



ELSEVIER

The Journal of Systems and Software 55 (2001) 231–243

 **The Journal of
Systems and
Software**

www.elsevier.com/locate/jss

A timed workflow process model

Hai Zhuge^{a,c,*}, To-yat Cheung^b, Hung-keng Pung^c

^a *Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100080, People's Republic of China*¹

^b *Department of Computer Science, City University of Hong Kong, Hong Kong*

^c *Department of Computer Science, National University of Singapore, Singapore*

Received 12 February 2000; received in revised form 25 May 2000; accepted 14 July 2000

Abstract

An internet-based workflow management system (WfMS) enables business participants to work co-operatively at sites belonging to different time zones. Time-related factors have to be incorporated into the traditional workflow processes so as to adapt to the globally distributed applications. This paper proposes a timed workflow process model through incorporating the time constraints, the duration of activities, the duration of flow, and the activity distribution with respect to the multiple time axes into the conventional workflow processes. The model provides an approach for temporal consistency checking during both build-time and run-time. The proposed model and approach provide a vehicle for global business process modeling, planning and monitoring. © 2001 Elsevier Science Inc. All rights reserved.

Keywords: Consistency checking; Distributed system; Duration; Time; Workflow

1. Introduction

Workflow is the computerized facilitation or automation of a business process, in whole or part (WfMC, URL). A workflow management system (WfMS) is a system that completely defines, executes and manages the workflow through the execution of software whose execution order is driven by a computer representation of the workflow logic (WfMC, URL). A generic workflow reference model has been suggested in (WfMC, URL). The benefits of workflow-based applications, like flexibility, integration and reusability, were discussed (Georgakopoulos and Hornick, 1995).

A WfMS usually consists of three parts: (1) the build-time functions, which are responsible for defining and modeling the workflow processes; (2) the run-time functions, which are responsible for executing, monitoring and managing the workflow processes in an operational environment; and (3) the run-time interactions with human users and application mechanisms for processing the activities (tasks).

A workflow process model is an abstraction of the business process logic and the dependence relationships (transition or flow) among the business activities. A dependence relationship in traditional workflow processes can be of two types: execution dependence for control flow and information dependence for information flow (Lawrence, 1997; WfMC, URL). In some applications (e.g., e-commerce), material dependence (i.e., material flow) is also another type required.

To support global business applications, time-dependent factors have to be incorporated into both the build-time process definition and the run-time management system. The first reason is that activities of the workflow may belong to different time zones. The temporal order and the time difference need to be incorporated into the logic of the activities at the build-time. We can also make use of the time difference to reduce the duration of the whole workflow process. The second reason is that any activity will take certain execution duration. Flows will also take certain transmission durations from one site to another. Flow duration may be longer than the execution duration of the related activities, e.g., the duration of the material flow for delivering products from the supplier to the customer.

* Corresponding author.

E-mail address: haizhug@hotmai.com (H. Zhuge).

¹ Correspondence address.

Traditional workflow process models do not pay much attention to the time relevant factors (WfMC, URL; Leymann and Roller, 1997; Puustjarvi, 1997). Recently, the time modeling of workflow processes has attracted increasing attention (Eder et al., 1999; Geppert et al., 1998; Marjanovic and Orłowska, 1999). These previous approaches are only based on the activity-related time duration and constraints. They neglect: the flow-relevant time factor, the time difference due to geographically different location, the relationship between the logic order and the time order, and the relationship between the build-time duration (constraint) and the run-time duration. These missing hindered them from supporting the consistent checking (the time constraint satisfactory and the logic order consistent with the time order in both the build-time and the run-time), and the workflow process planning for the real globally distributed business process.

Other research works on time management include active databases, the real-time data management and temporal databases (or knowledge base) (Baekgaard and Godsken, 1998; Bestougeff and Ligozat, 1992; Botti et al., 1998; Casati et al., 1999; Jensen and Snodgrass, 1999; Lee et al., 1998), where the time interval specifies the data life period and keeps the temporal consistency in data model. A time-stamp is usually used to mark the time of an event occurrence (Casati et al., 1999). The other time-related research works concern timed Petri nets (Liu, 1998; Tsai et al., 1995) and time management e.g., project management and temporal constraint networks (Dechter et al., 1991). The difference between the workflow and the project management has been discussed (Marjanovic and Orłowska, 1999; Rose, 1998). The current commercial products offer very limited support for time representation and management (Marjanovic and Orłowska, 1999). These approaches are not eligible for the timed process modeling of the globally distributed business process.

