Temporal Conformance of Federated Choreographies

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

[1] Alpen-Adria University of Klagenfurt, Dept. of Information Systems, A-9020 Klagenfurt, Austria
eder@isys.uni-klu.ac.at
[2] University of Vienna, Dept. of Knowledge and Business Engineering, Rathausstrasse 19/9, A-1010 Vienna, Austria
amirreza.tahamtan@univie.ac.at

Abstract. Web service composition is a new way for implementing business processes. In particular, a choreography supports modeling and enactment of interorganizational business processes consisting of autonomous organizations. Temporal constraints are important quality criteria. We propose a technique for modeling temporal constraints in choreographies and orchestrations, checking whether the orchestrations satisfy the temporal constraints of a choreography and compute internal deadlines for the activities in an interorganizational workflow.

Keywords: Web Services, Composition, Choreographies, Orchestration, Temporal Conformance.

1 Introduction

A major step toward integration of web services into organizations is their composition, enabling single web services be composed to an orchestration and choreographies describing the collaboration of orchestrations. Temporal aspects are important quality criteria in business processes. Temporal constraints are envisioned as part of the negotiations for setting up choreographies. It must be ensured that activities are performed in a timely manner and right information is delivered to right activities at the right time such that overall temporal restrictions are satisfied. Choreographies and orchestrations may have deadlines which specify the latest time point in which they must finish execution. Temporal conformance checking assists organizations to provide services with a higher QoS and reduces the process costs as violation of temporal constraints leads to costly exception handling mechanisms [1].

Federated Choreographies have been introduced in [2] as a hierarchical architecture for a consistent modeling of choreographies and orchestrations. Central to the model is the notion of conformance. Structural [3], temporal, data flow, and messaging conformance are different aspects.

Here we propose an algorithm for checking of temporal constraints of federated choreographies and generate a temporal execution plan. Based on this, one can

S.S. Bhowmick, J. Küng, and R. Wagner (Eds.): DEXA 2008, LNCS 5181, pp. 668–675, 2008.

decide wether execution of the model leads to temporal exceptions and necessary modifications can be done. The algorithm works in a distributed manner and there is no need for a central role. A choreography defines the interaction among partners, accessible activities and which activities this partner has to execute. Because of the distributed functionality of the algorithm, one partner may need data from another partner to process locally. A partner can request for data which is only associated to its accessible activities. Such activities are defined in the choreography and are public to all partners, i.e. data provider has not to reveal its private data but only required data for interaction. This enables partners to hide their internal process logic whilst allow for interaction. Temporal conformance checking has a build-time and a run-time aspect. At build time we check whether all orchestrations meet the temporal restrictions given by the choreographies. At run-time, execution is monitored to allow for predictive and pro-active time management, i.e. to diagnose potential violations of temporal constraints early enough such that counter-measures still can be taken.

Temporal aspects of web services have been studied in [4,5,6]. [4] uses temporal abstractions of protocols for compatibility and replaceability analysis based on a finite state machine formalism. [5,6] exploit an extension of timed automata for modeling time properties of web services. Our approach can cover cases modeled in these works and additionally allows for explicit deadlines. This work extends previous works by addressing the conformance and verification problem and provides an a priori execution plan at build-time (both best and worst cases) consisting of valid execution intervals for all activities with consideration of the overall structure and temporal restrictions. The calculated execution plans can be monitored at run-time.

2 Federated Choreographies

There are mainly two ways for modeling choreographies: by protocols between orchestrations and by abstract processes [7]. Protocols have the advantage of flexibility and that only a minimum of information about processes is disclosed. They have the disadvantage that the overall process is never explicitly modeled, a drawback for process awareness. Abstract processes model the choreography as a (global) process which integrates the disclosed views of the participating processes [8]. They have the disadvantage that all partners share the global process, even, if subsets of partners have closer relationships [9,10]. *Federated choreographies* [2] overcome this disadvantage by allowing abstract processes with different level of abstraction, organized in a hierarchy. The abstract processes describe a business process in various level of detail. This approach is more flexible than typical compositional approaches and provides advantages like a better business secrecy, extendability and uniform modeling. The main idea of this approach is presented in fig. 1. The upper layer is composed of shared choreographies. A choreography may support another one, i.e. the former is a partial extension of the latter. The supporting choreography is an extended subset (of activities) of the supported choreography. The support relationship

