Performance Modeling of Distributed Collaboration Services with Independent Inputs/Outputs

Toqeer Israr, Gregor v Bochmann

Department of Electrical Engineering and Computer Science
University of Ottawa
800 King Edward Ave. Ottawa Ontario K1N 6N5 Canada
{tisra051, bochmann}@eecs.uottawa.ca

Abstract. This paper deals with modeling and performance analysis of distributed applications, service compositions and workflow systems. From the functional perspective, the distributed application is modeled as an activity involving several roles, where behavior is defined in terms of compositions from several sub-activities using the standard sequencing operators found in UML Activity Diagrams. Each activity is characterized by a certain number of input and output events, and the performance of the activity is defined by the minimum delays that apply for a given output event in respect to each input event. We use a partial order to model these events, whose delays can be measured through testing. We also provide general formulas to calculate the performance of a composite activity from the performance of its constituent sub-activities and the control structure specifying the order of execution. Proofs of correctness for these formulas, along with a simple example are also given.

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

1 Introduction

Many commercial systems rely on multiple communicating components for parts of their business processes. These are often structured as distributed systems, with components running on different processors or in different processes. For example, a multi-tiered system might start with requests from Web clients that invoke a process in a Web Application Server, which in turn makes calls to some "third party" servers – components involved in this system would be the client, Web-Application Server and the "third party" servers.

When developing such distributed reactive systems, there is a need to analyze performance of both from a global system perspective and a local or component-wise perspective. The global perspective specifies and analyzes the collaborative behavior of a distributed system in an abstract manner, while the local perspective identifies different system components along with their behaviour such that their interactions give rise to a behaviour satisfying the global perspective. Numerous methodologies have been employed for analyzing such systems, such as Queuing Models, Stochastic

Petri Nets (SPN), Performance Evaluation and Review Technique (PERT) [4], and UML Profile: Modeling and Analysis of Real Time Embedded Systems (MARTE). 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.

Such distributed reactive systems are typically accompanied by performance specifications which are generally formalized by legal Service Level Agreements carrying financial penalties for non-compliance. Hence, it is becoming a commercial imperative to ensure that the participants in a workflow meet certain time criteria under all possible scenarios. McNeile in [6] modeled and analyzed end-to-end workflow delays but their analysis assumed that all the components are available at the beginning of the workflow, yielding a single workflow delay. This is not realistic as components become available at different time. This leads to relationships and delays between outputs and inputs of various components.

To this end, we introduced Partially Ordered Specification (POS) in [5], a modeling paradigm composed of UML Activity diagrams [7] and a partially ordered set of inputs and outputs. In this paper, we revise and extend the existing work done in [5] by considering performance characteristics of composite activities with **independent** inputs and outputs. We analyze and derive formulas for composite activities, composed of sub-activities with concurrency and alternatives.

We start off by reviewing modeling with partial orders. In Section 3, we review some of our previous work from [5] to describe POS and performance characteristics of basic operators. In Section 4, we propose formulas and provide proofs for calculating the performance of composite collaborations, composed with **alternate** and **concurrency** operators for **independent** input and output events.

2 Modeling Distributed Collaboration Services

2.1 Modeling Events with Partial Orders

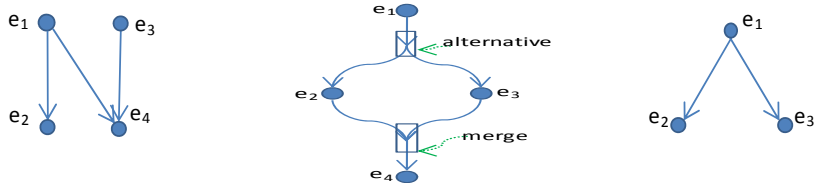
In [5], we adapted the modeling methodology from [4] to model inputs and outputs of UML Activities (Collaborations) [7] as partially ordered events. We extend the previous work done in [5] by modeling events in sequence, alternatives and in a concurrent manner.

Modeling Events in Sequence with Partial Orders: A set of events may have a dependency on other events, and hence cannot occur until all the required events have occurred. Such is shown in Figure 1.0a, where event e_4, is shown to be dependent on e_1 and e_3, while e_2 is only dependent on e_1.

Modeling Alternatives with Partial Orders: As partial ordering does not allow modeling of alternative paths, inspired by the choice symbol in UCM [2], we introduce a new symbol in partial ordering to represent a choice in a behavior. As can be seen in Fig 1b, a UML "decisionNode" is modeled as a rectangular box with one incoming edge and multiple outgoing edges. The UML "mergeNode" is modeled as a similar rectangular box with multiple incoming edges and one outgoing edge.

