Performance Modeling of Distributed Collaboration Services

Toqeer Israr
School of Information & Technology,
University of Ottawa
800 King Edward Avenue, Ottawa,
Ontario, Canada K1N 6N5
1 613 562 5800 x 6433
tisra051@uottawa.ca

Gregor v. Bochmann
School of Information & Technology,
University of Ottawa
800 King Edward Avenue, Ottawa,
Ontario, Canada K1N 6N5
1 613 562 5800 x 6205
bochmann@site. uottawa.ca

ABSTRACT
This paper deals with performance modeling of distributed applications, service compositions and workflow systems. From the functional perspective, the distributed application is modeled as a collaboration involving several roles, and its behavior is defined in terms of a composition from several sub-collaborations using the standard sequencing operators found in UML Activity Diagrams and similar formalisms. From the performance perspective, each collaboration is characterized by a certain number of independent input events and dependent output events, and the performance of the collaboration is defined by the minimum delays that apply for a given output event in respect to each input event on which it depends. We use a partial order to model these delays. The paper explains how these minimum delays can be measured through testing. It also provides general formulas by which the performance of a composed collaboration can be calculated from the performance of its constituent sub-collaborations and the control structure which determines the order of execution of these sub-collaborations. Proofs of correctness for these formulas are given and a simple example is discussed throughout the paper.

Categories and Subject Descriptors
D.2.8[Software Engineering]: Metrics - Performance measures

General Terms
Algorithms, Measurement, Performance, Verification.

Keywords
performance modeling, software performance, partial order, collaborations, UML Activity Diagrams, distributed applications, web services.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICPE'11, March 14–16, 2011, Karlsruhe, Germany.
Copyright 2011 ACM 978-1-4503-0519-8/11/03...$10.00.

1. INTRODUCTION
Various kinds of models are used for a system design and development process. Amongst several notations, some are UML Activity Diagram[15], Use Case Maps (UCM)[5], the Process Definition Language(XPDL), Business Process Execution Language (BPEL), Web Services Choreography Description Language (WS-CDL) [19] and Petri Nets. All these mentioned notations can potentially be decomposed into sub activities and further into sub-sub-activities. Most of these notations, though, assume the basic activities in the decomposition to be allocated to a single system component. However, most of the applications have activities which are modeling collaborations between several system components, for instance an interaction between a client and a server. To this end, a modeling paradigm based on collaborations has been proposed [1] which can be represented through a combination of UML Collaboration diagrams and a partially ordered set of input and outputs.

While the realization of distributed designs for such collaboration services often pose tricky questions for the correctness of the required communication protocols in terms of the messages being exchanged between the different system components participating in the realization of the distributed service (see for instance [1] and [6]), we concentrate in this paper on the performance aspects of such collaborations. We use the concept of partial orders to model the temporal relationships between the different sub-activities within a collaboration, and use ideas from the PERT (Project Evaluation and Review Technique) technique [8] commonly used for project management. We also build on a testing technique developed for systems that implements a behavior defined in terms of a partial order relationship between input events and output events [4], and show how a similar technique can be applied for performance analysis of distributed applications. We base our analysis on the collaboration roles involved and we make the assumption that each role can be implemented by a multi-threaded component.

The paper is structured as follows. In Section 2, we review the modeling paradigm based on collaborations and introduce an example. This section also describes the rules that underlie the concepts of strong and weak sequencing. In Section 3, we discuss how such system models can be modeled using partial orders. In Section 4, we introduce performance considerations, mainly

related to the delays implied by the execution of the different sub-activities within collaborations. A general timing constraint is established which determines the earliest time that a given output event of a collaboration can be produced. Inspired by the testing procedure of [4], we will also show how the parameters determining the performance of a collaboration can be measured. In Section 5, we propose and prove formulas to calculate the performance of composite collaborations, and applied to a simplified version of the running example introduced earlier.

2. MODELING DISTRIBUTED COLLABORATION SERVICES

An example of Cab Dispatcher System (CDS) is presented in Figure 1.0. It consists of 5 sub-collaborations and 3 roles – client, dispatcher and cab. For each of these roles, there exists an input event (client requesting cab, dispatcher being available, and cab being available) and an output event (client reaches their destination and makes payment to driver, cab signs off, and dispatcher signs off). These input and output events are initiating and terminating events, respectively of this CabDispatcher collaboration. Initiating event[1] represents the starting of an action in the collaboration, for which there are no other actions in that collaboration that precedes that action. Similarly the terminating event [1] represents the end of an action in a collaboration, for which there is no other action in a collaboration that succeeds this action in that flow of execution.

