Temporal Management of WS-BPEL Processes

Amirreza Tahamtan[1], Christian Österle[1] A Min Tjoa[1], and Abdelkader Hameurlain[2]

[1] Institute of Software Technology & Interactive Systems, Information & Software Engineering Group, Vienna University of Technology, Favoritenstrasse 9-11/188, A-1040 Vienna, Austria
tahamtan@ifs.tuwien.ac.at,
WWW: http://www.ifs.tuwien.ac.at/~tahamtan
[2] Université Paul Sabatier, 118, route de Narbonne,
31062 Toulouse cedex, France

Abstract. WS-BPEL is de-facto industry standard for business processes. One of its major shortcomings is lack of temporal management capabilities. WS-BPEL offers no possibility for definition, calculation and monitoring of temporal values such as activity duration and deadlines as well as checking the temporal conformance of processes. This paper tackles temporal management of WS-BPEL based on two different techniques: interval-based and probabilistic. This paper describes temporal management of cooperating WS-BPEL abstract and executable processes. We have implemented a temporal management prototype as a WS-BPEL extension.

Key words: WS-BPEL, Time, Temporal Management, Extension, Conformance

1 Introduction

Temporal conformance and compliance are important quality criteria for business processes and interorganizational workflows. Processes may have deadlines. The assigned deadlines may be part of the service level agreement between partners or enforced by law or organizational policies. It must be ensured that the right information is delivered to the right activity at the right time and the process executes in a timely manner in order to be able to hold the deadlines. Temporal conformance on the one hand increases the QoS and on the other hand reduces the cost of process execution as costly exception handling mechanisms can be avoided. Temporal management can be used for three different purposes [14]:

- Predictive time management: to predict the possible temporal behavior of the system and pre-calculate future possible violations of temporal constraints.
- Pro-active time management: to detect potential future violations and raise alarm in these cases such that counter-measure mechanisms can be triggered early enough.

– Reactive time management: to react and trigger exception handling mechanisms if a temporal failure has already occurred.

Web Services and SOA offer several advantages for implementation of business processes such as interoperability, loosely coupling and composition. WS-BPEL has become the accepted standard for description end execution of business processes based on Web Services. In the realm of web services we mainly talk about two concepts: choreographies and orchestrations or in the WS-BPEL notation, abstract and executable processes.

A WS-BPEL executable process or orchestration is controlled and run by one partner. A partner's internal logic and business know-how are contained in his executable process. Other tasks such as data transformations, data handling, arithmetic operations and the actual performed work are as well contained in this process. An executable process is solely visible to its owner and other external partners have no view on and knowledge about it. An executable process is a process viewed only from the perspective of its owner.

On the other hand, a choreography, which is called abstract process in WS-BPEL, describes business protocols. An abstract process may be used to describe observable message exchange behavior of each of the parties involved, without revealing their internal implementation [12]. An abstract process can use all the construct of an executable process and have the same expressive power but it is not intended to be executed. It has merely a descriptive role. An abstract process defines a collaboration among involved partners to reach an overall business goal. It contains only visible exchanged messages between partners in course of a business process.

In order to ensure that cooperating business processes are temporally compliant, it must be guaranteed that both the tasks performed in executable processes and the protocols described in abstract processes have a compliant temporal behavior. In our previous work [15] we have implemented a time management extension for WS-BPEL, called BPEL-TIME. In this work we describe how BPEL-TIME handles temporal management of a set of cooperating WS-BPEL abstract and executable processes. BPEL-TIME consists of two components, a design time component and a run time component. The design time component allows the definition of temporal constraints. At design time it is checked if the model is temporally feasible, i.e. if there is a solution that satisfies all the temporal constraints. If the system is temporally not feasible, it can be detected at design time and necessary modifications performed. We calculate a valid temporal window for each activity in this phase. If an activity executes within its valid temporal window it is guaranteed that the whole process terminates successfully. The run time component monitors the execution of the process and informs the process manager if any deviation from valid temporal windows is detected. Based on the calculations at design time, the run time component predicts the future temporal behavior of the flow and informs the process manager about its status. Our approach is based on prevention of errors rather than repairing them after their occurrence. By predicting the behavior of a flow appropriate measures can be triggered in order to guarantee its successful execution. BPEL-TIME offers two different possibilities: an interval-based and a probabilistic approach, The interval-based approach allows the definition of fixed and/or variable temporal constraints such as deadlines and durations. The probabilistic approach enables a probabilistic representation of temporal constraints and takes also branching probabilities into account.

2 Model Description

For modeling and calculation of temporal plans of WS-BPEL executable and abstract processes three types of constraints have to be considered:

- **Implicit constraints** are derived implicitly from the structure of a process, e.g. an activity can start execution if and only if all of its predecessors have finished execution.
- **Explicit constraints**, e.g. assigned deadlines, can be set explicitly by the process designer or enforced by law, regulations or business rules.
- **Dependencies with other processes** may impose a temporal restriction on a process. It is not enough to perform a temporal analysis in isolation.

The first two constraints are needed to calculate the temporal plan of one single process. Note that abstract and executable processes are both handled uniformly. From the point of view of temporal management it makes no difference if the WS-BPEL process is abstract or executable. The third constraint must be considered in order to check the temporal conformance and calculate the temporal plans of a set cooperating processes. We model the first two constraints using two different modeling approaches: the interval-based and the probabilistic approach.

