Probabilistic Time Management of Choreographies

Johann Eder[1], Horst Pichler[1], and Amirreza Tahamtan[2]

[1] Alpen-Adria University of Klagenfurt, Dept. of Informatic-Systems, Austria
{johann.eder,horst.pichler}@uni-klu.ac.at
[2] University of Vienna, Dept. of Knowledge and Business Engineering, Austria
amirreza.tahamtan@univie.ac.at

Abstract. Temporal conformance of web service compositions guarantees the timely execution of service calls, decreases follow-up costs and increases QoS by avoiding deadline violations. Since it is impossible to make certain statements about the execution intervals of upcoming web service executions - mainly due to varying activity durations and hard-to-predict branching behavior - we propose a probabilistic approach to model flow structures and temporal information, and show how to validate the temporal conformance of web service compositions.

Keywords: Web Services, Probabilistic Time Management, Choreography, Orchestration, Temporal Conformance.

1 Introduction

An essential aspect of web service composition is the conformance of web service choreographies and the orchestrations which realize a choreography[8]. In inter-organizational environments the degree of influence on when partners execute their activities can be rather limited. Therefore, it is quite important to verify temporal conformance, as it ensures that information is delivered to the right activity at the right time such that the overall temporal restrictions are satisfied. This reduces the overall process cost by avoiding deadline violations that trigger expensive exception handling routines, and increases the quality of service.

Temporal aspects of web services have been studied in [2,3,4]. [2] uses temporal abstractions of business protocols for their compatibility and replaceability analysis based on a finite state machine formalism. [3,4] exploit an extension of timed automata formalism called Web Service Time Transition System (WSTTS) for modeling time properties of web services. All these approaches apply deterministic timing concepts, but in real life it is almost impossible to make *certain* statements about the future of a process. Two factors contribute to this uncertainty, varying durations of activities and multiple execution-paths after conditional nodes. Therefore, we propose a probabilistic temporal concept.

This paper addresses the conformance and verification problem of choreographies integrating the approaches for temporal conformance checking [9] and probabilistic time management for workflows [10]. We check whether the temporal

D. Ardagna et al. (Eds.): BPM 2008 Workshops, LNBIP 17, pp. 443–454, 2009.

constraints can be satisfied and provide an a priori execution plan at build-time, consisting of probabilistic execution intervals for all activities of participating choreographies and orchestrations, considering structural and temporal restrictions.

2 Choreographies and Orchestrations

Web services can be composed by orchestrations and choreographies [7,8,1]. A choreography is an abstract process that defines the interaction between the participants. Partners of a choreography can take part in a choreography through their (internal) workflow or a view on that workflow, resp. [9]. A choreography is equally visible to all of its partners and there is no central role in charge of control. On the other hand, an orchestration is an executable process, owned and run by one partner. An orchestration is a partner's private process and solely visible to its owner. The process logic, communication actions and internal tasks are defined and executed in an orchestration. A partner can take part in a choreography and the parts of the choreography that belong to this partner are realized in his orchestration. If a partner takes part in more than one choreography, its orchestration realizes all of its parts in different choreographies.

We model both choreographies and orchestrations as workflows. A workflow can be represented by a directed acyclic graph $G = (N, E)$ which consists of a set of nodes $n \in N$ and a set of edges $n_1 \Rightarrow n_2 \in E$ (see also Fig. 1). Each node has a unique name and a type $n.type$, which may either be an activity or a control node (XOR-split, XOR-join, AND-split, AND-join). Edges connect these nodes.

In this work we model the relationship between two nodes in different choreographies or orchestrations by event correspondence. During process execution each node can be associated with two events: its start event, denoted as n_s and its end event, denoted as n_e. $e_1 \equiv e_2$ denotes that event e_1 corresponds to event e_2. Note that e_1 and e_2 may belong to nodes of different choreographies or orchestrations.