This paper proposes a timed workflow process model for handling globally distributed business processes. We first investigate some order relationships and the duration of workflow processes and define the basic components of the model. Then, the timed workflow process and the approaches for build-time consistency checking are described. The run-time consistency checking approach and the process planning approach are proposed based on the build-time approach. An example is used to demonstrate the building of the timed workflow process and the approaches for consistent checking and process planning. Through comparison and discussion we show the advantage of the proposed approach.

2. Order, duration, and constraints in temporal workflow processes

2.1. Time, time order, and dependence order

From the computer's point of view, time is a linear variable related to the time zones. A complete time variant can be represented by a time type such as year-month-date-hour:minute:second(zone)week, e.g., 1999-Jul-23-11:03:56(HKT)Fri. In businesses, the managers usually have to estimate the time duration of a process (including the activities and flows) and check whether the process is executed under certain build-time and run-time time constraints.

Business activities of a workflow process may be carried out in different time zones. Hence, a time axis is needed to represent the time advancement of the activities within the same time zone. We use $axis_k$ ($k \in [1, n]$) to identify n axes, $axis_k(i)$ to denote the time axis of activity A_i , $TD(axis_i, axis_j)$ to denote the time difference between $axis_i$ and $axis_j$ (TD in simple), and a function $TM(\langle \text{time}, axis_i \rangle, axis_j)$ to transform between two time axes, $axis_i$ into $axis_j$.

Time order is the order in which events occur. When comparing time orders, we map the times in different time onto the same (reference) time axis. Formally, time order (denoted as \leq_t) is a partial order relationship on a set of $\langle \text{time}, axis \rangle$ pairs. $\langle \text{time}_i, axis_i \rangle \leq_t \langle \text{time}_j, axis_j \rangle$ if and only if (1) $\text{time}_i \leq_t \text{time}_j$ if $axis_i = axis_j$; and (2) $\text{time}_i \leq_t TM(\langle \text{time}_j, axis_j \rangle, axis_i)$ if $axis_i \neq axis_j$.

A dependence order (denoted as \leq_D) is an execution-dependent logic relationship among the activities involved. The order allows an activity to be executed only after the executions of its predecessors have been completed. Traditional workflow process models mainly reflect such dependence orders among the activities. A dependence order among a set of activities should be consistent with their temporal order.

Definition 1. A workflow process is called the order consistent if for any two activities A_i and A_j in the process we have: $A_i \leq_D A_j$ if and only if $A_i \leq_t A_j$ holds.

2.2. Timed activity and timed flow

In build-time, an activity (denoted as A) should be assigned an execution time axis and given an estimated build-time duration. This duration can be reflected by a maximum value $D(A)$ and a minimum value $d(A)$. A flow F_{ij} is the

passing of control or data from activity A_i to activity A_j . The build-time duration of F_{ij} is an estimation on the time for passing the flow, which can be reflected by a maximum duration $D(F_{ij})$ and a minimum duration $d(F_{ij})$.

In run-time, a timed activity has a start time (denoted as $S(A)$) and an end time (denoted as $E(A)$). The run-time duration, $D_R(A)$, is defined as $D_R(A) = E(A) - S(A)$. A timed activity is called active during the period from the beginning to the end of its run-time. The resources required by the activity are only occupied during its active period. We use $D_R(A_i, A_j)$ to denote the run-time duration between the start time of A_i to the end time of A_j (time difference needs to be considered when A_i and A_j belong to different time zone) which satisfy $A_i \leq_d A_j$.

The run-time flow duration $D_R(F_{ij})$ is defined as the time interval from the time of the flow leaving activity A_i to the time of arrival activity A_j . Since A_i and A_j can be distributed onto the places belonging to two different time zones, the computation of $D_R(F_{ij})$ needs to consider the time transformation from one zone into another as follows:

$$D_R(F_{ij}) = \begin{cases} S(F_{ij}) - E(F_{ij}) & \text{if } A_i \text{ and } A_j \text{ are on the same axis;} \\ TM(\langle E(F_{ij}), \text{axis}(j) \rangle, \text{axis}(i)) - S(F_{ij}) & \text{if } A_i \text{ and } A_j \text{ are on different axes and axis}(i) \text{ is the reference axis;} \\ E(F_{ij}) - TM(\langle S(F_{ij}), \text{axis}(i) \rangle, \text{axis}(j)) & \text{if } A_i \text{ and } A_j \text{ are on the different axis and axis}(j) \text{ is the reference axis.} \end{cases}$$

If no exception happens, we have: $d(A) \leq D_R(A) \leq D(A)$; $d(A) \leq D_R(F_{ij}) \leq D(A)$; $S(F_{ij}) = E(A_i)$; and $E(F_{ij}) = S(A_j)$ holds. In case of exception happens, the duration of the exception handling can be regarded as a part of the duration of relevant activity or relevant flow according to where the exception happens.

