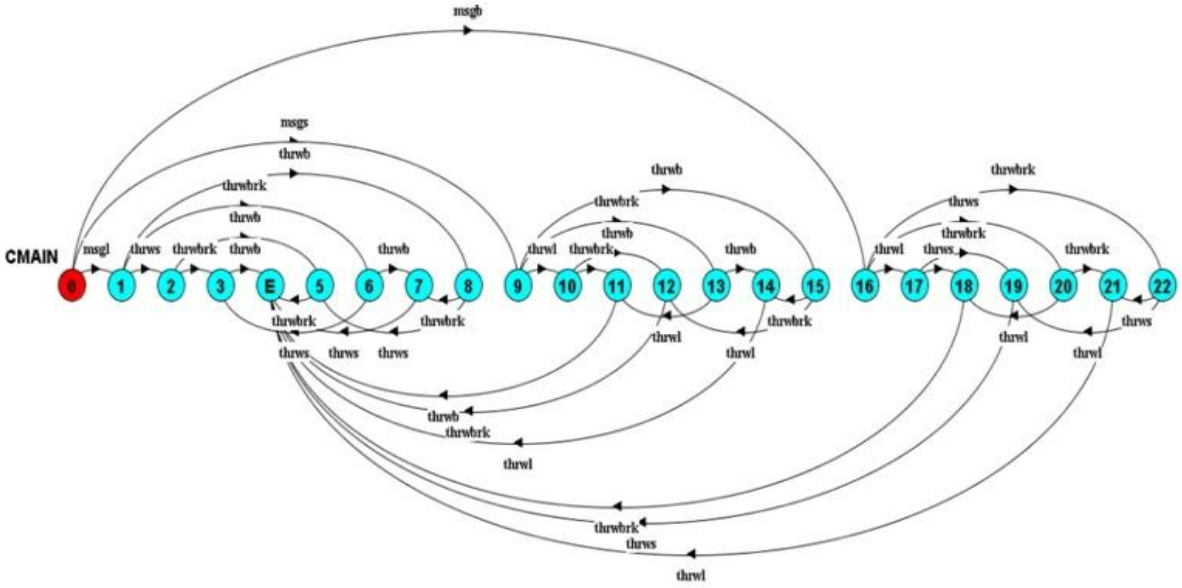


Fig. 5. LTSA representation of CMAIN process.

CARBROKERSERVICE Process is the parallel composition of all engaged processes to the system with all safety properties. All major services, their compensation processes, all messages throwing processes, CMAIN the main compensation process all together composed in CARBROKERSERVICE. All the services have been synchronized with each other through the relabeling functions (Fig. 6).

```
||CARBROKERSERVICE = (
BUYER||BROKER||SUPPLIER||LOANSTAR||
MSGB||MSGS||MSGL||CMAIN||
COMP_B||COMP_BRK||COMP_S||COMP_L||
SAFE_COMP_B||SAFE_COMP_S||SAFE_COMP_L||
SAFE_MSG_BRK||SAFE_MSG_B||SAFE_MSG_S||
SAFE_MSG_L|  SAFE_SYSTEM||SAFE_REQ1||SAFE_REQ2||
SAFE_REQ3)
/{
rcv_order/order, rcv_rfq/rfq_to_supp,
rcv_qt_supp/send_qt,       rcv_qt/send_qt_buyer,
rcv_req/req_loan, rcv_brk_order/order_supp,
rcv_buyerack/send_b_ack, rcv_loanack/send_l_ack,
rcv_suppack/send_s_ack
  }.
```

Fig. 6. FSP encoding of final service composition.

A. Composition Verification

It is mentioned earlier that modeling compensation of each process is one of the central focuses of this work. In the first stage of verification, we check whether the compensation of each process act accordingly whenever an interrupt is thrown from any process. For each process, a property process is defined in FSP and then it is composed in parallel with the main process. It is assumed that interrupt in the form of negative acknowledgement can be thrown by buyer, supplier and/or Loanstar and corresponding property processes are defined for each of them as shown in Fig. 7.

```
property SAFE_COMP_B =
(send_b_nak->cancel_rcv_qt->SAFE_COMP_B).
property SAFE_COMP_S =
(send_s_nak->cancel_brk_order->SAFE_COMP_S).
property SAFE_COMP_L =
(send_l_nak->cancel_loan_req->SAFE_COMP_L).
||BSAFE = (BUYER||COMP_B||SAFE_COMP_B).
||SSAFE = (SUPPLIER||COMP_S||SAFE_COMP_S).
||LSAFE = (LOANSTAR||COMP_L||SAFE_COMP_L).
```

Fig. 7. Safety property of Buyer, Supplier and LoanStar.