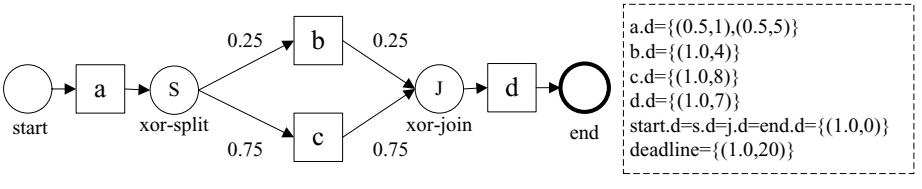

Fig. 1. A simple workflow graph

3 Probabilistic Timed Graph

The probabilistic timed graph (PTG) extends the regular process graph with explicitly defined and implicitly derived stochastic and temporal information, in particular a temporal interval for each activity that describes the time frame

within which an activity can execute, such that no deadline violations occurs. We provide a brief description of concepts and operations; for further details we refer to [6,10].

3.1 Time Histograms and Histogram Operations

Temporal information in the PTG, i.e. time points, durations, intervals, and deadlines, is represented as a multiset of (probability,time)-tuples, called time histograms [6]. For instance $a.d = \{(0.15, 10), (0.75, 12), (0.10, 20)\}$ describes the probability distribution of time values for the duration of an activity a. Time is given in form of (user defined) chronons which are the smallest indivisible units of time. Tuples with equal time values are always merged by summing up their probabilities. For the sake of simplicity we assume that all temporal information within these tuples is given in some basic time unit (like seconds). The *histogram addition* $h1 + h2$ generates the cartesian product of the tuples of two histograms, where probabilities are multiplied and time values are added. The *histogram subtraction* $h_1 - h2$ is a variation of the addition, with the difference that time values of the tuples are subtracted. The *histogram (max-)conjunction* also generates the cartesian product. Again probabilities are multiplied, but the maximum time value of each tuple-combination determines the time value of the result. A variation of this operation is the *min-conjunction* which determines the time value of the resulting tuple by applying a minimum-operation. The *weight*-operation, which multiplies all probabilities in a histogram with a given probability, appears always in combination with the *histogram disjunction*, which merges two weighted histograms (followed by an aggregation). Conjunction and disjunction are commutative and associative and can be extended to k histograms. The *histogram comparison* compares two histograms. Unlike discrete values, two histograms h_1 and h_2 may partially "overlap". Thus an expression like $h_1 < h_2$ can be true and false at the same time, each at a certain degree. The comparison of two histograms $h_1 \bowtie_{deg} h_2$ with the comparison-operator $\bowtie \in \{\leq, <, =, >, \geq\}$ for a given degree $0 \leq deg \leq 1$ is evaluated as follows:

$$h_1 \bowtie_{deg} h_2 = \begin{cases} true & : \ \Sigma p_1 * p_2 \geq deg \wedge t_1 \bowtie t_2, \forall (p_1, t_1) \in h_1, \forall (p_2, t_2) \in h_2 \\ false & : \ otherwise \end{cases}$$

3.2 Explicit and Implicit Information

Explicit information is given by the process designer: (i) the process structure represented as graph $G = (N, E)$, (ii) a duration for each graph-node $n.d$ represented as time histogram (a control node has always a 0-duration histogram $\{(1.0, 0)\}$), (iii) a (relative) deadline $G.\delta$ for the whole process, and (iv) a "branching" probability $p_{xs \Rightarrow a}$ for each edge between an XOR-split xs and an activity a; and a probability $p_{b \Leftarrow xj}$ for each edge between an activity b and an XOR-join xj. All stochastic information can be extracted from logs or must be estimated by an expert. Fig. 1 shows a small sample PTG with the deadline $\delta = \{(1.0, 20)\}$.

Table 1. Forward calculation rules for e-histograms per node type

type of v	$v.eps =$	$v.epe =$
start	$\{(1.0, 0)\}$	$v.eps + v.d$
end,activity,and-split,or-split	$pred.epe$	$v.eps + v.d$
and-join	$\bigwedge_{max}(pred.epe), \forall pred \in v.Pred$	$v.eps + v.d$
or-join	$\bigvee(pred.epe * p_{pred \Rightarrow v}), \forall pred \in v.Pred$	$v.eps + v.d$

Implicit Information is based on explicit information: four implicit time constraints can be calculated for each node. (i) The earliest possible start (eps) of an activity, denotes the earliest point in time it may possibly start. In the simplest case this will be the sum of durations of all nodes between this node and the start node (transitive predecessors). (ii) The earliest possible end (epe) identifies the earliest point in time a node may end (= eps + duration). (iii) The latest allowed end of a node (lae), is the latest point in time a node may end, such that finishing the whole process within its deadline is still possible! Basically it is equal to the difference between the (relative) process deadline and the sum of durations of all nodes between this node and the last node (transitive successors). (iv) Finally, the latest allowed start (las) is lae minus duration. We assume that there is no delay between finishing a node and starting its successor, therefore the eps (lae) of a node will always be equal to the epe (las) of its predecessor. Furthermore, due to their 0-duration, the eps (lae) and epe (las) of control nodes will always be equal.