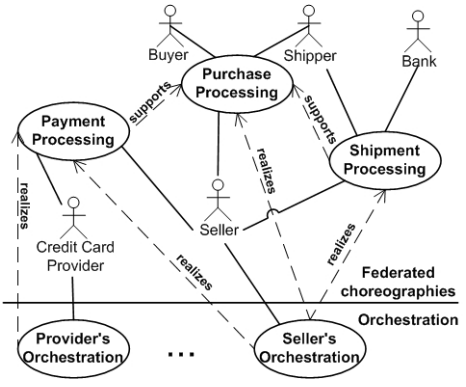

Fig. 1. Federated choreographies

is acyclic. A choreography which is only supported and realized by other nodes and in turn supports no choreography is called *the global choreography*. A node refers to a choreography and/or orchestration. Informally, the global choreography captures the core of the business process and its supporting choreographies reflect how its parts are carried out in reality. The bottom layer consists of realizing orchestrations. Each partner provides its own internal realization of relevant parts of the choreographies. The presented approach is fully distributed. Each partner has local models of all choreographies in which it participates. Additionally, each partner holds and runs its own model of the orchestration. There is no need for a centralized coordination. For a detailed discussion refer to [2,3].

Both nodes are modeled as workflows. To avoid the complex metamodel of [2], this paper is based on a simplified process model. A generic workflow model is used in this paper as a structure for representing temporal information. A workflow is a collection of activities and the dependencies between them. Activities correspond to individual steps in a process. Dependencies determine the execution sequence of activities and the data flow. Activities can be executed sequentially, in parallel and conditionally. Consequently, a workflow can be represented by a directed acyclic graph, where nodes correspond to activities and edges to dependencies. All activities have a unique name and two corresponding events. An event is either start of an activity (denoted a_s for an activity a) or its end (denoted a_e for an activity a). In this paper we model the relationship between a supporting and a supported choreography simply by event correspondence. $e_1 \equiv e_2$ denotes that event e_1 corresponds to event e_2. Note that e_1 and e_2 may belong to different nodes.

To represent time information, the workflow model is augmented with the following temporal types: time points, durations and deadlines. We refer to such a graph as timed graph (TG) [1] (See Fig. 3 and 4)

3 Temporal Conformance

Amongst other conformance issues temporal conformance is a key requirement of the federated choreographies. It should be ensured that the flow of information and tasks is done in a timely manner with consideration of the structural dependencies. In addition it must be checked that no deadline is violated.

3.1 Prerequisites

The concepts used for calculation of temporal plans come from the field of operations research. Two kinds of constraints are used:

- **Implicit constraints** are derived implicitly from the structure of the process.
- **Explicit constraints**, e.g. assigned deadlines, can be set or enforced explicitly.

All activities of a node have durations. In this work deterministic values for durations are used. We are aware that activity durations may vary. However, we use fixed values for clarification of the concepts. We calculate an interval in which an activity can execute as described in [11]. This interval is delimited by *earliest possible start*(eps) and *latest allowed end*(lae). *eps* is the earliest point in time in which an activity can start execution and *lae* the latest point in time in which an activity can finish execution without violating the deadline. Both *eps* and *lae* are calculated for *best case* and *worst case*. The best case is given, if the shortest path is executed and the worst case when the longest path is executed.

3.2 The Proposed Approach

Fig. 1 illustrates the starting point of the algorithm. To make the presentation less complex we assume that only one global choreography exists.

The calculation of a node's TG is based on iteratively delimiting the initial intervals of activities because of implicit and explicit constraints. In addition, other nodes with a link may also impose a restriction on the TG because of additional activities present in them which may further tighten the interval. A link identifies a dependency between two nodes and is either a support or realize relationship. A valid execution interval is calculated when all constraints are considered: implicit, explicit constraints and links. The conformance condition must always be met i.e. eps+duration of an activity $eps + d$ must be less or equal to its *lae* in both best and worst cases.