Modeling Concurrency with Partial Orders: As illustrated in Fig 1c, there is a

single event e_1 which leads to two events e_2 and e_3. Both events e_2 and e_3 must occur after e_1, but they are incomparable between one another.



a) Sequential b) Alternative c) Concurrency

Fig. 1. Modeling the Ordering of Events Using Various Operators

2.2 Describing Collaborations with Partially Ordered Specifications

Partially Ordered Specifications (POS) rely on UML Activity Diagrams (AD) to capture the dynamic behaviour of a system. This is comprised of actions and sequencing operators such as sequence, alternative, concurrency, and loops to define the relationship between these actions.

For a given activity, we consider input and output events. Input (/output) events, shown as unfilled circles in Fig 2, are events which mark the beginning (/ending) of the execution of actions by a specific role in a given activity. Note that a given activity has an input and an output event for each involved role.

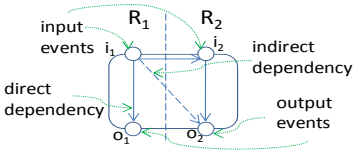


Fig. 2. Partial Order Events

These events form a partially ordered set, where a causal relationship may exist between some of these events, shown by arcs "→". The output events are not ordered relative to one another directly but each output event has a dependency on some corresponding input events.

Figure 2 illustrates an activity, with input events i_1 and i_2 and output events o_1 and o_2. As i_1 and o_1 are input and output events of the same role R_1, o_1 must occur after i_1 due to local sequencing. Due to the relationship $i_1 \rightarrow i_2$ and $i_2 \rightarrow o_2$, there is an indirect dependency from i_1 to o_2, shown by the dashed arrow "-->." Output events o_1 and o_2 are incomparable and may occur in any order.

2.3 Delay for a Given Activity

We devised an approach to determine the dependencies amongst the input and output events within a given sub-activity in [5]. For a given sub-activity, according to this approach, one measures the delay between the time instance of the occurrence of input event i and a dependent output event o, provided all the other events on which o depends have occurred long time before. This delay is called Nominal Execution Time Delay (NETD), written as Δ^i_o.

This leads to the following formula which yields the performance of a collaboration D based on dependencies between input and **dependent** output events:

$$_D T_o = \max_{i \in I(D)} (_D T_i + _D \Delta^i_o) \qquad (1)$$

where T_o is the time of the output event o, T_i is the time of the input event i, Δ^i_o is the NETD from input event i to the dependent output event o and I(D) is the set of input events. Subscript "D" indicates all the notations are for the abstract activity D.

3 Performance Characteristics of Composite Activities

So far this paper provides a somewhat summarized version of modeling and performance analysis of composite activities given in [5]. The remainder of this paper is new material analyzing performance of activities composed of alternative and concurrent sub-activities with independent input and output events.

3.1 Independent Events

Formula (1) was derived to calculate the time of the output events based on the time of the input event and the NETD that exists between them, provided the output events depended on input events. We can relax this condition by revising the definition of (1) to include all events, by stating that the NETD between an input event i and a non-dependent output event o' is: $${}_A\Delta^i_{o'} = -\infty \quad (2)$$

Then the formula in (1) is not limited to dependent events, but rather is valid for all involved events – **dependent and independent events** alike. Note, for an input and output event of the same role, the output event cannot occur until its input event occurs (also known as local sequencing). Hence for a given output event, at least one input event (input event of the same role) will always exist which will make (1) yield a positive value.

3.2 Consideration of Control Flow Paths

We define **control flow paths *(cp)*** to depict a single execution of a given system. As an activity may have alternatives, interruptions and loops, this causes multiple control flow paths to exist. It is clear that different control flow paths, in general, lead to different execution time delays. For instance, the different branches of an alternative may result in different delays. In general, the number of different control flow paths is unlimited. For instance, the number of times a while loop executes may be unbounded, and/or the body of a loop or alternative may include other loops or alternatives, resulting in a recursive structure.

Therefore the Nominal Execution Time Delay (NETD) defined above depends on a particular control flow path that was followed during the execution of the collaboration. Throughout this paper, we only consider a single particular path, say *cp*. Then we write ${}^{(cp)}{}_A\Delta^i_o$ for the Nominal Execution Time Delay between output event o and input event i for the control flow path *cp* of collaboration A.