These initiating and terminating events should not be confused with the starting and the ending events. The starting event represents the starting of an action for a given role, for which there are no other actions in that collaboration for that role, that precedes that action. Similarly, an ending event represents the end of an action for a given role, for which there are no other actions in that collaboration for that role that succeeds that action. A starting(/ending) event can be an initiating(/terminating) event but does not have to be whilst an initiating(/terminating) event has to be one of the starting(/ending) events.

In order to define the behaviour of a collaboration, we use Bochmann's [1] diagram notation, based on UML Activity Diagram as shown in Figure 1.0. For each of these sub-collaborations, initiating events are represented by dark dots "●" while terminating events are represented by thick vertical bars " ▌". If an event is an initiating (/terminating) aspect and a starting(/ending) event, then only the initiating(/terminating) event) is modeled on the diagram. More discussion on events will follow in context of partial ordering. We also note that the initial node not only models the start of all the flows in a collaboration, but also represents the starting events of all the roles involved.

We consider the example of cab dispatcher system. The client (Cl) requests a cab from the dispatcher (Di). The cab, when available, comes in and the dispatcher adds the cab to the queue of available cabs. The dispatcher also concurrently services the client's requests. The client, while waiting for the cab, has the option to cancel the cab. Once there is a cab in the queue, the dispatcher assigns the cab to the client and goes back and waits for more cabs and clients. The dispatcher does this for *n* times and after that, the dispatcher ends the shift. Meanwhile, the cab takes the client to the destination and is paid for the services rendered, upon which the sequence for the cab and for the client then ends. This system can accept new clients and drivers, which would be serviced as long as the dispatcher continues to loop and execute.

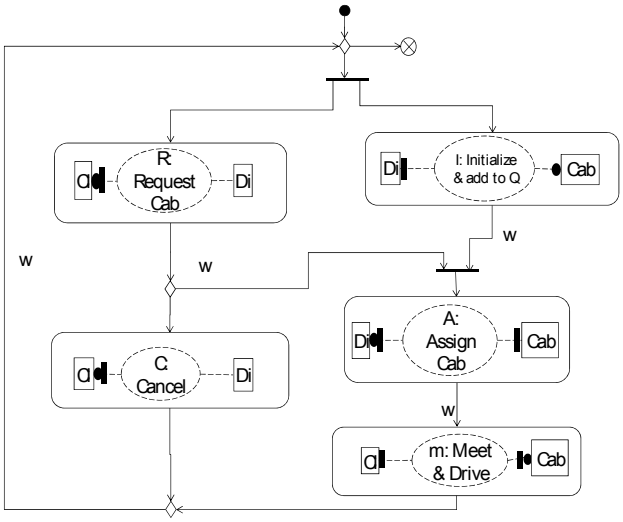

Figure 1.0 – Collaboration of Cab Dispatcher System

3. DESCRIBING COLLABORATIONS WITH PARTIAL ORDERS

3.1 Partial Orders

Inspired by Partial Order Input Output Automata (POIOA) [4], an extension of Input Output Automata, we introduce Partial Order Systems (POS). In a POS, each collaboration has a set of independent initiating events, which trigger the execution of some internal actions resulting in a set of output events. A Partial Order System realizes this dependency relationship between a set of input and a set of output events.

3.1.1 POS of a Single Collaboration

For a given collaboration, each initiating and starting event represents an input event of a POS. Each terminating and ending event represents an output event of a POS. We use the previously discussed notation to model initiating input event (dark dot "●"), starting input event (unfilled dot "o"), terminating output event (thick vertical bar " ▌"), and ending output event (thin vertical bar "|") in a POS. The output events are not ordered relative to one another directly but each output has a dependency on the inputs as indicated by the arrows "→". There also exists a set of derived dependency amongst the input and the output events relative to each other also, shown by the dashed arrow. This dependency may exist between an initiating event and an ending event of two different roles.

Consider the *AssignCab* sub-collaboration in Figure 2.0a, involving 1 initiating input event (initiated by dispatcher), and 2 terminating output events (cab getting assigned and dispatcher updating the cab available queue). Event AO_1' can only occur after the event AI_1 has occurred, but for the event AO_2' to occur, both events, AI_1 and AI_2, must occur first. We can write this as ($AI_1 \rightarrow AO_1$'), ($AI_1 \rightarrow AO_2$') and ($AI_2 \rightarrow AO_2$').