In order to check the main compensation process itself, four property processes are defined. A process is defined to confirm that when an interrupt is thrown (negative acknowledgement) either by Buyer, Supplier or LoanStar, the main compensation process will run the compensation process of Broker by throwing appropriate messages. Another process is defined to confirm that the compensation handler process runs the compensation process of Buyer by throwing a message when a negative acknowledgement is received from Supplier or LoanStar. Similarly, the other two process are defined to confirm the execution of compensation of Supplier and Loanstar process when interrupt is thrown from other respective processes. All these property processes are then composed into parallel to check the correctness the compensation processes together (Fig. 8).

```
property SAFE_MSG_BRK =
(msgb->thrwbrk->SAFE_MSG_BRK
|msgl->thrwbrk->SAFE_MSG_BRK|msgs->thrwbrk->SAFE_M
SG_BRK).
property SAFE_MSG_B =
(msgs->thrwb->SAFE_MSG_B|msgl->thrwb->SAFE_MSG_B).
property SAFE_MSG_S =
(msgb->thrws->SAFE_MSG_S|msgl->thrws->SAFE_MSG_S).
property SAFE_MSG_L =
(msgb->thrwl->SAFE_MSG_L|msgs->thrwl->SAFE_MSG_L).
```

```
||CMAIN_CHECK=(CMAIN||COMP_B||COMP_BRK||COMP_S||CO
MP_L||SAFE_MSG_BRK||SAFE_MSG_B||SAFE_MSG_S||SAFE_M
SG_L).
```

Fig. 8. Correctness check of compensation process.

B. Verifying System Composition

The property process SAFE_SYSTEM is defined to ensure that Buyer, Broker and Supplier processes synchronize correctly in their desired synchronizing points in the system up to selection of quote by broker. After selecting a quote, the quote is sent to Buyer, a loan request for the quote is placed to LoanStar and an order is placed to Supplier simultaneously. These requests are received by those three service processes using rcv_qt, rcv_brk_order, rcv_req actions. SAFE_REQ1, SAFE_REQ2 and SAFE_REQ3, these three safety properties ensures that the requests placed in parallel to Buyer, Supplier and LoanStar by Broker is successfully received. Fig. 9 shows the FSP encoding and transition diagram.

If the property SAFE_SYSTEM and SAFE_REQ1, SAFE_REQ2, SAFE_REQ3 are composed in parallel with the processes BUYER, BROKER, SUPPLIER and LOANSTAR in MAINSYSTEM_CHECK and the traces of MAINSYSTEM_CHECK does not show any violation in LTS diagram we can say that our system is verified with the written SAFE_SYSTEM property process which ensures that the system has been synchronized successfully.

```
property SAFE_SYSTEM =
(rcv_order->rcv_rfq->rcv_qt_supp->select_qt->SAFE_
SYSTEM).
property SAFE_REQ1 =
(select_qt->rcv_qt->SAFE_REQ1).
property SAFE_REQ2 =
(select_qt->rcv_brk_order->SAFE_REQ2).
property SAFE_REQ3 =
(select_qt->rcv_req->SAFE_REQ3).
```

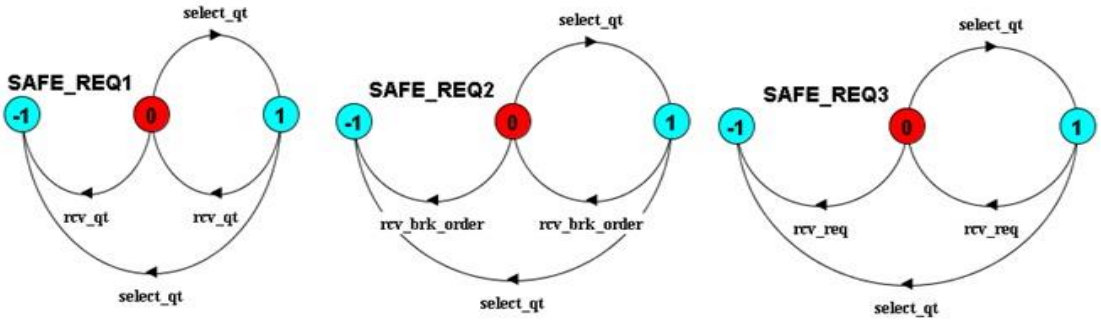


Fig. 9. FSP and LTSA representation of safety property SAFE_REQ1, SAFE_REQ2, SAFE_REQ3.

On the other hand we can also say that the requests are made by the Broker are successfully received by its partner services. If there any violation occurs in the traces we can say that system synchronization might not ok or else there is a fault in the message passing system. As Including LOANSTAR in the composition of MAINSYSTEM_CHECK, it generates too many states. So, by omitting LOANSTAR from the composition we can generate the LTSA representation of the process MAINSYSTEM_CHECK (see Fig. 10).