We recognize the delays can be of a stochastic nature. However, for our work, we consider only **fixed delays** while considering a single control flow path.

Shared Resources: We have assumed that the NETD will actually be attained during a control flow path, which may not be realistic if shared resources are involved in the processing of several inputs on which a single output depends. Hence we assume in

the following that there are no shared resources and each role or all concurrent activities of a given role are implemented by an independent processor.

3.3 Basic Performance Characteristics

Events can be combined using various operators such as sequences, alternatives, concurrency, loops etc. We can define the basic performance characteristics involving partially ordered events.

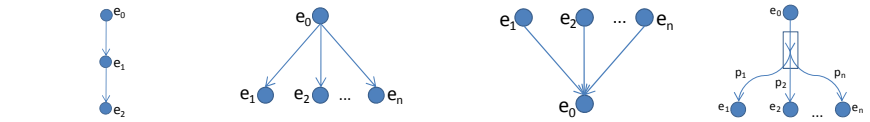
Sequence: Single Event (SE) Dependent on a Single Event (SE). Figure 3a shows three events in a sequence where the dependency $e_1 \rightarrow e_2 \rightarrow e_3$ exists. If the delays between these events are considered to be fixed delays, then it is quite clear that the delay from e_1 to e_3 (Δ^{e1}_{e3}) is the sum of the delays from e_1 to e_2 (Δ^{e1}_{e2}) and e_2 to e_3 (Δ^{e2}_{e3}):

$$\Delta^{e1}_{e3} = \Delta^{e1}_{e2} + \Delta^{e2}_{e3} \quad (3)$$

Concurrency: Multiple Events (ME) Dependent on a Single Event (SE). If there are multiple events dependent on a single event such as shown in Figure 3b, we examine two sets of delays – earliest event *y* and latest event *z* amongst $e_1...e_n$. The earliest (/latest) event amongst $e_1...e_n$, is an event for which there are no other events which precedes (/succeeds) this event amongst $e_1...e_n$. The delay to these events from event e_0 can be calculated by considering the event *y (/z)* amongst all the events, which has the minimum (/maximum) delay from the input *i*:

Earliest event:
$$\Delta^{i}_{y} = \min_{o \in \{e1,e2..en\}} (\Delta^{i}_{o}) \quad (4)$$

Latest event:
$$\Delta^{i}_{z} = \max_{o \in \{e1,e2..en\}} (\Delta^{i}_{o}) \quad (5)$$



a. SE → SE **b)** SE → ME **c)** ME → SE **d)** SE →AE

Fig. 3. Various Dependencies

Merge: Single Event (SE) Dependent on Simultaneous Multiple Events (ME). Figure 3c shows a single event e_o dependent on multiple events $e_1...e_n$. Event e_0 can only occur when all of its dependencies are satisfied i.e. events $e_1...e_n$, all, have occurred. If we assume that all the events $e_1...e_n$ occur at the same time, then we can calculate the delay for e_0 to occur by taking the maximum of all the individual NETDs:

$$\Delta^{i}_{o} = \max_{j \in \{e1,e2..en\}} (\Delta^{j}_{o}) \quad (6)$$

Alternative Events (AE) Dependent on a Single Event (SE). Figure 3d shows an alternative sequence, where the event $e_1...e_n$ occur with probability $p_1...p_n$, respectively, each having a different control flow path. However, if we consider the control flow path where a specific event e_m occurs, where $e_m \varepsilon \{e_1,..,e_n\}$, then the time delay between event *i* and event e_m, is the measured/known NETD: Δ^{i}_{m} (7)

4 Deriving General Formulas for Sequencing Operators

Notation: The set of roles involved in activity *X* is denoted by R(*X*). The set of input and output events in activity *X* are denoted by I(*X*) and O(*X*) respectively.
We use activity D to abstract the sequence of sub-activities A and B. Hence, the set of roles involved in activity D consists of all the roles involved in sub-activities A and B, that is, $R(D) = R(A) \cup R(B)$. The set of roles common to both collaboration A and B is denoted by $R^C(D)$, where $R^C(D) = R(A) \cap R(B)$. The non-common roles of A are the set of roles involved only in sub-activity A and not in sub-activity B, denoted by $R^{NC}(A) = R(A) - R^C(A)$. A similar is used for output roles.