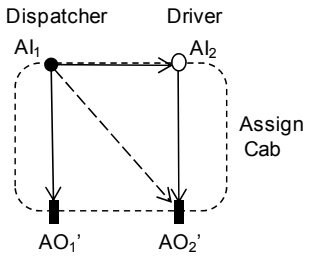

Figure 2.0a – POS of the Assign Cab sub-collaboration

3.1.2 Modeling with Partial Orders

In Fig 1.0, *AssignCab* is weakly sequenced with *Meet&Drive*, which is abstracted by *ClientServing* collaboration in Fig 2.0b. This helps us realize when each role is available, before and after the weak sequence operation.

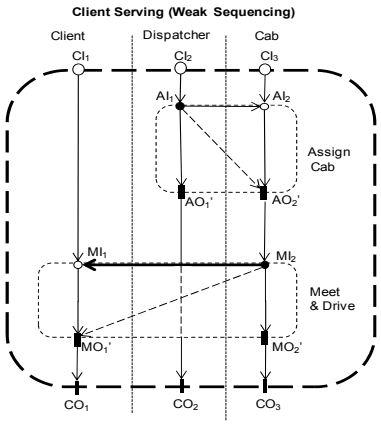

Figure 2.0b – POS of 2 Weakly Sequence Collaborations

In the *ClientServing* collaboration of Figure 2.0b, *AssignCab* shall execute as soon as inputs are available for Cab and Dispatcher. Since Client is involved only in *Meet&Drive*, the Client could potentially start executing *Meet&Drive* as soon as a Client input is available.

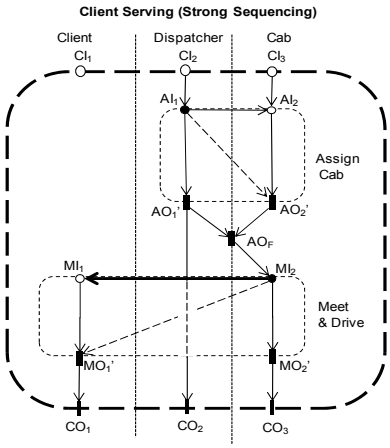

Figure 2.0c – POS of two Strongly Sequenced Collaborations

Since *AssignCab* and *Meet&Drive* are weakly sequenced, local ordering applies amongst the roles involved. Hence, the cab can start as soon as it completes *AssignCab.* As soon as execution of the dispatcher completes in *AssignCab,* it does not wait for the execution of *Meet&Drive* and it can start its next collaboration after *ClientServing.* The Client could have potentially started *Meet&Drive* before the completion of *AssignCab*, but it could not since Cab is the only initiating role in *Meet&Drive* and hence Client has to wait for the cab to start *Meet&Drive* before it starts its execution.

In Figure 2.0c, we model the same two sub-collaborations, but now we assume they are strongly sequenced. As discussed before, this means that all the initiating events in *Meet&Drive* will occur only all the terminating events of the previous collaboration, *AssignCab*, have occurred. We call this condition Final Action event – when all the terminating events have occurred, denoted by AO_F (Final Output of sub-collaboration *AssignCab*). The time of all of the initiating events for the next collaboration, *Meet&Drive*, is the time of the Final Action event of the previous collaboration, *AssignCab*.

3.1.3 Modeling Alternatives with Partial Orders

Having an operator to model alternatives is a very common and a useful concept in UML. However, we had to improvise as Partial Ordering does not allow modeling of alternative paths. Inspired by the choice symbol in UCM[5], we introduce a new symbol in Partial Ordering to represent a choice in a system. As can be seen in Figure 2.0d, it is a rectangular box with multiple branches stemming out, with each branch having an event associated with it.