2.1 Interval-Based Approach

As the basic modeling language, we adapt the model presented in [14]. It is a timed graph augmented with start and end events for activities. Timed graphs are familiar workflow graphs where nodes correspond to activities and edges to the dependencies between activities, enriched with temporal information. Fig. 1 shows an example of the model. All activities have a unique name, duration and two corresponding events: start and end event. $a.d$ denotes the duration of an activity a and a_s and a_e its start event and end event respectively. Because BPEL-processes are full-blocked [2], in this work we restrict the graphs to full-blocked ones, i.e. each split node has a counterpart join node and vice versa.

The interval-based approach allows modeling of fixed and variable durations of activities. The duration of an activity can take any value within an interval bounded by minimum and maximum durations, e.g. $a.d = [a.d_{min}, a.d_{max}]$, where $a.d$ refers to duration of an activity a. We use upper-bound and lower-bound constraints [7] to model this interval. Let a be the source event and b the destination event.

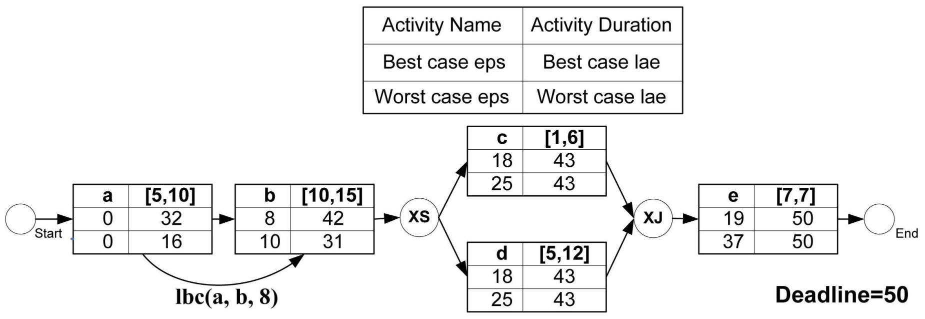


Fig. 1. An example of a timed graph with deadline= 50

Lower-bound constraint identifies the minimum temporal distance between two events. $lbc(a, b, \delta)$ denotes that between a and b at least δ time points must pass.

Upper-bound constraint identifies the maximum temporal distance between two events. $ubc(a, b, \delta)$ denotes that between a and b at most δ time points can pass.

$a.d_{min}$ and $a.d_{max}$ can be modeled by defining the minimum and maximum allowed time points between start event and end event of an activity respectively. This scenario as depicted in fig. 2, where $a.d = [2, 7]$. Obviously an activity can also be assigned a fixed value, if $a.d_{min} = a.d_{max}$. Note that in the rest of this paper, for the sake of brevity, we do not illustrate *lbc* and *ubc* as well as start and end events in the graphs, unless it contributes to a better understanding. *lbc* and *ubc* are not only used for modeling d_{min} and d_{max} of activities. They can also be used for modeling temporal constraints between different activities. For example to model requirements such as approval or rejection of an application may take at most one week after its receipt and sending a notification to the applicant takes at least three days.

For calculation of temporal values, we extend the algorithms developed in our previous works [11, 10]. An interval in which an activity may execute is calculated. This interval is delimited by *earliest possible start* (eps-value) and *latest allowed end* (lae-value). *a.eps* denotes the *eps*-value of an activity a and is the earliest point in time in which the activity a can start execution. *eps*-values reflect the implicit constraints of a flow. *a.lae* represents the latest point in time in which an activity a can finish execution in order to hold the assigned deadline. *lae*-values reflect the explicit constraints of a flow. Both *eps* and *lae* values are calculated for *best case* and *worst case.* Best case and worst case identifies the execution of the shortest and longest path of a flow respectively. *a.bc.eps* refers to best case *eps* and *a.wc.eps* refers to worst case *eps* of an activity a. The same applies to *lae*-values. Temporal values of a simple graph are depicted in fig. 1. The algorithms and techniques for calculation of the interval-based temporal values are described in our previous works [14, 11, 15] and omitted here.

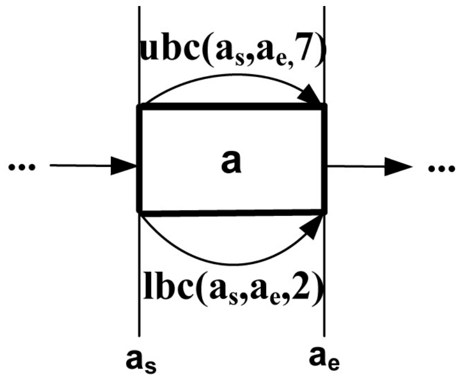


Fig. 2. Modeling d_{min} and d_{max} by lbc and ubc

Given known activity durations, in addition we can calculate *earliest possible end* (epe-values) and *latest allowed start* (las-values) for an activity, using the following formulas: $a.epe = a.eps + a.d$ and $a.las = a.lae - a.d$. We refer to eps-values and epe-values as e-values and to lae-values and las-values as l-values. In this approach we handle loops as complex activities. For calculation of the temporal plan at design time we consider only one iteration. Because the actual iterations of a loop is not known at design time, its execution is monitored at run time and process manager receives notifications about the temporal status of the process.

2.2 Probabilistic Approach

In some use cases one may need to consider branching probabilities. The probabilistic approach described in this subsection caters for probabilistic representation of temporal constraints.