One important issue to consider is when one node has multiple incoming links as depicted in fig. 2. The numbers beside the arrows show the order of execution. The method *assignValuesTo* is described in subsection 3.3. Temporal values are assigned from *G* to *S1* (*1*), after recalculations at *S1*, they are assigned to *G* (*2*). An assignment may change the values of the target node. This cycle is repeated for *S2* (*3,4*). If *S2* again modifies *G*, the most recently modified values may again

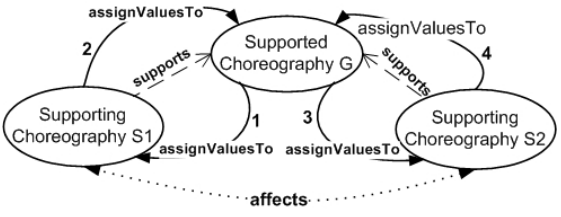

Fig. 2. Supported choreography with multiple incoming links

impose a restriction on *S1*. In other words, two or more nodes with the same supported/realized node may affect each other indirectly even with no direct link. This cycle of assignment-recalculation must be iterated for all supporting and realizing nodes of a choreography as long as a stable state is reached i.e. no values are changed after an assignment. Change of a TG can be propagated in both directions, i.e. along incoming and outgoing links. After change of the values of a node G, all of its incoming and outgoing links are marked and the recalculated values are propagated for all links with a source or target node G.

3.3 Methods

Following notations are used in the methods: *a.bc.eps* and *a.bc.lae* denote the best case *eps* and *lae*. *a.wc.eps* and *a.wc.lae* represent these values for the worst case. *a.d* duration of an activity *a*. *a.pred* and *a.succ* set of predecessors and successors of an activity *a* respectively. *a.pos* position of an activity *a* in the TG. a_s denotes the start-event of an activity *a* and a_e its end-event. *G.deadline* deadline of a TG *G*. *d.max* maximum allowed duration of a node. Upper case letters represent the nodes and lower case letters the activities.

initialize(G)
This method initializes the eps and lae-values of a node to 0 and ∞ respectively. The reason is that eps can only become greater and lae smaller.

calculate$(G, G.deadline)$
This method takes as input a node and the output is the calculated TG for best and worst cases.

When recalculating a node's TG, existing eps(lae) is replaced by the recently calculated eps(lae), if the calculated values are greater(smaller) than existing values. This method is used for pre-calculation of TGs as well as for recalculation of a TG after assignment from another node.

assignValuesTo(G, H)
This method assigns values from one node to another. It uses event correspondence for assignment. Correspondence of start events is used for assignment of eps and correspondence of end events for lae.

checkConformance(G)
The above method checks if the conformance condition is fulfilled (See section 3.2)

Calculation of Timed Graphs and Temporal Conformance Checking

```
temporalConformanceFederation()
  -initialization and precalculation-
  conf := true
  initialize(C_g)
  calculate(C_g)
  conf := checkConformance (C_g)
    for all directly and indirectly supporting choreographies
    and realizing orchestrations G of C_g in a topological
    order {
          initialize(G)
          change := assignValuesTo(C_g, G)
          if change = true
             G.deadline := G.first.eps + G.d.max
             calculate (G)
          endif
          change: = AssignValuesTo(G, C_g)
          if change = true
             calculate(C_g)
             conf := checkConformance(C_g)
             mark all incoming and outgoing edges of C_g
          endif
    endfor }
  -recalculation and conformance checking-
  Repeat {
    select randomly a marked edge e such that G is the
    supported choreography and H the supporting
    choreography or realizing orchestration
       change: = AssignValuesTo(G, H)
       if change = true
          calculate H
          conf := checkConformance (H)
          mark all incoming and outgoing edges ∈ H
       endif
       change: = AssignValuesTo(H, G)
       if change = true
          calculate G
          conf := checkConformance (G)
          mark all incoming and outgoing edges ∈ G
       endif
       unmark e }
  Until (all edges are unmarked ⋁ conf = false)
```

In the initialization and precalculation phase after initialization of the global choreography, its TG is calculated without considering other nodes. That means only implicit and explicit constraints are considered. Maximum duration $d.max$ is considered for calculating the deadline of other nodes than the global choreography which is the maximum duration during which a workflow can execute whereas a deadline denotes a point in time. Like deadlines, $d.max$ is given a