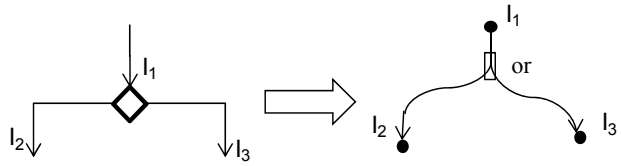

Figure 2.0d – Choice representation in POS

3.2 Modeling Cab Dispatcher System with Partial Orders

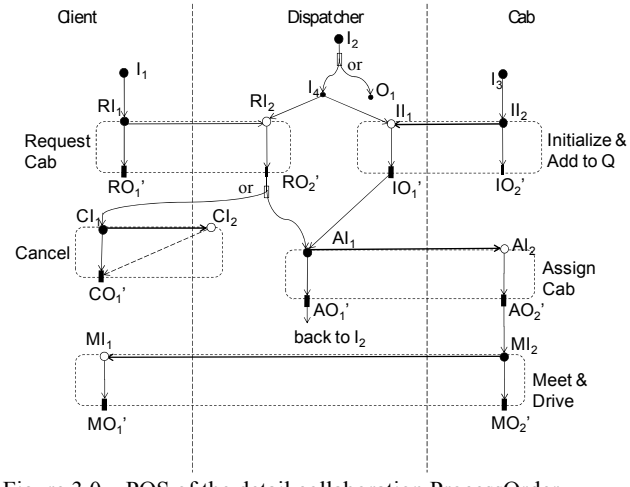

Figure 3.0 – POS of the detail collaboration ProcessOrder

In Figure 3.0, the Cab Dispatcher System is modeled as a Partially Ordered System. From Figure 1.0, we know that there are three roles involved, driver, dispatcher and the client, resulting in 3 initiating events, I_1, I_2, and I_3 as shown in Figure 3.0. Event I_1

leads to event RI_1 and event I_3 leads to event II_2, which are initiating the *RequestCab* and *Initiatlize&AddtoQ* collaboration respectively.

For the *RequestCab* sub-collaboration POS, RI_1 and RO_1' are the initiating and the terminating events respectively. Events RI_1 and RI_2, both, are the starting events also for the Cab and Dispatcher respectively, while RO_1' and RO_2' are the ending events for the Cab and Dispatcher roles respectively. Hence, there are dependencies between the two inputs, RI_1 and RI_2, i.e. ($RI_1 \rightarrow RI_2$). Also, the output event RO_1' will occur only after RI_1 due to local ordering and same is true for RO_2 and RO_2', i.e. (RO_1' $\rightarrow RI_1$) and ($RO_2 \rightarrow RO_2$').

4. PERFORMANCE CHARACTERISTICS OF PARTIAL ORDER SPECIFICATIONS

Bochmann et al., in [4] describe a method for testing a given Partial Order Input/Output Automata (POIOA) in order to determine whether this automaton realizes the partial-order relationships between input and output events as defined by a behaviour specification, also given in the form of a POIOA. This testing method proceeds as follows (adapted here to the simplified case where all input events are independent of one another). We assume that the set of input events is IS and the set of output events OS.

For each input event $i_x \in$ IS, perform the following two steps:

Step 1: Apply all inputs $i_y \in$ IS \ { i_x } and observe all outputs that will occur. Call this set OS1.

Step 2: Apply input i_x and observe all outputs that will occur. Call this set OS2.

Note: All outputs in OS1 have no partial-order dependency on input i_x, while the outputs in OS2 have a partial-order dependency on input i_x. The union of OSI and OS2 should be equal to OS.

We propose to extend this testing procedure by collecting at the same time some performance measurements, as follows: For the occurrence of each output o_m in the set OS2, we measure the delay between the time instance of the occurrence of input i_x and the occurrence of the output o_m . We call this delay the Minimum Execution Time Delay (METD), written Δ^{ix}_{om}, for the tested automaton.

If the inputs are generated in an arbitrary order with arbitrary timing, it is clear that the time delay between the execution of an input i_x and an output o_m that depends on the input can never be smaller than the METD Δ^{ix}_{om} measured during the above testing procedure. We therefore have

$t_{om} <= t_{ix} + \Delta^{ix}_{om}$ where t_{om} and t_{ix} are the time instants of execution of t_{om} and t_{ix} respectively.

Considering that this relationship must be satisfied for all inputs on which t_{om} depends, we get the

$$t_{om} = \max_x (t_{ix} + \Delta^{ix}_{om}), \quad (1)$$

where we have assumed that the METD will actually be attained during this execution scenario, which may not be realistic if shared resources are involved in the processing of several inputs on which the output depends.

The above formula shows that the performance characteristics of the partial-order specification can be given by stating for each output, the values of the METD Δ^{ix}_{om} in respect to the inputs on which that output depends.