In order to express variable duration of activities, the notion of time histograms [13, 9] is used. A duration histogram is a data structure for representation of the (probabilistic and variable) duration of basic activities, complex activities, subworkflows and workflow itself. A duration histogram is a tuple (p, d), where p is a probability and d a duration. For example the probabilistic duration of an activity can be represented as $\{(0.1, 10), (0.25, 12), (0.32, 15), (0.33, 20)\}$. This duration can be interpreted as follows: the duration of this activity is with the probability 10%, 10 time points, with the probability 25%, 12 time points, with the probability 32%, 15 time points and with the probability 33%, 20 time points. If a duration histogram contains any tuples whose time values are the same, these tuples must be merged by adding the probabilities of tuples with the same duration. A workflow graph augmented with probabilistic temporal information for activities and nodes is referred to as probabilistic timed graph (PTG). Fig. 3 illustrates an example of such a probabilistic timed graph. The duration of activities are given in the table above the graph.

The deadline of the workflow is also given in form of a (probability, duration) tuple. Analogous to duration histograms (d-histograms), [13] defines e-histograms for presentation of e-values and l-histograms for presentation of l-values. Note that for probabilistic calculations we do not consider best and worst case or *lbc* and *ubc*. The calculation of the probabilistic temporal values has been described in [15] and hence omitted here.

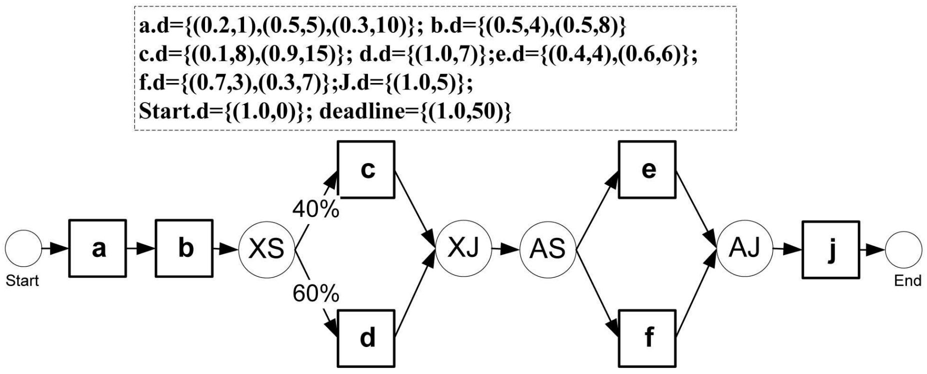

a.d={(0.2,1),(0.5,5),(0.3,10)}; b.d={(0.5,4),(0.5,8)}
c.d={(0.1,8),(0.9,15)}; d.d={(1.0,7)};e.d={(0.4,4),(0.6,6)};
f.d={(0.7,3),(0.3,7)};J.d={(1.0,5)};
Start.d={(1.0,0)}; deadline={(1.0,50)}
Start
a
b
XS
40%
60%
c
d
XJ
AS
e
f
AJ
j
End

Fig. 3. A sample probabilistic timed graph (PTG)

3 Temporal Management of WS-BPEL Executable and Abstract processes

The interval-based and the probabilistic approach can be used to calculate the temporal plan of one single process in isolation. However, in order to calculate the temporal plans of a set of cooperating processes and check their temporal conformance, it is necessary to consider the dependencies between them. Processes may be linked in several ways. As a generic architecture we use the model described in [5, 17, 16]. This architecture (see fig. 4) is generic enough for modeling different relationships between executable and abstract processes. It covers scenarios as e.g. described in [3, 4], one shared abstract process among a set of executable processes. This scenario is depicted in fig. 5. The generic architecture in fig. 4 covers also other scenarios as depicted in fig. 6

The architecture in fig. 4 is a two layered model. The first layer consists of abstract processes. An abstract process may support another abstract process. This means the former contributes to the latter and partially elaborates it. The first abstract process may describe the business process in details needed for a specific group of partners and the second abstract process describes the same process in more details for another group of partners involved in the process. In this way different groups of partners can have different views on the same process. The second layer consists of executable processes that realize the abstract