3.3 Calculating the Probabilistic Timed Graph

To determine eps and epe-histograms of every node v the forward calculation rules specified in table 1 have to be applied to each node in a topological order, where $v.pred$ denotes the set of predecessor of a node of v. During the calculation usually the duration-histograms are summed up to determine the according eps and epe-values, except for join-nodes: here multiple paths converge. Therefore the histograms must be *merged*. For AND-joins we use the max-conjunction as the longest path (or histogram-tuple) determines the resulting tuple. For XOR-splits we must weight the histograms of predecessors with the according branching probability before we merge them with the conjunction. Analogously, to determine las and lae-histograms, backward calculations have to be applied in a backward topological order. As we reversed the direction, now we apply the histogram subtraction, starting from the last node (whose lae has been initialized with the deadline), except for split-nodes. Here we apply the min-conjunction at AND-splits and weight&disjunction at XOR-joins. Applying these rules on the process sample in Fig. 1, we get the following results (equal histograms are caused by 0-durations and the fact that we assume no delay between end of one and start of the next activity or node):

$$
\begin{aligned}
&\underline{\textit{Forward Calculation : starting with time point } 0} \\
a.eps &= start.epe = start.eps = \{(1.0, 0)\} \\
b.eps &= c.eps = s.epe == a.epe = a.d + a.eps = \{(0.5, 1), (0.5, 5)\} \\
b.epe &= b.d + b.eps = \{(0.5, 5), (0.5, 9)\} \\
c.epe &= c.d + b.eps = \{(0.5, 9), (0.5, 13)\} \\
j.eps &= (b.epe * 0.25) \vee (c.epe * 0.75) = \{(0.125, 5), (0.5, 9), (0.375, 13)\} \\
d.eps &= j.epe = j.eps \\
end.epe &= end.eps = d.epe = d.d + d.eps = \{(0.125, 12), (0.5, 16), (0.375, 20)\} \\
&\underline{\textit{Backward Calculation : starting with deadline } \delta = 20} \\
d.lae &= end.las = end.lae = \{(1.0, \delta)\} \\
b.lae &= c.lae = j.las = j.lae = d.las = d.lae - d.d = \{(1.0, 13)\} \\
b.las &= b.lae - b.d = \{(1.0, 9)\} \\
c.las &= c.lae - c.d = \{(1.0, 5)\} \\
s.lae &= (b.las * 0.25) \vee (c.las * 0.75) = \{(0.25, 9), (0.75, 5)\} \\
a.lae &= s.las = s.lae \\
start.las &= start.lae = a.las = a.lae - a.d = \{(0.125, 8), (0.75, 4), (0.125, 0)\}
\end{aligned}
$$

4 Temporal Conformance

Conformance of web service compositions includes different aspects like structural, dataflow, messaging and temporal conformance. Temporal conformance ensures that the flow of information and tasks is done in a timely manner, considering the dependencies between activities, which may reside in different choreographies and orchestrations. In addition it must be checked that no explicitly assigned deadline is violated.

4.1 The Proposed Approach

Calculation of PTG of choreographies and orchestrations is based on iteratively delimiting the initial intervals of activities because of implicit and explicit constraints. In addition, orchestrations with a link may impose a restriction on the timed graph because of private (hidden) activities present in them. The imposed restriction further tightens the interval. Note that a link indicates a choreography is (partially) realized by this orchestration. A valid execution interval can be calculated by taking all affecting factors into account: implicit and explicit constraints and links with orchestrations. During calculation the conformance conditions must be always satisfied: *eps*-histogram of an activity must be less or equal to its *las*-histogram and $eps + d$ must be less or equal to *epe* (by means of histogram-comparison).