5. DERIVING GENERAL FORMULAS FOR STANDARD SEQUENCING OPERATORS

Bochmann and his group in [1], has developed and implemented a methodology to derive component designs from global service and workflow specifications based on common sequencing operators described in UML Activity Diagrams and High-Level MSCs such as weak sequence, strong sequence, weak while loop, strong while loop, alternatives and parallel. Each of the role's behaviour is derived using transformation rules based on the sequencing operators.

We derive the performance of the global collaboration based on the performance of each sub-collaboration, each sub-collaboration represented as a partial order system. We wish to derive the general performance formulas which we can apply to the sequencing operators of sub-collaborations to yield the performance metrics of a global collaboration.

We already have devised a methodology to measure the METD, Δ^{ix}_{om}, which is used in (1) to calculate t_{om}. In the next section, we propose and provide proofs for the various definition of Δ^{ix}_{om} for collaborations composed of sub-collaborations sequenced with the above mentioned sequencing operators.

5.1 Strong Sequence

Proposition 1.0:
If two sub-collaborations, sub-collaboration A with k inputs and k' outputs and sub-collaboration B with h inputs and h' outputs, are strongly sequenced and the METD for each sub-collaboration (Δ^{AIx}_{AOy} and Δ^{BIs}_{BOm}) are known, then the METD for the composite collaboration(Δ^{AIx}_{BOm}) are:

$$\Delta^{AIx}_{BOm} = \max_{y=1..k'}(\Delta^{AIx}_{AOy}) + \max_{s=1..h}(\Delta^{BIs}_{BOm}) \quad (2)$$

Proof:
Figure 4.0 shows strong sequencing between two collaborations A and B and its resultant POS as discussed in Section 3.2.1. We assume Δ^{Aix}_{AOy} and Δ^{BIs}_{BOm} is known, either given or measured using methodology from section 4.2.

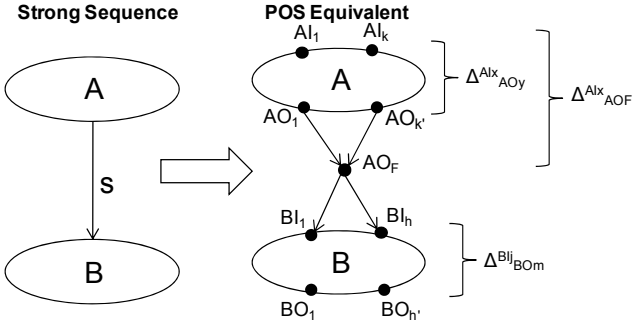

Figure 4.0 – Analysis of Strong Sequence operator

The delay from the Final Action event (AO_F) relative to the input events of A, has the value:

$$\Delta^{AIx}_{AOF} = \max_{y=1..k'}(\Delta^{AIx}_{AOy}) \quad (3)$$

because it is, as defined in Section 3.2.1, the maximum delay between each of the output events relative to A's input events.

The METD for collaboration B for an input s to an output m is Δ^{BIs}_{BOm}. To calculate the maximum it would take to produce the output m, we take the maximum over all the given inputs:

$\Delta^{BIj}_{BOm} = \max_{s=1..h}(\Delta^{BIs}_{BOm})$ (4)

Now it is clear from the right side of Figure 4.0, that the total time delay Δ^{AIx}_{BOm} is the sum of Δ^{AIx}_{AOF} and Δ^{BIj}_{BOm}:

$\Delta^{AIx}_{BOm} = \Delta^{AIx}_{AOF} + \Delta^{BIj}_{BOm}$ (5)

$= \max_{y=1..k'}(\Delta^{AIx}_{AOy}) + \max_{s=1..h}(\Delta^{BIs}_{BOm})$ (6)

5.2 Weak Sequence

Figure 5.0 shows weak sequencing between two collaborations A and B, hence local synchronization between the roles of collaboration A and B. Note, not necessarily all the roles which have an output event in collaboration A will have an input event in collaboration B. We assume that roles 1 to g are participating in both collaborations A and B and roles g+1 to h are participating only in collaboration B, but not in A. We further assume there is no dependency of any sort between any of the roles in collaboration B and the roles g+1 to h in collaboration A. Compared to strong sequencing, we consider not only initiating and terminating events, but also the starting and the ending events for a given role. For the roles involved only in collaboration B and not in collaboration A, we assume the input events for these roles occur very early and any actions depending only on these events have completed.