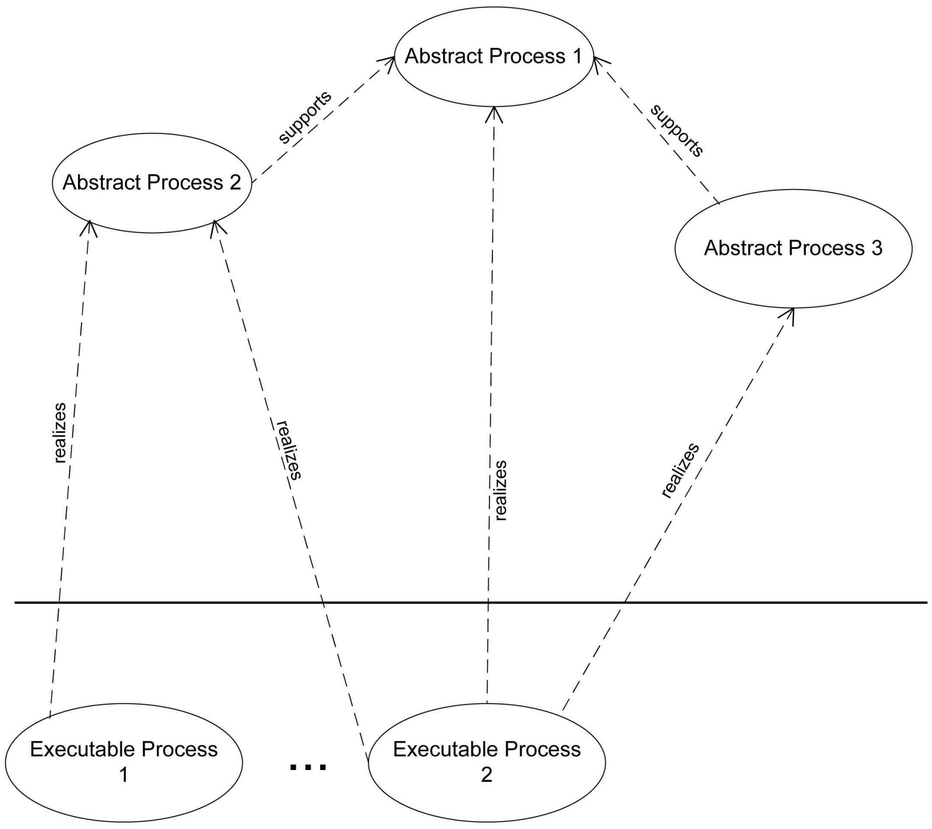


Fig. 4. A Generic architecture for abstract and executable processes

processes. The presented approach is fully distributed and there is no need for central coordination.

The most important issue to consider regarding the dependency between two nodes (abstract and/or executable processes) is the greatest common divisor (GCD) of their activities. GCD identifies the set of common activities in two nodes. A dependency between two nodes implies that GCD of activities of two nodes is not empty, i.e. these two nodes have at least one activity in common. Dependencies between activities are depicted using links between them as in fig. 4 and fig. 6. Fig. 8 depicts the abstract process C and the executable process *O1* in fig. 5. The GCD of two graphs include the activities *Receive request* and *Reply request*. For the sake of brevity the the executable process O2 is omitted. Note that even if $O1$ and $O2$ has no direct link with each other, through their common activities with C they may affect each other indirectly. This is due to the fact that $O1$ may affect C and change its temporal value and this change may in turn affect $O2$ or vice versa.

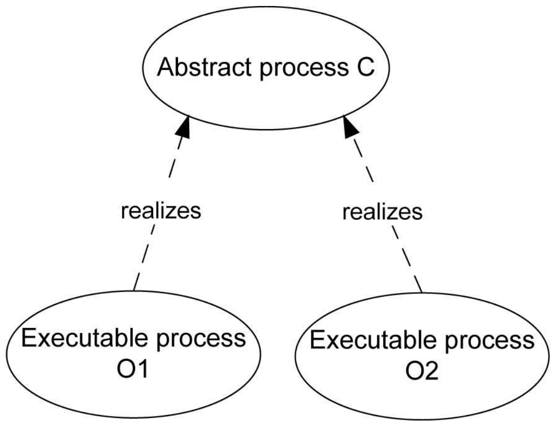


Fig. 5. A typical scenario of Web Service composition

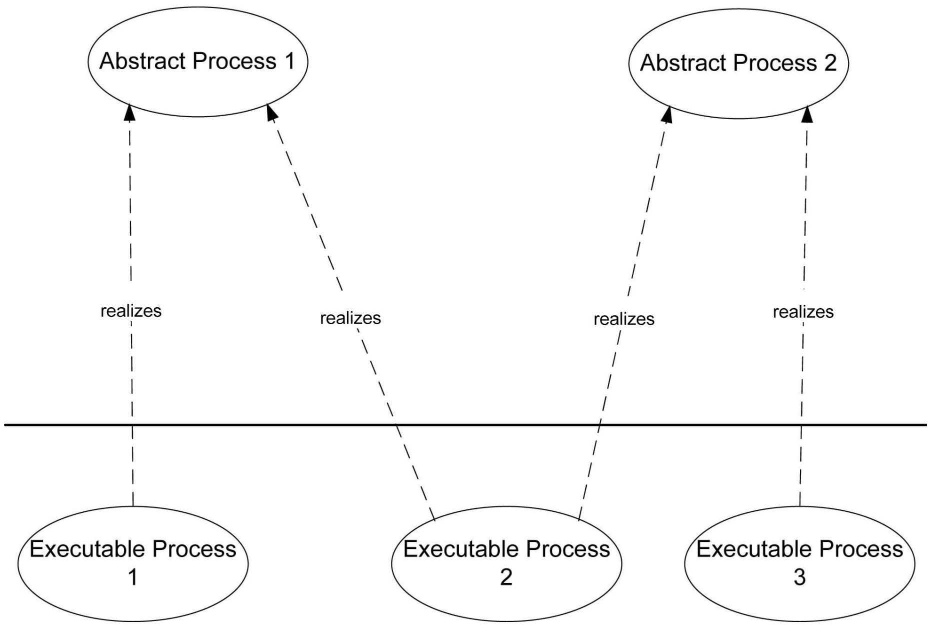


Fig. 6. A special case of the generic architecture in fig. 4

If two nodes have no common activities, these two nodes are temporally independent and their temporal plans can be calculated in isolation. If GCD of two nodes is not empty these two nodes affect each other through the common activities. In this case a cycle of calculation-assignment-recalculation must be repeated. In the assignment phase, temporal values of the activities of GCD are assigned from the source node to the target node. The assignment is only performed if e-values of the source node are greater than e-values of the target node. l-values are assigned if l-values of the source node are smaller than those of the target node. In other words an assignment is only allowed if the current valid execution interval of an activity in the target node becomes tighter. The idea of assignment is depicted in fig. 7. The assignment uses the concept of event correspondence. Event correspondence describes if different events in different graphs are actually the same. $e_1 \equiv e_2$ denotes that event e_1 corresponds to event e_2. Note that e_1 and e_2 may belong to different abstract and/or executable processes. In the top part of the fig. 7 (case a), event correspondence is used for propagation of the temporal values of a complex activity a. Start of the complex activity a corresponds to the start of the first activity i and its end to the activity j. Case b shows how event correspondence is used for propagation of the temporal values of the same activity in different abstract and executable processes.