One important issue to be addressed is the case when one choreography has multiple realizing orchestrations as depicted in fig. 2. The numbers beside the arrows show their order of execution. eps or lae-histograms are propagated from *C* to *O1* (*1*), after recalculations at *O1*, they are propagated back to *C* (*2*). Note that propagation may change the the histograms of the target node. This cycle is

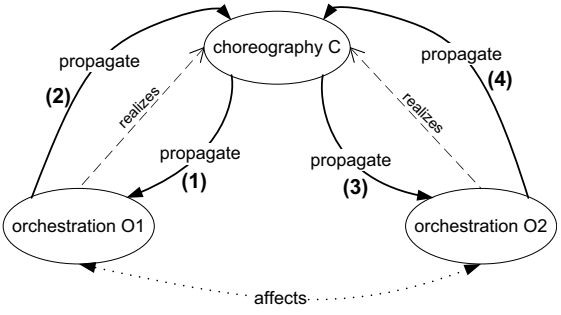

Fig. 2. Realized choreography with multiple incoming links

again repeated for *O2 (3,4)*. If *O2* again modifies the histograms of *C*, the most recently modified histogram may again impose a restriction on intervals of *O1*. In other words, two or more orchestrations with the same realized choreography may affect each other indirectly. This downward, upward cycle of propagation-recalculation must be iterated for all realizing orchestrations of a choreography as long as a "stable" state is reached i.e. no histogram is changed after propagation. If the histograms of a choreography or orchestration are changed, this change can be propagated in both directions i.e. to the choreography that it realizes and to the orchestrations by which it is realized. After such a change all incoming and outgoing links to other choreographies or orchestrations are marked and the recalculated histograms are propagated for all links in both directions.

4.2 Methods

Method *initialize.* This method initializes all e and l-histograms in a given graph to $\{(1.0, 0)\}$ and $\{(1.0, \infty)\}$ respectively. The properties $a.eps'$ and $a.lae'$, both initialized with $\emptyset$, are used for the propagation of interval restrictions.

Method *propagate.* This method propagates temporal values from one choreography or orchestration G to another one H, but only if G constrains the interval [eps,lae] of H. This means that propagation will only occur if eps will be increased or lae decreased. As we deal with histograms we need a parameter *certainty*. It defines the probability (degree) to be applied on histogram comparison operations (see also section 3). A 100%-certainty ensures that the compared histograms have no overlapping regions at all, but a very high certainty is more vulnerable to non-conformance conflicts than lower ones (see method *checkConformance* for further details). The propagated histograms will be stored in $x.eps'$ and $x.lae'$ of an activity x for further usage in subsequent calculations. The method uses event correspondence for assignment of *eps* and *lae*-histograms from a source to a target activity. The correspondence of the start events are used for assignment of *eps*-histograms and the correspondence of end events for *lae*-histograms.

propagate$(G, H,$ ***certainty)***
change := false
for all activities $\{x \in H \mid \exists a \in G : x_s \equiv a_s\}$ in a
topological order {
  if $x.eps <_{certainty} a.eps$
    $x.eps' := a.eps$
    change := true
  endif
  if $x.lae >_{certainty} a.lae$
    $x.lae' := a.lae$
    change := true
  endif
endfor }

Method *calculate*. Input-parameters are a choreography or orchestration. This method is used for pre-calculation of timed graphs as well as for repeated recalculations after interval-propagations. Therefore it is a slightly modified version of the PTG-calculation described in 3 with forward and backward-calculation of eps and lae-histograms. We had to consider that the execution interval of an activity a (eps and lae-histogram) may already be constricted due to a prior propagation from another orchestration/choreography (stored in $a.eps'$ and $a.lae'$). If this is the case we simply merge the calculated histogram with the propagated one. As this merge has the exactly same semantics as an ordinary AND-structure we can use the histogram conjunction operation (max for eps, min for lae), which ensures that the eps-histogram will increase and the lae-histogram will decrease, hence further restricting the valid interval.