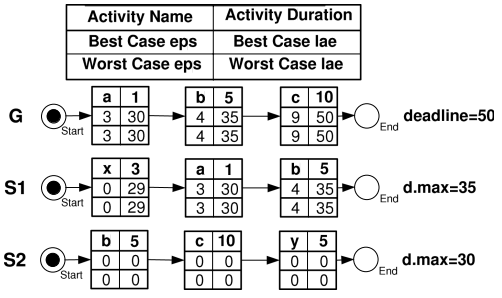

Fig. 3. After steps 1 and 2

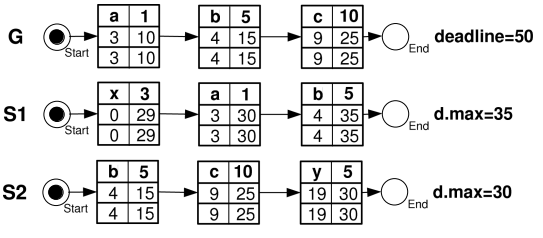

Fig. 4. After steps 3 and 4

priori. It suffices in this phase to assign the values to each node only once. These values just serve as initial values for further calculations. A variable *change* indicates the change of a TG. If *change* becomes true all incoming and outgoing links of the corresponding node are marked. Start and target node of each marked link must be revisited and recalculated if any value is changed. Multiple marks on an edge has no additional effect.

The recalculation and conformance checking phase consists of recalculation of the precalculated TGs in the previous phase. For all marked edges, the cycle of assignment-recalculation is repeated until a stable state is reached or the conformance condition is violated. A stable state is reached if all marked edges are unmarked. Figures 3 and 4 show by a numeric example how TGs are calculated. Because of space limitations a simple example is chosen. The dependency between nodes and the steps of this example are described in fig. 2. Note that fig. 3 and fig. 4 demonstrate only one cycle. Note that this cycle must be iterated as long as a stable state is reached.

We have implemented a prototype as proof of concept. The workflow specifications of nodes together with deadlines, *d.max* and links are read as input. The tool calculates the execution plan for all nodes and checks if the conformance condition is met.

3.4 Proof of Termination

We informally prove that the algorithm terminates. Algorithm terminates in two cases: if there are no marked edges or the conformance condition is violated. Because the number of edges is finite, in a finite number of steps the stable state is reached. If such a stable state does not exist, after a finite number of steps the conformance condition is violated. The reason is with each iteration, if changes are made, the *lae* becomes smaller and the *eps* greater until $eps + a.d > lae$.

4 Conclusions

Temporal quality criteria play an important role in business processes. We proposed a technique for modeling and checking of temporal constraints of choreographies and orchestrations. A time plan is generated for each choreography and orchestration representing valid execution intervals for their activities. This time plan is used at run-time for monitoring purposes and allows for pro-active time management to avoid temporal failures. The algorithm is fully distributed such that there is no need to compromise the autonomy of partners in interorganizational workflows.

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

References

1. Panagos, E., Rabinovich, M.: Predictive workflow management. In: Proc. of the 3rd Int. Workshop on Next Generation Information Technologies and Systems (1997)
2. Eder, J., Lehmann, M., Tahamtan, A.: Choreographies as federations of choreographies and orchestrations. In: Proc. of CoSS 2006 (2006)
3. Eder, J., Lehmann, M., Tahamtan, A.: Conformance test of federated choreographies. In: Proc. of IESA 2007 (2007)
4. Benatallah, B., Casati, F., Ponge, J., Toumani, F.: On temporal abstractions of web service protocols. In: Proc. of CAiSE Forum (2005)
5. Kazhamiakin, R., Pandya, P., Pistore, M.: Timed modelling and analysis in web service compositions. In: Proc. of ARES 2006 (2006)
6. Kazhamiakin, R., Pandya, P., Pistore, M.: Representation, verification, and computation of timed properties in web service compositions. In: Proc. of ICWS 2006 (2006)
7. Andrews, T., et al.: Business process execution language for web services (bpel4ws). ver. 1.1. BEA, IBM, Microsoft, SAP, Siebel Systems (2003)
8. Banerji, A., et al.: Web services conversation language (wscl) 1.0. Technical report, W3C (2002)
9. Peltz, C.: Web services orchestration and choreography. IEEE Computer 36(10), 46–52 (2003)
10. Dijkman, R., Dumas, M.: Service-oriented design: A multi-viewpoint approach. Int. J. Cooperative Inf. Syst. 13(4), 337–368 (2004)
11. Eder, J., Panagos, E., Rabinovich, M.: Time Constraints in Workow Systems. In: Jarke, M., Oberweis, A. (eds.) CAiSE 1999. LNCS, vol. 1626, Springer, Heidelberg (1999)