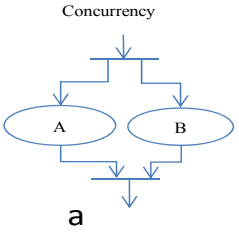
In [5], we proposed (and provided proofs for) definitions of Δ^i_o, for strong and weak sequencing operators for input and dependent output events. In the following, we extend this work by analyzing performance of sub-activity A and B using **concurrency** and **alternative** operators with **independent input** and **output** events. These compositions are abstracted by activity D.

4.1 Concurrency

Figure 4 shows concurrent execution of sub-activities A and B and the partial order equivalent. For the non-common roles, the roles become available for their next activity, as soon as execution is completed in the respective sub-activities. For the common roles, execution in both of the sub-activities must be completed before the role is available for its next activity, as shown in Figure 4b.
NETD: The NETD for a composite activity D consisting of concurrent execution of sub-activities A and B, ${}^{(cp)}_{D}\Delta^{w}_{z}$, is given by the following expressions:

Table 1. Fixed Delays for Concurrency

Case	Condition	Fixed Delays
(1)	$(w \varepsilon I^C(D)$ and $z \varepsilon O^C(D))$	$\max({}^{(cp)}_{A}\Delta^{w}_{z}, {}^{(cp)}_{B}\Delta^{w}_{z})$ (8)
(2)	$(w \varepsilon I^{NC}(A)$ and $z \varepsilon O(A))$ or $(w \varepsilon I(A)$ and $z \varepsilon O^{NC}(A))$	${}^{(cp)}_{A}\Delta^{w}_{z}$ (9)
(3)	$(w \varepsilon I^{NC}(B)$ and $z \varepsilon O(B))$ or $(w \varepsilon I(B)$ and $z \varepsilon O^{NC}(B))$	${}^{(cp)}_{B}\Delta^{w}_{z}$ (10)
(4)	$(w \varepsilon I^{NC}(A)$ and $z \varepsilon O^{NC}(B))$ or $(w \varepsilon I^{NC}(B)$ and $z \varepsilon O^{NC}(A))$	$-\infty$ (11)



a

Fig 4a. Concurrency Control Flow

Proof: We consider the proofs for all cases separately:
Case (1): As mentioned and illustrated in Figure 4b, the common role must complete its execution in both sub-activities before any subsequent executions can be performed by that role. Since all the processes of a common role become available at the same time for both sub-activities, the time of the input is the same in both sub-activities and hence from (6) we know that the NETD is the maximum of two delays.
Case (2): If the input is a non-common role of A, then the output is any role of A. As can be seen in Figure 4b, except for the delay from the input event to the output event of the common roles, the NETD is due only to the delay from the dependency input

event w to output event z. There is no other dependency which needs to be satisfied for event z to occur. Hence, the NETD for this case is NETD from w to z, ${}_A\Delta^w{}_z$. Similarly for a non-common output.

Case (3): Proof for this scenario is similar to the proof of Case (2).

Case (4): As illustrated in Figure 4b, there is no dependency from an input of a non-common role of activity A to an output event of a non-common role of activity B or vice-versa. As defined by (2), the NETD for events with no dependency is $-\infty$.

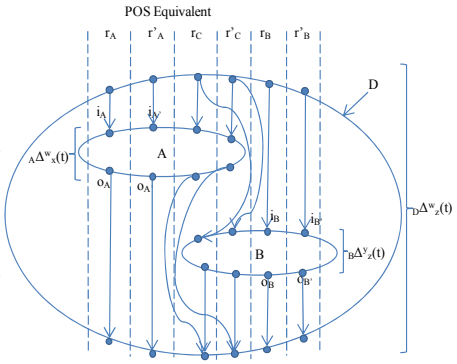


Fig 4b – POS of Concurrency

4.2 Alternatives

Figure 5 shows an alternative operator and its POS equivalent between two sub-activities, A and B. We assume the choice, made by role c*, is done instantaneously and therefore does not directly add any delays. However, no action of the sub-activities in the body of an alternative may start to execute until this choice is made. Hence, this causes a dependency between c* and all the other roles involved in the sub-activities of the alternative body, shown by the introduction of sub-activity *Choice* in Figure 5b. Note: this does not have any impact on NETD of sub-activities.

As discussed in section 3.3, there is a control flow path for each branch of an alternative, leading to different execution time delays.