calculate(G)
for all nodes $n \in N, G = (N, E)$ in forward topological order {
  calculate $n.eps$ according to table 1
  if $n.type = activity \wedge n.eps' \neq \emptyset$
    $n.eps := n.eps \wedge_{max} n.eps'$
    $n.eps' = \emptyset$
  endif
  calculate $n.epe$ according to table 1
endfor }
for all nodes $n \in N, G = (N, E)$ in backward topological order {
  calculate $n.lae$ according to table 1 (in reverse order)
  if $n.type = activity \wedge n.lae' \neq \emptyset$
    $n.lae := n.eps \wedge_{min} n.eps'$
    $n.eps' = \emptyset$
  endif
  calculate $n.las$ according to table 1 (in reverse order)
endfor }

Method *checkConformance*. This method checks if the basic conformance conditions is satisfied: (1) the earliest possible start time of an activity must always be less or equal than its latest allowed start time $eps \leq las$, and (2) the sum of earliest possible start time and durations must not exceed its earliest

possible end time $eps + d \leq epe$. Otherwise the boolean variable *conf* is set to false, which stops the algorithm. This condition must always be met for all activities of all choreographies and orchestrations. For the same reason as before, the variable *certainty* is used for histogram comparison operations (see also Chapter 3). A 100%-certainty ensures that the compared histograms will have no overlapping regions at all, but a very high certainty will be more vulnerable to non-conformance conflicts than lower ones. A relaxed certainty, will allow overlapping regions, which might prove useful as it deals with outliers. Furthermore it is possible to use the certainty as adjusting bolt, to select a strategy, from very conservative (strict) to risky (which allows more possible violations during run-time).

checkConformance(G,certainty)
for all activities $a \in G$ in a reverse topological order {
  if $a.eps >_{certainty} a.las$
    conf := false
  elsif $a.eps + a.d >_{certainty} a.epe$
    conf := false
  endif
endfor }
return conf

4.3 The Temporal Conformance Checking Algorithm

The algorithm consists of two parts: (1) the initialization and precalculation phase, and (2) the recalculation and conformance checking phase. Note that the algorithm needs a *certainty*-value as input-parameter (the higher this value, the stricter the conformance check).

In the first phase each choreography in C and each of its realizing orchestrations are initialized, followed by the calculation of their timed graphs (e and l-histograms). Note that the value max depicts the explicitly defined maximum duration for each graph (orchestration or choreography), which is needed to initialize the according deadline δ (necessary for the backward calculation of a timed graph). In this phase propagation will only occur between the choreography and its realizing orchestrations (and vice versa). The resulting e and l-histograms serve as initial values for further calculations.

Each calculation is followed by an initial basic conformance check. The value of the flag $conf$ signals if temporal conformance can be guaranteed (at least up to the given certainty-value). The only reason why the check may fail at this stage is a too tight deadline (caused by a too low maximum duration). A boolean variable *change* serves as an indicator if temporal values of a node are changed. If this variable becomes true all incoming and outgoing links of the corresponding choreography or orchestration are marked. Start and target node of each marked link are to be revisited and eventually recalculated in the next phase. Multiple marks on one edge have no additional effect.

temporalConformanceFederation(certainty)

```
- - (1) initialization and precalculation- -
for all c_i ∈ C {
   conf := true
   initialize(c_i)
   c_i.δ := c_i.max
   calculate(c_i)
   conf := checkConformance (c_i, certainty)
   for all realizing orchestrations o of c_i in a topological
   order {
         initialize(o)
         change := propagate(c_i,o,certainty)
         if change = true
            o.δ := o.first.eps + o.max
            calculate (o,certainty)
         endif
         change: = propagate(o, c_i,certainty)
         if change = true
            calculate(c_i,certainty)
            conf := checkConformance(c_i,certainty)
            mark all incoming and outgoing edges of c_i
         endif
   endfor }
endfor }
if conf = false return false
```

If basic temporal conformance can be guaranteed, the second phase of the procedure starts: recalculation and conformance checking. For all marked edges, the cycle of propagation, recalculation, and conformance-check is repeated until a stable state is reached or the conformance condition is violated. A stable state is reached if no edge has a mark on it.

```
- - (2) recalculation and conformance checking
repeat {
   select randomly a marked edge e such that c is the
   choreography and o the orchestration
      change: = propagate(c, o, certainty)
      if change = true
         calculate(o,certainty)
         conf := checkConformance (o, certainty)
         mark all incoming and outgoing edges ∈ o
      endif
      change: = propagate(o, c, certainty)
      if change = true
         calculate c
         conf := checkConformance (c, certainty)
         mark all incoming and outgoing edges ∈ c
      endif
      unmark e }
until (all edges are unmarked ⋁ conf = false)
return conf
```