2.3. Super-constraints on time constraints

Various kinds of time constraints have been discussed, such as the fixed-point constraint, the lower bounds and upper bounds on the durations, and the interdependency constraints (Eder et al., 1999; Marjanovic and Orłowska, 1999; Zhuge et al., 2000). The fixed-point constraint is the time point of the workflow process should be on, before, or after a pre-determined absolute time value. The duration constraint is the duration between any two points of a workflow is less than a constant duration or a duration range. The interdependent constraint is a constraint that is relative to a previous constraint. It is a combination of the previous two kinds of constraints.

Users set the time constraints according to domain requirement. Unreasonable constraints may occur. To reduce the unreasonable constraints, we need to set super-constraints set according to the logic order of the workflow process and to check the time constraints according to the super-constraints. We herein focus on the super-constraints on the fixed-point constraint and the duration constraint.

(1) *Super-constraint on fixed-point constraints.* Given a set of fixed-point constraints, then any two of them should satisfy the duration constraint between the two points. For example, given $S(A_i) \leq p_i$ and $S(A_j) \leq p_j$, and $A_i \leq_d A_j$, we have two super-constraints on the fixed-point constraints: $D_R(A_i, A_j) - D(A_j) \leq p_j - p_i$; and $d(A_i, A_j) - D(A_j) \leq (p_j - p_i) \leq D(A_i, A_j) - D(A_j)$.

(2) *Super-constraint on duration constraints.*

Case 1. Given a duration constraint $D_R(A_i, A_j) \leq p_j$, then we have a super-constraint on the duration constraint $d(A_i, A_j) \leq p_j \leq D(A_i, A_j)$. If A_j is an ‘and-join’ node, i.e., there exists another path from A_i to A_j through A_k such that F_{kj} is the input flow of A_j , then we have another two super-constraints on the duration constraint $D_R(A_i, A_k)$ depends on $D_R(A_i, A_k)$; and $D_R(A_i, A_k) + D_R(F_{kj}) + D(A_j) \leq p_j$.

Case 2. Given three duration constraints: $D_R(A_i, A_m) \leq p_1$, $D_R(A_i, A_j) \leq p_2$, $D_R(A_l, A_m) \leq p_3$, and the logic order $A_i \leq_d A_j \leq_d A_l \leq_d A_m$, then we have the super-constraint on the duration constraints $p_1 \leq p_2 + p_3 + D(A_j, A_l)$.

3. Basic components of timed workflow process model

A globally distributed workflow process can include several time axes leveled top-down in terms of their time difference. One of these axes should be referred as the reference axis. The time on the other axes will be mapped onto the referent axis when making time comparisons. We herein just focus on the case that has two different time axes (named as axis_1 and axis_2) and assume axis_1 to be the referent axis. The cases of having more time axes can be similarly designed.

3.1. Timed sequential connection and timed parallel connection

Sequential routing (i.e., sequential connection herein) is defined as a segment of a process instance, where activities are executed in sequence (WfMC, URL). In timed workflow, the sequential connection between two activities A_i and A_j (denoted as $A_i \cdot A_j$) can be carried out in two cases: one is that the two activities are on the same axis (case 1 in Fig. 1)