If any temporal value changes after the assignment, the temporal plan of the graph must be recalculated. Again here, current values can be overwritten if newly calculated e-values are greater than current values and newly calculated l-values are smaller than current l-values. How the cycle of calculation-assignment-recalculation works can be seen in fig. 8. The dependency between these processes is illustrated in fig. 5. The top part of the fig. 8 illustrates the graphs after calculation of *C* and *O1* and assignment of values from *C* to *O1*. The bottom part of the figure illustrates the values after recalculation at *O1*, assignment of the values from *O1* to *C*. A recalculation at *C* does not change the values at *C* and hence these values are final values. As you can see the same activities in different graphs have the same final temporal values. The arrows only shows the assigned values. For example in the top part of the figure the e-values of the activity *Receive request* in the abstract process *C* are equal to those in the executable process O1 and hence not assigned.

Calculation of a set of cooperating abstract and executable processes consists of two phases: the initialization and precalculation phase and the recalculation and conformance checking phase. In the following we refer to the abstract process with no outgoing links, the *global abstract process* (e.g. abstract process 1 in fig. 4).

In the initialization and precalculation phase after initialization of the global abstract process, its graph is calculated without considering other nodes. That means only implicit and explicit constraints are considered. If there is more than one global abstract process present in the model, any of them can be chosen randomly for beginning the calculations. The temporal graphs are initialized by setting e-values to 0 and l-values to ∞. Maximum duration *d.max* is considered for calculating the deadline of other nodes than the global abstract process. Analogous to the maximum duration of activities, maximum duration of the whole process identifies the maximum duration during which a process can execute whereas a deadline denotes a point in time. Like deadlines, *d.max* is given a 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

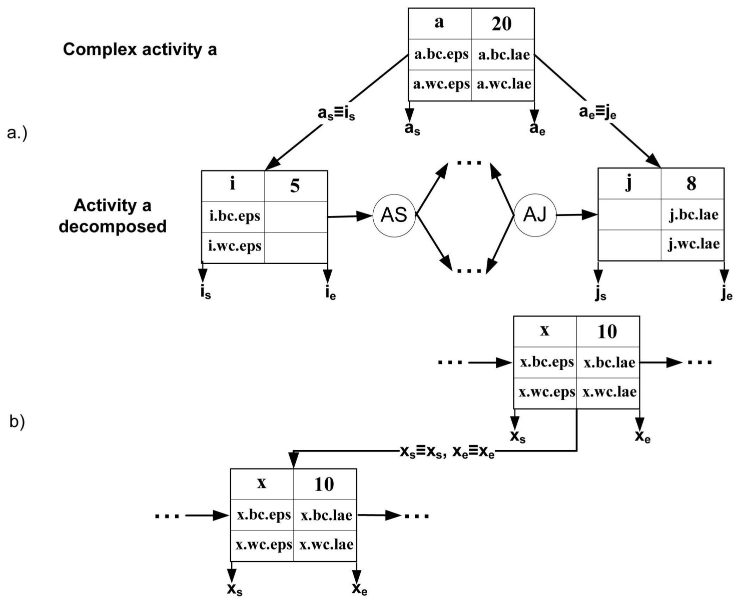


Fig. 7. Assignment of values using event correspondence

change indicates the change of a graph, i.e. if any temporal value in a graph is changed due to assignment and/or recalculation. 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 graphs 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. The conformance condition is violated if e-value of an activity (no matter in which process) becomes greater that its corresponding l-value. .

The probabilistic approach, as well, uses the same concept for assignment and calculation of temporal plans. Due to the presence of probabilities in this approach an additional parameter *certainty* must be considered for comparison of nodes and propagation of values. A detailed discussion can be found in [15]. Note that in addition to temporal conformance, other conformance issues of processes may also be verified. Structural conformance has been studies in [6].

4 Prototypical Implementation

Our prototype has been implemented based on the open source softwares Eclipse BPEL designer and Apache ODE (Orchestration Director Engine). The proto-