We are interested in the METD between the output events of collaboration B relative to the input events of collaboration A.

Proposition 2.0:
As shown in Figure 5.0, if two collaborations, A and B, are weakly sequenced, then the minimum execution time delay for the collaboration C is:

$\Delta^{AIx}_{BOm} = \max_{s=1..g}(\Delta^{AIx}_{AOs} + \Delta^{Bis}_{BOm})$ (7)

Proof:
As discussed in Section 4.2, the METD of collaboration A is Δ^{AIx}_{AOy} and collaboration B is Δ^{Bis}_{BOm}.

From (4), we know the METD from any of the inputs to output m of collaboration B is $\Delta^{BIj}_{BOm} = \max_{s=1..h}(\Delta^{BIs}_{BOm})$.

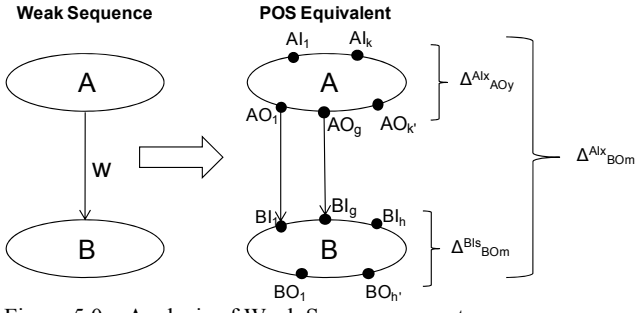

Figure 5.0 – Analysis of Weak Sequence operator

It is quite clear from the right side of Figure 5.0, that the METD of the composed collaboration is the sum of the METD of both collaborations A and B.

As this is not strong sequencing, there is no Final Action Event to synchronize the outputs of collaboration A. To calculate the METD for the composed collaboration, note role *s* has to be the terminating role of collaboration A and also the starting role of collaboration B:

$\Delta^{AIx}_{BOm} = \max_{s=1..g}(\Delta^{AIx}_{AOs} + \Delta^{Bis}_{BOm})$ (8)

5.3 Generalization to Performance Distributions

In the performance analysis, throughout this paper, we have only considered a fixed time for the time instants and for time delays. However, reality begs to differ. Realistic scenarios also include time delays and time instants which that may vary and could be characterized by some kind of distribution – perhaps binomial, exponential, etc. The distributions of the Minimum Execution Time Delays can be measured by performing the Test Suite of Section 4.3 several times and obtaining some statistics about the possible values. The properties of the distribution can then be realized by analyzing the resultant data, for each collaboration. If the distribution of these time delays for each sub-collaboration is known, we can then calculate the time delays for a composition of these sub-collaborations. For the sequential execution of two sub-collaborations, the folding operation on the respective distribution can be used to obtain the distribution of the overall execution delay, which is easily evaluated in the case of Normal distributions. However, the determination of the distribution of the maximum of two existing distributions is more complex. We do not discuss these issues any further in this paper.

[8] discusses Project Evaluation and Review Technique (PERT), a methodology for planning and scheduling interrelated tasks in a large system. PERT is a concept similar to our work here, except for some differences. Its basic idea is to optimize time and resource-constrained systems. The idea is based on building a network model where the time delays are known, a concept very similar to Dijktra's shortest route algorithm. PERT methodology determines the path to the final goal with the maximum execution time and hence, the time instant when the goal can be reached. In order to deal with the distribution of execution times in the real world, PERT may consider the minimum, maximum and most probable execution time for each subactivity and, as a consequence, would be able to determine the minimum, maximum and most probable overall delay for reaching the goal.

6. FUTURE WORK

We note only the basic strong and weak sequence operation of Bochmann's [1] sequencing operators were analyzed for performance. We would like to consider the remaining operators, such as alternatives, parallel, loops(weak/strong), and the interruption operator. An asset would also be an illustration of this methodology with a practical example.

The possibility of using distributions for characterizing the time delays of collaborations would also be an interesting extension of this work, as mentioned in Section 5.3. Perhaps, our work would benefit from analyzing complex systems with distributed time delays using methodologies such as Continuous Time Markov Chains.

Implementation of the here proposed testing methodology and the performance derivation in a tool environment and to extend the algorithm for any possible scenarios which our work does not support will also be the next logical step.