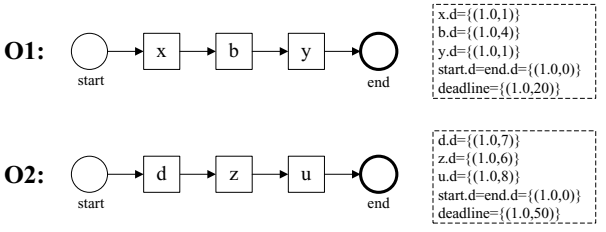

Fig. 3. Orchstrations $O1$ and $O2$ of figure 2

The following example shows numerically how the conformance checking algorithm works. Because of space limitations we chose a very simple scenario: it consists of one choreography, realized by two orchestrations (see also fig. 2). The graph of the choreography C is represented by the graph displayed in figure 1. The graphs of orchestrations $O1$ and $O2$ are illustrated in figure 3. The choreography is linked to the realizing orchestrations as follows:

- by the corresponding node b, which is one of the conditional nodes in C and the middle nodes in $O1$. Both nodes refer to the same activity, and therefore they have the same duration.
- by the corresponding node d, which is the last activity in C and the first activity in $O2$.
- activities a and c in C correspond to further orchestrations which we, for reasons of simplicity, do not consider here.

For the algorithm we define a very strict $certainty = 99\%$. The first phase starts with the initialization and calculation of C. The resulting timed graph has already been calculated in subsection 3.3. Now we initialize the eps and lae-histograms in $O1$ (with 0 and ∞ values), followed by the propagation between corresponding nodes of C and $O1$ (activity b). Propagation, will only occur, if it further constrains (narrows) the interval [eps,lae] of $O1.b$. The method *propagate* first checks if $O1.b.eps <_{0.99} C.b.eps$; this is the case, therefore we set $O1.b.eps' = \{(0.5, 1), (0.5, 5)\}$. Analogously we check the lae-histograms: as $O1.b.lae >_{0.99} C.b.lae$ the method *propagate* sets the intermediate $O1.b.lae' = \{(1.0, 13)\}$. Now the calculation of O1 starts as described in method $calculate(G)$ (calculation-details only for max-conjunction at b.eps and min-conjunction at b.lae):

$$
\begin{aligned}
x.eps &= start.epe = start.eps = \{(1.0, 0)\} \\
b.eps &= x.epe = \{(1.0, 1)\} \\
b.eps &= b.eps \wedge_{max} b.eps' = \{(0.5, 1), (0.5, 5)\} \\
y.eps &= b.epe = \{(0.5, 1), (0.5, 5)\} \\
end.epe &= end.eps = y.epe = \{(0.5, 2), (0.5, 6)\} \\
y.lae &= end.las = end.lae = \{1.0, 20)\} \\
b.lae &= y.las = \{(1.0, 19)\} \\
b.lae &= b.lae \wedge_{min} b.lae' = \{(1.0, 13)\} \\
x.lae &= b.las = \{(1.0, 9)\} \\
start.las &= start.lae = x.las = \{(0.5, 8), (0.5, 4)\}
\end{aligned}
$$

In the next step the calculated values must be propagated from $O1$ to C, if the according propagation-conditions apply for $b.eps$ and $b.lae$. None apply, therefore the timed graph of C does not change! Now the initialization and precalculation starts for $O2$. After the initialization, the method *propagate* sets $O2.d.eps = \{(0.125, 5), (0.5, 9), (0.375, 13)\}$ and $O2.d.lae = \{(1.0, 20)\}$, and the subsequent calculation of the probabilistic timed graph yields:

$$\begin{aligned}
d.eps &= start.epe = start.eps = \{(1.0, 0)\} \\
d.eps &= d.eps \wedge_{max} d.eps' = \{(0.125, 5), (0.5, 9), (0.375, 13)\} \\
z.eps &= d.epe = \{(0.125, 12), (0.5, 16), (0.375, 20)\} \\
u.eps &= z.epe = \{(0.125, 18), (0.5, 22), (0.375, 26)\} \\
end.epe &= end.eps = u.epe = \{(0.125, 26), (0.5, 30), (0.375, 34)\}
\end{aligned}$$