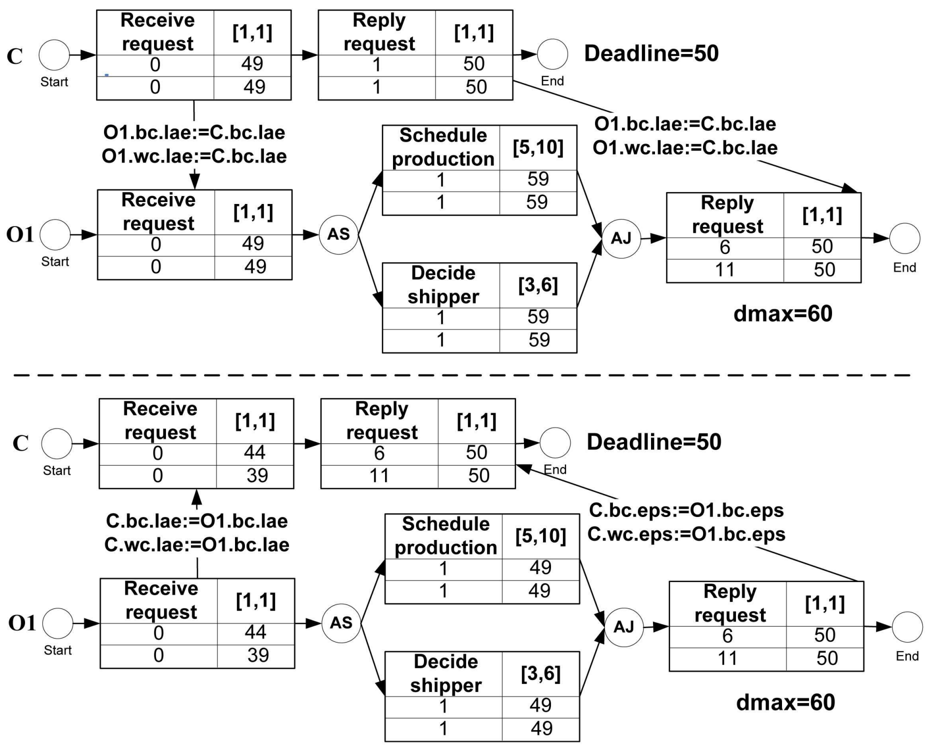


Fig. 8. Propagation of values for the same activity in different graphs

type consists of two components: A design time component that allows the definition of cooperating processes, their dependencies and temporal constraints (deadline, activity durations, lower and upper bound constraints between activities). The design time component also checks the temporal satisfiability of the model. It checks if there exists a solution that satisfies temporal constraints of all processes. If such a solution does not exist the user will be informed which part of the process has temporal conflicts. In this case process structure or temporal constraints should be redefined such that the temporal constraints can be satisfied. Design time component calculates a valid temporal window for each activity. If activities execute within their valid temporal window it can be guaranteed that all cooperating processes execute and terminate in a temporally compliant way. The run time component monitors the execution of each activity and checks if it executes within its valid temporal window. If any deviation is found the user receives an alarm about the process status. We use the traffic light model presented in [7]. If all activities executes within their valid temporal window the traffic light is green and everything is ok. If some activities deviate from their precalculated valid temporal window but it is still possible to hold the deadline and satisfy the temporal constraints (e.g. by executing the shortest path of a conditional structure) the traffic light turns to yellow. If some activities took longer than expected and in any case, even in best case scenario, temporal constraints will be violated, the traffic light turns to red. In this case the process manager can decide to cancel the execution prematurely or skip some activities. The prototype, an installation and troubleshooting guide and an introduction how to perform basic tasks such as defining BPEL-process, preparing wsdl-files and setting up variables can be found on our homepage [1].

4.1 Design Time Component

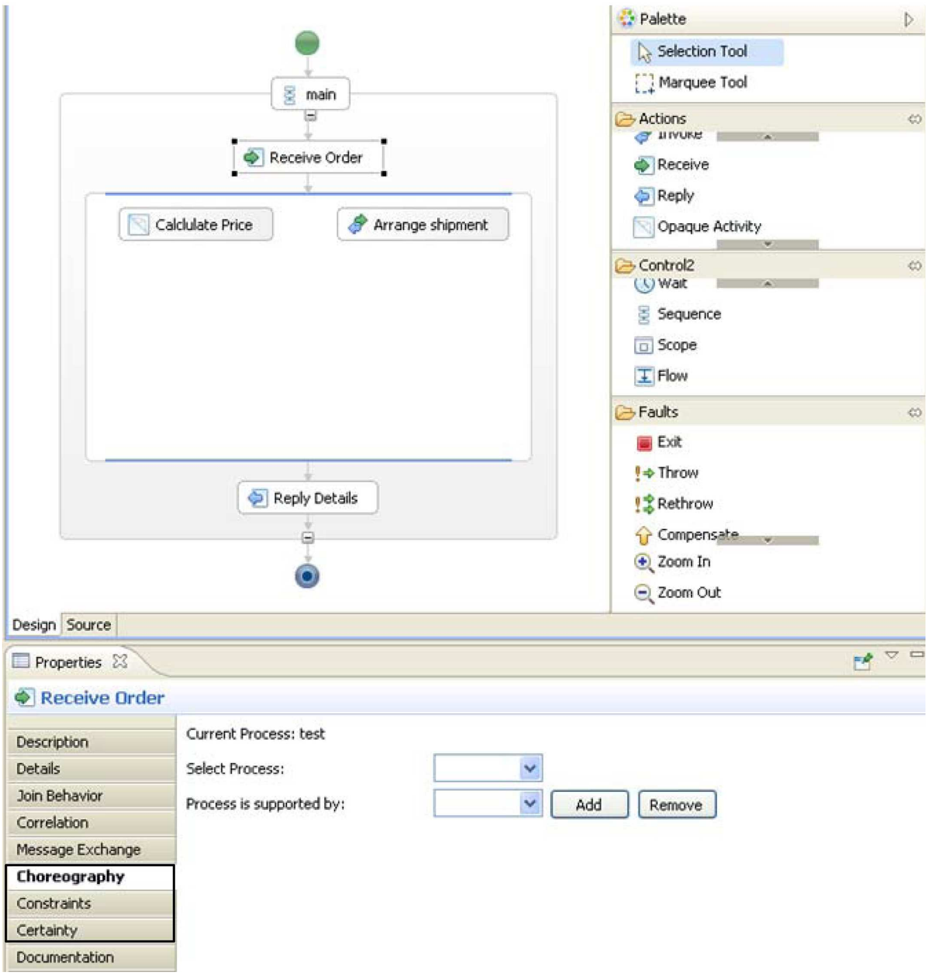
The design time component is prototyped under Eclipse Helios. The required functionalities are implemented under *properties* as depicted in fig. 9. After definition of the structure of the processes, the dependencies between processes can be defined using the property *choreography.* A supported process can be chosen using *Select Process.* A supported process identifies processes that have a link to this process, i.e. their GCD is not empty. This can be an executable process or another abstract process. The combo box beneath allows for choosing processes that support this process. Dependencies can be added or removed. The result is written in an XML-file called dependencies.xml.