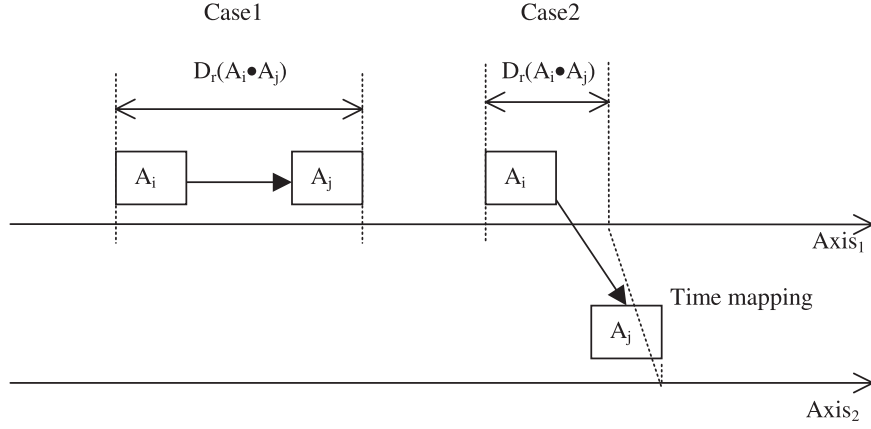


Fig. 1. Sequential connection of two activities across two axes.

and another is that two activities are on two different axes, respectively (case 2 in Fig. 1). The build-time (or run-time) duration of the sequential connection is determined by the build-time (or run-time) duration of the two activities and that of the flow; and the time difference (if the two activities execute at two different axes, respectively). The duration computation of case 1 does not need to consider the time difference between the two axes (i.e., $TD = 0$), while case 2 needs to consider the time difference. The run-time duration ($D_R(A_i \cdot A_j)$) and the build-time duration ($D(A_i \cdot A_j)$ and $d(A_i \cdot A_j)$) of the sequential execution of two activities A_i and A_j are defined as follows:

$$D_R(A_i \cdot A_j) = \begin{cases} E(A_j - S(A_i)) & \text{if } A_i \text{ and } A_j \text{ are on the same axis;} \\ TM(\langle E(A_j), \text{axis}_2 \rangle, \text{axis}_1) - S(A_i) & \text{if } A_i \text{ and } A_j \text{ are on different axes.} \end{cases}$$

$$D(A_i \cdot A_j) = \begin{cases} D(A_i) + D(F_{ij}) + D(A_j) & \text{in case 1;} \\ D(A_i) + D(F_{ij}) + D(A_j) - TD & \text{in case 2;} \end{cases}$$

and

$$d(A_i \cdot A_j) = \begin{cases} d(A_i) + d(F_{ij}) + d(A_j) & \text{in case 1;} \\ d(A_i) + d(F_{ij}) + d(A_j) - TD & \text{in case 2.} \end{cases}$$

Parallel routing (i.e., parallel connection herein) is defined as a segment of a workflow instance, where workflow activity instances are executing in parallel and there are multiple threads of control (WfMC, URL). The execution durations of two parallel activities (denoted as $A_i || A_j$) are overlapped. In timed workflow model, the run-time duration of the parallel execution of two activities A_i and A_j (denoted as $D_R(A_i || A_j)$) is determined according to two cases: case 1: A_i and A_j are on the same axis; and case 2: A_i is on the axis₁ and A_j is on the axis₂.

$$D_R(A_i || A_j) = \begin{cases} \text{Max}\{E(A_i), E(A_j)\} - \text{Min}\{S(A_i), S(A_j)\} & \text{in case 1;} \\ \text{Max}\{E(A_i), TM(\langle E(A_j), \text{axis}_2 \rangle, \text{axis}_1)\} - \text{Min}\{S(A_i), TM(\langle S(A_j), \text{axis}_2 \rangle, \text{axis}_1)\} & \text{in case 2.} \end{cases}$$

The maximum and the minimum build-time duration of the parallel execution of two activities A_i and A_j can be computed by $D(A_i || A_j) = D(A_i) + D(A_j)$ and $d(A_i || A_j) = \text{Max}\{d(A_i), d(A_j)\}$.

3.2. Timed ‘and-split’ connection and timed ‘or-split’ connection

In traditional workflow process, ‘and-split’ is defined as a single thread of control splits into two or more parallel activities (WfMC, URL). After taking time into account (especially the flow duration) the definition needs to be adapted. The ‘and-split’ successors may not be parallel in time order. They may execute sequentially in time order, e.g., A_i and A_k as shown in the first case of Fig. 2. So ‘and-split’ just defines the execution dependence structure that two or more activities all execute after the execution of a common predecessor. Assume two activities A_j and A_k are the and-split successor of activity A_i , denoted as $A_i \cdot (A_j \wedge A_k)$. There are three cases of such an ‘and-split’ as shown in Fig. 2. The run-time duration and the build-time duration of such an and-split execution is described as follows:

$$D_R(A_i \cdot (A_j \wedge A_k)) = \begin{cases} \text{Max}\{TM(\langle E(A_j), \text{axis}_2 \rangle, \text{axis}_1), TM(\langle E(A_k), \text{axis}_2 \rangle, \text{axis}_1)\} - S(A_i) & \text{in case 1;} \\ \text{Max}\{TM(\langle E(A_j), \text{axis}_2 \rangle, \text{axis}_1), E(A_k)\} - S(A_i) & \text{in case 2;} \\ \text{Max}\{E(A_j), E(A_k)\} - S(A_i) & \text{in case 3.} \end{cases}$$