$$\begin{aligned}
u.lae &= end.las = end.lae = \{(1.0, 50)\} \\
z.lae &= u.las = \{(1.0, 42)\} \\
d.lae &= z.las = \{(1.0, 36)\} \\
d.lae &= d.lae \wedge_{min} d.lae' = \{(1.0, 20)\} \\
start.las &= start.lae = d.las = d.las = \{(1.0, 13)\}
\end{aligned}$$

The reverse propagation – of d from $O1$ to C – does not change any value in the timed graph of C. As no further choreographies exist, and no marked edges are left, the algorithm finishes and returns true. This specific composition temporally conforms, and no deadline will be violated if the real durations of activities adhere to the estimated/mined durations. The build-time calculation is now finished.

4.4 Proof of Termination

We informally prove that the proposed algorithm terminates. The algorithm terminates in two cases: (1) as the number of edges is finite, a stable state will be reached in a finite number of steps. Or (2), if such a stable state does not exist, after a finite number of steps the conformance condition will be violated, because with each iteration the lae becomes smaller and the eps value greater until $eps >_{certainty} lae$, since there is only a finite number of chronons between time points.

5 Conclusions

Temporal quality criteria play an important role in service compositions and distributed business processes. In inter-organizational environments the degree of influence on how and when partners execute their activities is rather reduced. Therefore it is important to verify temporal conformance at build-time, process instantiation and during run-time, in order to increase the overall quality of the whole process. We proposed a stochastic technique to model nested choreographies and orchestrations, temporal information and temporal constraints. The

probabilistic time plan delivers valid execution intervals for all activities which can then be used to verify their temporal conformance. Furthermore, these execution plans can additionally be monitored at run-time, which allows for predictive and pro-active time management [5], i.e. to diagnose potential violations of temporal constraints early enough such that counter-measures can still be taken to guarantee correct executions of the flow.

Acknowledgments. This work is partly supported by the Commission of the European Union within the project WS-Diamond in FP6. STREP.

References

1. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services: Concepts, Architectures and Applications. Springer, Heidelberg (2004)
2. Benatallah, B., Casati, F., Ponge, J., Toumani, F.: On temporal abstractions of web service protocols. In: Proc. of CAiSE Forum (2005)
3. Kazhamiakin, R., Pandya, P., Pistore, M.: Timed modelling and analysis in web service compositions. In: Proc. of ARES 2006 (2006)
4. Kazhamiakin, R., Pandya, P., Pistore, M.: Representation, verification, and computation of timed properties in web service compositions. In: Proc. of ICWS 2006 (2006)
5. Eder, J., Panagos, E., Rabinovich, M.I.: Time Constraints in Workflow Systems. In: Jarke, M., Oberweis, A. (eds.) CAiSE 1999. LNCS, vol. 1626, p. 286. Springer, Heidelberg (1999)
6. Eder, J., Pichler, H.: Duration Histograms for Workflow Systems. In: Pro. of IFIP TC8/WG8.1 Working Conference on Engineering Information Systems in the Internet Context. Kluwer Publishers, Dordrecht (2002)
7. Barros, A., Dumas, M., Oaks, P.: Standards for web service choreography and orchestration: Status and perspectives. In: Bussler, C.J., Haller, A. (eds.) BPM 2005. LNCS, vol. 3812, pp. 61–74. Springer, Heidelberg (2006)
8. Peltz, C.: Web services orchestration and choreography. IEEE Computer 36(10), 46–52 (2003)
9. Eder, J., Tahamtan, A.: Temporal Consistency of View Based Interorganizational Workflows. In: Kaschek, R., Kop, C., Steinberger, C., Fliedl, G. (eds.) UNISCON 2008. LNBIP, vol. 5, pp. 96–107. Springer, Heidelberg (2008)
10. Eder, J., Pichler, H.: Probabilistic Workflow Management. Technical report, Universitä Klagenfurt, Institut für Informatik Systeme (2005)