Temporal constraints can be set under *Constraints* (see fig. 10). It is possible to set minimum and maximum duration of activities and the deadline for the process. Further it is possible to add and remove optional lower and upper bound constraints between activities. The right part of the fig. 10 shows the temporal values of each activity after calculation.

Certainty allows the definition of probabilistic values: durations of activities and their probabilities as well as the deadline of the whole process. The probabilistic temporal values can be calculated by the *Calculate* button.

4.2 Run Time Component

The run time component is implemented in Apache ODE 1.3.4. It monitors the process execution and checks if activities are executed within their valid temporal interval. At process instantiation time, an actual calendar is used in order to transform all time information which was computed relative to the start of the flow to absolute time points [8]. For every instantiated activity, the calendar e-value is compared with the start date of the instantiated activity. In the same way, the mapped calendar l-value is also compared with the end date of the instantiated activity. The traffic light model [7] provides an overview for process manager. If activities execute within their calculated interval at design time it is guaranteed that all processes remain temporally compliant. If some activities are delayed and deviate from their valid temporal window, it is possible that a deadline be violated. In this case the traffic light turns to yellow indicating that some activities are delayed but it is still possible to finish the execution in time. The process manager may decide to force the process to execute the shortest path in order to guarantee the temporal compliance. If activities are delayed



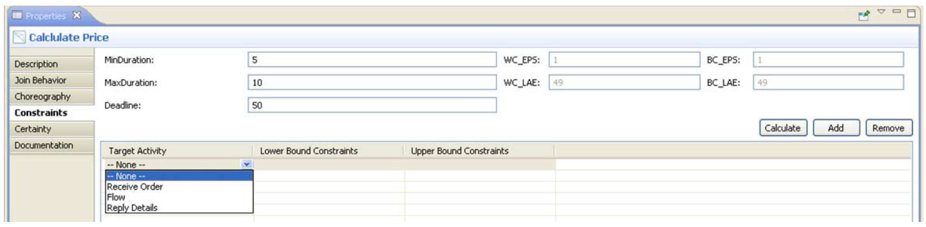
Current Process: test
Select Process:
Process is supported by:

Fig. 9. Definition of dependencies between processes

to the extent that future execution in any case leads to temporal violation, the traffic light turns to red. In this case, the process manager may want to cancel execution prematurely in order to reduce the process execution costs. Fig. 12 shows a screen shot of the run time component.

5 Conclusions

Temporal management and consistency are important quality criteria for business processes. They improve QoS and reduce costs. WS-BPEL lacks sufficient time management capabilities. In this work we introduced an extension of WS-BPEL that makes business processes time aware and overcomes this shortcoming.

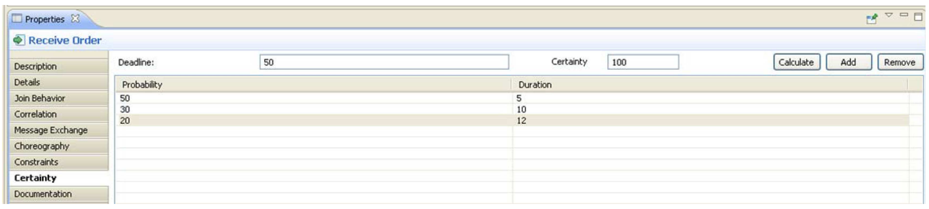


MinDuration: 5
WC_EPS: 1
BC_EPS: 1
MaxDuration: 10
WC_LAE: 49
BC_LAE: 49
Deadline: 50