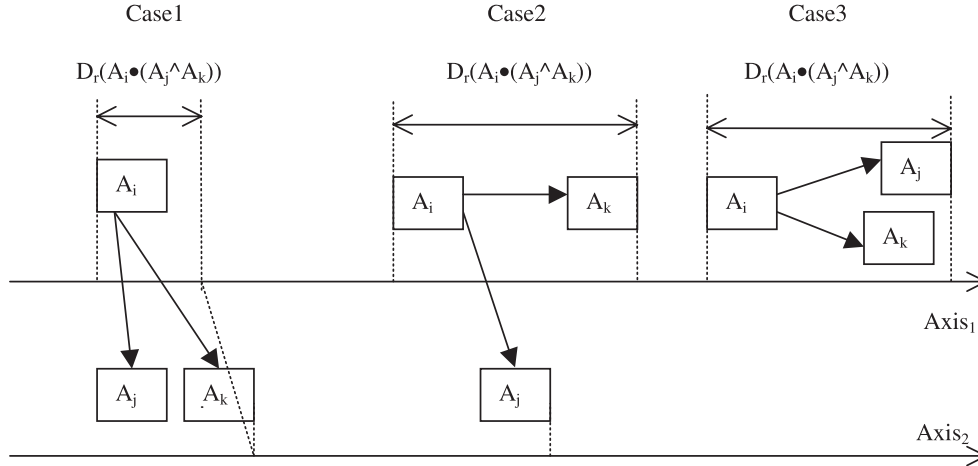


Fig. 2. ‘And-split’ connection of three activities across two axes.

$$D(A_i \cdot (A_j \wedge A_k)) = \begin{cases} \text{Max}\{D(A_i) + D(F_{ij}) + D(A_j) - TD, D(A_i) + D(F_{ik}) + D(A_k) - TD\} & \text{in case 1;} \\ \text{Max}\{D(A_i) + D(F_{ij}) + D(A_j) - TD, D(A_i) + D(F_{ik}) + D(A_k)\} & \text{in case 2;} \\ \text{Max}\{D(A_i) + D(F_{ij}) + D(A_j), D(A_i) + D(F_{ik}) + D(A_k)\} & \text{in case 3.} \end{cases}$$

$$d(A_i \cdot (A_j \wedge A_k)) = \begin{cases} \text{Max}\{d(A_i) + d(F_{ij}) + d(A_j) - TD, d(A_i) + d(F_{ik}) + d(A_k) - TD\} & \text{in case 1;} \\ \text{Max}\{d(A_i) + d(F_{ij}) + d(A_j) - TD, d(A_i) + d(F_{ik}) + d(A_k)\} & \text{in case 2;} \\ \text{Max}\{d(A_i) + d(F_{ij}) + d(A_j), d(A_i) + d(F_{ik}) + d(A_k)\}, & \text{in case 3.} \end{cases}$$

Or-split connection is defined as a single thread of control makes a decision upon which branch to take when encountered with multiple threads of branches (WfMC, URL). In other words, or-split connection is an execution dependence structure that one or more activities $(A_{j1}, A_{j2}, \dots, A_{jn})$ is selected to be executed after the execution of a predecessor activity A_i , denoted as $A_i \cdot (A_{j1} \vee A_{j2} \dots \vee A_{jn})$. The or-split connection of three activities also has the three cases like the and-split connection shown in Fig. 2. The run-time duration and the build-time duration of $A_i \cdot (A_j \vee A_k)$ can be computed in terms of three different cases like the ‘and-split’:

$$D_R(A_i \cdot (A_j \vee A_k)) = \begin{cases} TM(\langle E(A_j), \text{axis}_2 \rangle, \text{axis}_1) - S(A_i) \text{ or } TM(\langle E(A_k), \text{axis}_2 \rangle, \text{axis}_1) - S(A_i), & \text{case 1;} \\ TM(\langle E(A_j), \text{axis}_2 \rangle, \text{axis}_1) - S(A_i) \text{ or } E(A_k) - S(A_i), & \text{case 2;} \\ E(A_j) - S(A_i) \text{ or } E(A_k) - S(A_i), & \text{case 3.} \end{cases}$$