7. CONCLUSION

We use Bochmann's [1] method of representation to model collaborations and analyzing various scenarios. We proposed a partial order representation to model these collaborations to help us with the performance analysis of a distribution system. We proposed a general formula for a collaboration and proposed a set of formulas for various standard sequencing operators to derive the performance of a global collaboration given a set of sub-collaborations sequenced with these sequencing operators.

We believe that this approach to the performance modeling of distributed system designs is useful in many fields of application, including distributed workflow management systems, service composition for communication services, e-commerce applications, and Web Services.

We plan to work on the tool support for the proposed testing methodology and performance analysis. As well, we will also be extending our set of formulas to support various operators not considered in this paper as well as distributions for time delays.

Acknowledgements: We would like to thank Dr. Guy Jordan for many interesting discussions on the problems and issues related to this paper.

8. REFERENCES

[1] Bochmann, G.V. Deriving component designs from global requirements, in: Proceedings on International Workshop on Model Based Architecting and Construction of Embedded Systems (ACES), Toulouse, 2008, pp 55-69

[2] Bochmann, G.V. A General Transition Model of Protocols and Communication Services, IEEE Transtions on Communications COM-28, April 1980, pp 643-650

[3] Bochmann G.V. and Gotzhein, R. Deriving protocol specifications from service specifications, Proc. ACM SIGCOMM Symposium, 1986, pp 148-156

[4] Bochmann, G.V., Haar, S., and Jourdan, G.V. Testing Systems Specified as Partial Order Input/Output Automata, Proceedings of the 20th IFIP TC 6/WG 6.1 International Conference on Testing of Software and Communicating Systems: 8th International Workshop, Tokyo, 2008, pp 169-183

[5] Casselman, R. Use Case Maps for Object-oriented Systems, Prentice Hall, New Jersey, 1995

[6] Castejón, H.N, Bræk, R., and Bochmann, G. v., "Realizability of Collaboration-based Service Specifications". Proc. Of the 14th Asia-Pacific Software Engineering Conference (APSEC 2007), IEEE Computer Society, December 2007

[7] Castejón, H. N., Bræk, R., Bochmann, G.v., Realizability of Collaboration-based Service Specification. in Asia-Pacific Software Engineering Conference, Nov. 2007 pp73-80

[8] Chinneck, J., Practical Optimization: A Gentle Introduction, online textbook, see http://www.sce.carleton.ca/faculty/chinneck/po.html.

[9] Esparza, J. and Heljanko, K. *Unfoldings - A Partial-Order Approach to Model Checking*, New York: Springer-Verlag, 2008

[10] Grabiec, B, Traonouez, L., Jard, C., Lime, D and Roux, O.H. Diagnosis using unfoldings of parametric time petri nets. in 8th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS 2010), September 2010, Vienna, Austria. pp 137-151

[11] Kounev, S. Performance Modeling and Evaluation of Distributed Component-Based Systems Using Queueing Petri Nets, IEEE Transactions on Software Engineering, v.32 n.7, July 2006, p.486-502

[12] McMillan, K. Using Unfoldings to Avoid the State Explosion Problem in the Verification of Asynchronous Circuits, in Proceedings of the Fourth International Workshop on Computer Aided Verification, 1992, p.164-177

[13] Merlin, P. and Faber, D.J. Recoverability of communication protocols, IEEE Transaction Communication, vol. COM-24, no. 9, Sept. 1976

[14] Naumovich, G., Clarke, L. and Cobleigh, J. Using partial order techniques to improve performance of data flow analysis based verification. In Proceedings of the ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, Toulouse, France, Sept. 1999.

[15] OMG, Unified Modeling Language (UML), Version 2.1.1, February 2007

[16] Razouk, R.R. The Derivation of Performance Expressions for Communication Protocols from Timed Petri Net models, ACM SIGCOMM Computer Communication Review, v.14 n.2, pp 210-217, June 1984

[17] Sifakis, J. "Use of Petri Nets for performance evaluation"; in Measuring, modeling and evaluating computer systems, North-Holland 1977, pp 75-93

[18] Wolper, P. and Godefroid, P. Partial-order Methods for Temporal Verification. In Proc. 4th Int. Conference on Concurrency Theory (CONCUR), volume 715 of Lecture Notes in Computer Science, Springer, 1993. pp 233–246

[19] W3C, Web Services Choreography Description Language (WS-CDL), Version 1.0, December 2004