V. Comparison with cCSP

Both cCSP and FSP support prefix, sequence, choice, parallel operations over processes. The main distinction between these two algebras is the definition of compensation. In cCSP there is a compensable process that includes compensation in the process definition. The basic way of constructing a compensable process is via compensation pair $(P \div Q)$. It is constructed of two standard processes: P is the forward process that executed during normal operation and Q is the attached compensation that is executed to compensate the actions in P when the forward behavior of P throws an interrupt. The compensation for sequence and parallel operations are defined in such a way that when compensation is required the compensations attached to each process will execute in reverse to the original operations.

In FSP compensation process and the main process are two separate processes. The main process and its compensation process are composed in parallel. If any interrupt occurs in the main process the process throw a message, that message is received by the compensation process with the collaboration of a shared action. Thus both processes synchronize and the compensation process runs the compensating actions of those actions that have already took place by the main process. Here P is the main process also called forward behavior and Q is its corresponding compensation process, contains the reverse actions done by P. P_Q is the parallel composition of the main process P with its compensation process Q that resembles to the compensation pair (P÷Q) as referred in cCSP. The cCSP syntax and the corresponding FSP representation is illustrated in Table II.

```
||MAINSYSTEM_CHECK =
(BUYER||BROKER||SUPPLIER||LOANSTAR||SAFE_SYSTEM
||SAFE_REQ1||SAFE_REQ2||SAFE_REQ3)
/{rcv_order/order, rcv_rfq/rfq_to_supp,
rcv_qt_supp/send_qt, rcv_qt/send_qt_buyer,
```

```
  rcv_req/req_loan,     rcv_brk_order/order_supp,
  rcv_buyerack/send_b_ack,
rcv_loanack/send_l_ack,
  rcv_suppack/send_s_ack }
```

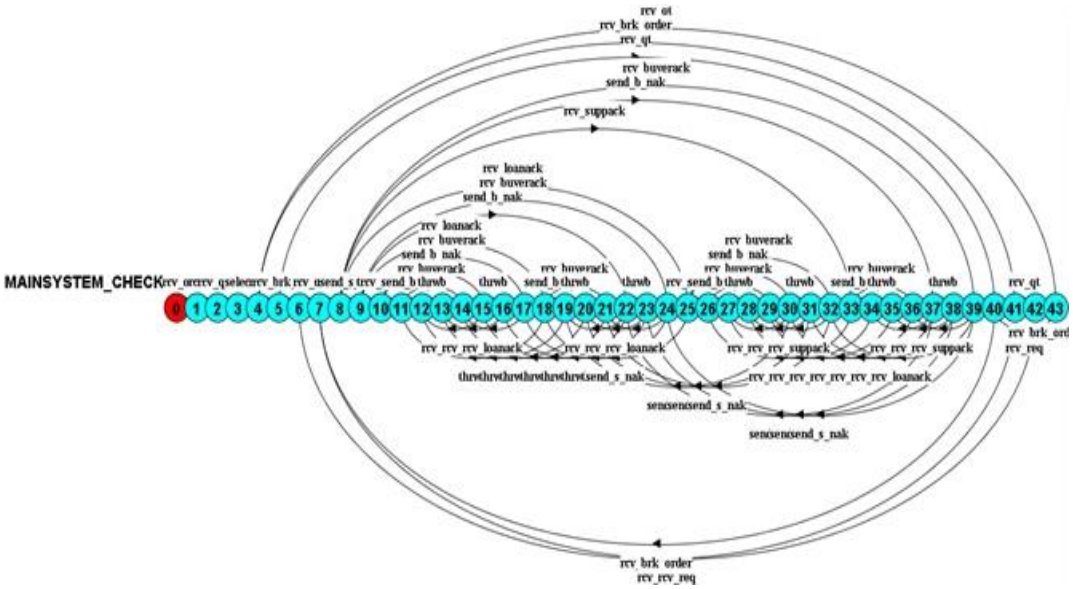


Fig. 10. FSP representation of MAINSYSTEM_CHECK.

TABLE II: COMPARISON BETWEEN CCSP AND FSP

Syntax	cCSP	FSP
Sequential Composition	P ; Q	P= (action1->END). Q= (action2->END). SEQUENCE=P;Q ;END.
Choice	P □ Q	Deterministic Choice: (x->P| y->Q) Non-Deterministic Choice: (x->P| x->Q)
Parallel Composition (without relabeling "/")	P ‖ Q	P = (a ->b -> END). Q = (c-> END). ||SYS = (P || Q).
Parallel Composition (with relabeling "/")	$P \|\|_x Q$	P= (a ->b ->e->P). Q = (c->d->Q). ||SYS = (P || Q)/ {c/b, e/d}.
Compensation Pair	P ÷ Q	P = (act1->(ack->P|nak->throw ->END)). Q = (throw->cancel_act1->END). ||P_Q = (P||Q).