NETD: The NETD for a composite activity D consisting of an alternate execution of sub-activities A and B, ${}^{(cp)}{}_D\Delta^w{}_z$, is given by the following table:

Table 2. Fixed Delays for Alternatives

Case	Condition	*Fixed Delays*	
(5)	(w ε I^C(D) and z ε O^C(D))	${}^{(ep)}{}_A\Delta^w{}_z$ ${}^{(ep)}{}_B\Delta^w{}_z$	*if A is executed* *otherwise*
(6)	(w ε I^{NC}(A) and z ε O(A)) or (w ε I(A) and z ε O^{NC}(A))	${}^{(ep)}{}_A\Delta^w{}_z$ *0* $-\infty$	*if A is executed* *if B is executed and w = z* *if B is executed and w ≠ z*
(7)	(w ε I^{NC}(B) and z ε O(B)) or (w ε I(B) and z ε O^{NC}(B))	*0* $-\infty$ ${}^{(ep)}{}_B\Delta^w{}_z$	*if A is executed and w = z* *if A is executed and w ≠ z* *otherwise*
(8)	(w ε I^{NC}(*A*) and z ε O^{NC}(*B*) or (w ε I^{NC}(*B*) and z ε O^{NC}(*A*))	$-\infty$	*if A or B is executed*

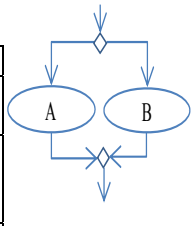


Fig5a. Alternative Control Flow

Case 5: Either sub-activity A will execute or B will. From (7) we know that depending on the sub-activity being executed, the NETD of the composite activity D will be that of the sub-activity being executed.

Case 6: If activity A executes then the delay is that of the execution in activity A. But if activity B executes, then when w = z, there is a dependency between the events due to local sequencing but there is no delay as there is no execution performed be-

tween these events. Hence the delay is 0. When w ≠ z, then there is no local sequencing dependency between the events and no delay. Hence, the delay is $-\infty$.

Case 7: This is similar to Case 6.

Case 8: This is similar to Case 4.

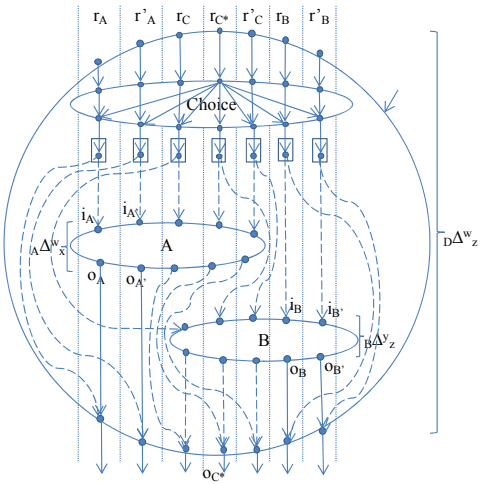


Figure 5b –POS of an Alternative

5 Conclusion

We analyzed performance of global collaborations composed from sub-collaborations with sequential, alternative and/or concurrent ordering, based on the delays of the constituent sub-activities. We believe that this approach to performance modeling of distributed systems is useful in many fields of application, including distributed work flow management systems, service composition for communication services, e-commerce applications, and Web Services. We have implemented a tool that takes as input an Activity Diagram including sub-activities with defined performance characteristics and provides as output the NETDs of the global collaboration.

In this paper, we have considered fixed delays. We plan on extending our work to consider different types of delays (stochastic and range of delays). We also plan to extend the here described work to include additional sequencing operators, such as strong and weak while loops [1] and interruptions.

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. Buhr, R.J.A., Casselman, R.S., Use CASE Maps for Object-Oriented Systems. Prentice Hall, 1996.
3. Castejón, H.N, Bræk, R., and Bochmann, G. v., On the realizability of collaborative services. Journal of Software and Systems Modeling, Vol. 10 (12 October 2011), pp. 1-21
4. Chinneck, J., Practical Optimization: A Gentle Introduction, online textbook, see http://www.sce.carleton.ca/faculty/chinneck/po.html.
5. Israr, Bochmann, Performance Modeling of Distributed Collaboration Services, Proceedings of the 2nd ACM/SPEC International Conference on Performance Engineering, Karlsruhe, Germany, 2011, pp 475-480
6. McNeile, A., "Using Motivation and Choreography to model Distributed Workflow", Proceedings of the 5th ACM SIGCHI Annual International Workshop on Behaviour Modelling - Foundations and Applications, Montpellier, France, 2013
7. Object Management Group. Unified Modeling Language Superstructure, V2.1.2. OMG Available Specification, November 2007. OMG document number: ptc/2007-11-02.