Target Activity	Lower Bound Constraints	Upper Bound Constraints
-- None --		

-- None --
Receive Order
Flow
Reply Details

Fig. 10. Definition and calculation of interval-based temporal values



Deadline: 50
Certainty 100

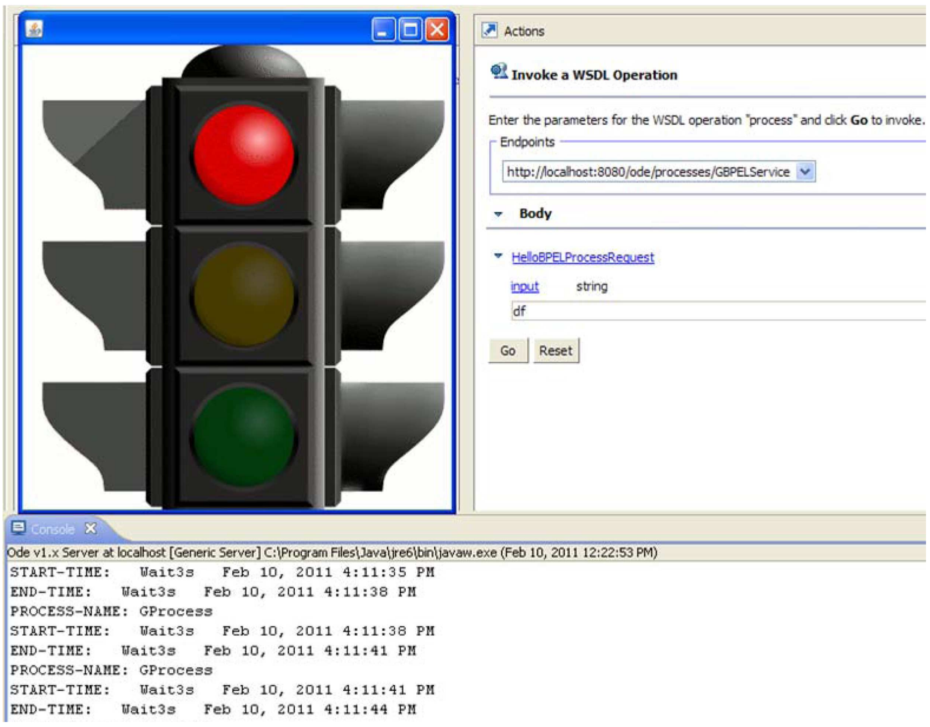
Probability	Duration
50	5
30	10
20	12

Fig. 11. Definition and calculation of probabilistic temporal values

The user can define different temporal constraints, check temporal feasibility and monitor the execution. The two considered techniques, interval-based and probabilistic, caters for different needs of the users in different scenarios. We have described how we handle temporal management of a set of cooperating abstract and executable processes.

References

1. http://www.ifs.tuwien.ac.at/.
2. Workflow process definition interface - xml process definition language. Technical Report WFMC-TC-1025, The Workflow Management Coalition, 2002.
3. G. Decker, H. Overdick, and J.M. Zaha. On the suitability of ws-cdl for choreography modeling. In *Proc. of EMISA'06*, 2006.
4. R.M. Dijkman and M. Dumas. Service-oriented design: A multi-viewpoint approach. *Int. J. Cooperative Inf. Syst.*, 13(4):337–368, 2004.
5. J. Eder, M. Lehmann, and A. Tahamtan. Choreographies as federations of choreographies and orchestrations. In *Proc. of CoSS'06*, 2006.
6. J. Eder, M. Lehmann, and A. Tahamtan. Conformance test of federated choreographies. In *Proc. of I-ESA'07*, 2007.
7. J. Eder and E. Panagos. *WfMC WorkFlow Handbook 2001*, chapter Managing Time in Workflow Systems. J. Wiley & Sons, 2001.
8. J. Eder, E. Panagos, and M. Rabinovich. Time constraints in workflow systems. In *Proc. of CAiSE*, 1999.
9. J. Eder and H. Pichler. Duration histograms for workflow systems. In *Proc. of the IFIP TC8 / WG8.1 Working Conference on Engineering Information Systems in the Internet Context*, 2002.



Enter the parameters for the WSDL operation "process" and click **Go** to invoke.

Endpoints: http://localhost:8080/ode/processes/GBPELService

HelloBPELProcessRequest

input string

df

```
Ode v1.x Server at localhost [Generic Server] C:\Program Files\Java\jre6\bin\javaw.exe (Feb 10, 2011 12:22:53 PM)
START-TIME:    Wait3s   Feb 10, 2011 4:11:35 PM
END-TIME:    Wait3s   Feb 10, 2011 4:11:38 PM
PROCESS-NAME: GProcess
START-TIME:    Wait3s   Feb 10, 2011 4:11:38 PM
END-TIME:    Wait3s   Feb 10, 2011 4:11:41 PM
PROCESS-NAME: GProcess
START-TIME:    Wait3s   Feb 10, 2011 4:11:41 PM
END-TIME:    Wait3s   Feb 10, 2011 4:11:44 PM
```

Fig. 12. The run time component monitors the execution

10. J. Eder, H. Pichler, and A. Tahamtan. Probabilistic time management of choreographies. In *Proc. of QSWS-08*, 2008.
11. J. Eder and A. Tahamtan. Temporal conformance of federated choreographies. In *Proc. of DEXA'08*, 2008.
12. A. Alves et al. Web services business process execution language version 2.0. Technical report, OASIS, 2007.
13. H. Pichler. *Time Management for Workflow Systems. A probabilistic Approach for Basic and Advanced Control Flow Structures.* PhD thesis, Alpen-Adria-Universitaet Klagenfurt, 2006.
14. A. Tahamtan. *Modeling and Verification of Web Service Composition Based Interorganizational Workflows.* PhD thesis, University of Vienna, 2009.
15. A. Tahamtan, C.Oesterle, A. Tjoa, and A. Hameurlain. Bpel-time: Ws-bpel time management extension. In *Proc. of ICEIS'11*, 2011.
16. A. Tahamtan and J. Eder. View driven interorganizational workflows. *Int. J. Intelligent Information and Database Systems, To Appear.*
17. A. Tahamtan and J. Eder. View driven federation of choreographies. In *Proc. of ACiiDS'10*, 2010.