VI. CONCLUSION

Proper management of compensation plays a key role in handling faults in long running transactions in web services where traditional fault handling principles do not work properly. Compensation is included as an integral part within the definition of processes in cCSP that allows the compensation to handle faults which might arise at any time during transaction by invocation of a throw from any participating process in the composite web services. The compensations are modeled in such a way that it performs the compensation task in reverse order to the original transaction. Due to lack of tool support for cCSP, this paper described a case study of modeling the choreography of compensable services in FSP. The compensable processes are designed by combining a normal process with its compensation process. By following the approach proposed in cCSP, the overall orchestration is designed in such a way that appropriate compensations are executed in proper order that cancel the actions of already completed process whenever there is an interrupt thrown from any process within the composition. It has also been illustrated how cCSP constructs are defined in FSP. Properties that check the composition and execution of compensations are also verified by using LTSA model checker. Such verification strengthens the claim of cCSP and confirms that the compensation modeling approach can be applied in long running transactions

REFERENCES

[1] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web services description language (WSDL) 1.1. W3c note," World Wide Web Consortium, March 2001.

[2] OASIS, "Introduction to UDDI: Important feature and functional concepts," Technical report, Organization for the Advancement of Structured Information Standard, 2004.

[3] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the web services web: An introduction to SOAP, WSDL, and UDDI," *IEEE Internet Computing*, vol. 6, no. 86–93, 2002.

[4] L. G. Meredith and S. Bjorg. "Contracts and types," *Communications of the ACM*, vol. 46, no. 10, pp. 41–47, October 2003.

[5] G. Sala ün, L. Bordeaux, and M. Schaerf, "Describing and reasoning on web services using process algebra," in *Proc. the IEEE International Conference on Web Services*, IEEE Computer Society, June 6-9, 2004.

[6] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann Publishers, 1993.

[7] M. Berger and K. Honda, "The two-phase commitment protocol in an extended pi-calculus," *Electronic Notes in Theoretical Computer Science*, vol. 39, no. 1, 2000.

[8] A. P. Black, V. Cremet, R. Guerraoui, and M. Odersky, "An equational theory for transactions," in *FSTTCS 2003: Foundations of Software Technology and Theoretical Computer Science*, P. K. Pandya and J. Radhakrishnan, Eds. vol. 2914, pp. 38–49, Mumbai, India, December 15-17, 2003, Springer-Verlag.

[9] L. Bocchi, C. Laneve, and G. Zavattaro, "A calulus for long-running transactions," in *FMOODS'03*, vol. 2884, pp. 124–138, Springer-Verlag, 2003.

[10] R. Bruni, C. Laneve, and U. Montanari, "Orchestrating transactions in join calculus," in *CONCUR '02: Proceedings of the 13th International Conference on Concurrency Theory*, L. Brim, P. Jancar, M. Kret´ınsk´y, and A. Kucera, Eds. vol. 2421, pp. 321–337, 2002.

[11] M. Butler, T. Hoare, and C. Ferreira, "A trace semactics for long-running transaction," in *Proc. 25 Years of CSP*, A. E. Abdallah, C. B. Jones, and J. E. Sanders, Eds., vol. 3525, London, 2004, Springer-Verlag.

[12] M. Butler and S. Ripon, "Executable semantics for compensating CSP," in *WS-FM 2005*, M. Bravetti, L. Kloul, and G. Zavattaro, Eds. vol. 3670, pp. 243–256, Springer-Verlag, September 1-3, 2005.
[13] S. H. Ripon, "Process algebraic support for web service composition," *SIGSOFT Softw. Eng. Notes*, vol. 35, no. 2, March 2010.
[14] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1985.
[15] J. Magee and J. Kramer, "Concurrency: State models and java programs," Text Book, 2nd Edition, John Wiley & Sons, Ltd., 2006.
[16] J. Magee, "Behavioral analysis of software architectures using LTSA," in Proc. the 21st international conference on Software engineering (ICSE '99), ACM, New York, NY, USA, pp. 634-637, 1999.

Shamim Ripon is an Associate Professor in the Department of Computer Science and Engineering, East West University, Dhaka, Bangladesh where he leads Software Engineering and Formal Method Research Group. Previously, he was a Research Associate in the Department of Computing Science, University of York, UK and Research Fellow in the Department of Computing Science, University of Glasgow, UK. He also served as a lecturer in Khulna University, Bangladesh.

Dr. Ripon holds a B.Sc. degree in computer science and engineering from Khulna University, MSc degree in computer science from National University of Singapore and PhD in Computer Science from University of Southampton, UK. His research interests focus on the requirement engineering, software product line, semantic web, natural language processing, data mining. His current research examines the formal representation and verification of knowledge based requirement specification.