$$D(A_i \cdot (A_j \vee A_k)) = \begin{cases} \text{Max}\{D(A_i) + D(F_{ij}) + D(A_j) - TD, D(A_i) + D(A_k) + D(F_{ik}) - TD\}, & \text{case 1;} \\ \text{Max}\{D(A_i) + D(F_{ij}) + D(A_j) - TD, D(A_i) + D(F_{ik}) + D(A_k)\}, & \text{case 2;} \\ \text{Max}\{D(A_i) + D(F_{ij}) + D(A_j), D(A_i) + D(F_{ik}) + D(A_k)\}, & \text{case 3.} \end{cases}$$

$$d(A_i \cdot (A_j \vee A_k)) = \begin{cases} \text{Max}\{d(A_i) + d(F_{ij}) + d(A_j) - TD, d(A_i) + d(F_{ik}) + d(A_k) - TD\}, & \text{case 1;} \\ \text{Max}\{d(A_i) + d(F_{ij}) + d(A_j) - TD, d(A_i) + d(F_{ik}) + d(A_k)\}, & \text{case 2;} \\ \text{Max}\{d(A_i) + d(F_{ij}) + d(A_j), d(A_i) + d(F_{ik}) + d(A_k)\}, & \text{case 3.} \end{cases}$$

3.3. Timed ‘and-join’ connection and ‘or-join’ connection

And-join is defined as two or more parallel executing activities converge into a single common thread of control (WfMC, URL). After taking time into account the definition needs to be adapted. The previous meaning of the parallel executing activities may change to be executed sequentially in time order, e.g., A_i and A_j as shown in the first case of Fig. 3. So, ‘and-join’ just defines the execution dependence structure that an activity executes after two or more predecessors. Fig. 3 shows an and-join connection, where activity A_k executes after two activities A_i and A_j , denoted as $(A_i \wedge A_j) \cdot A_k$. In traditional time modelling of workflow model, flow duration is not taken into account, so A_k needs to

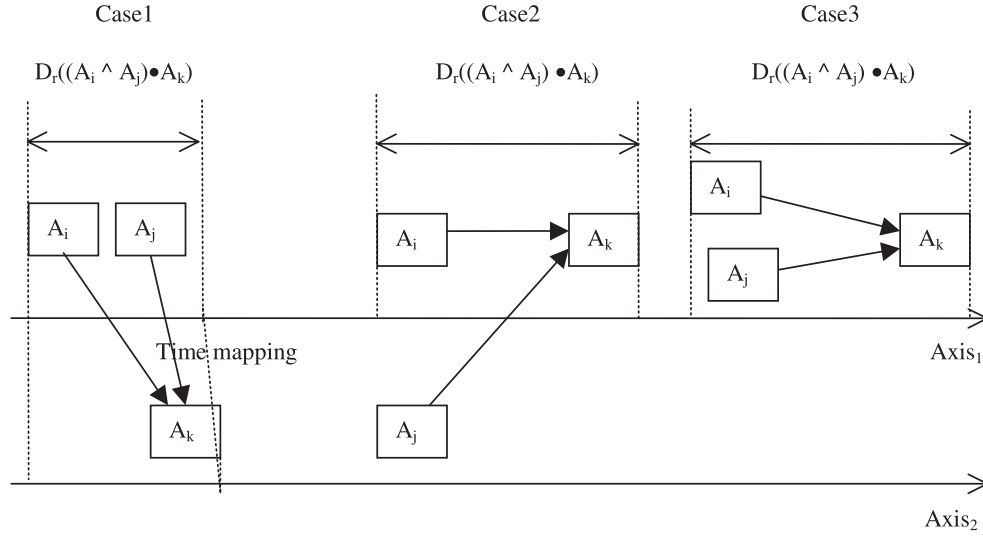


Fig. 3. 'And-join' connection of three activities across two axes.

wait for $\text{Max}\{d(A_i), d(A_j)\}$. But it is not true after taking the flow duration into account because the flow that starts earlier may end later. A_k needs to wait for $\text{Max}\{S(F_{ik}) + D(F_{ik}), S(F_{jk}) + D(F_{jk})\}$. The run-time duration and the build-time duration (maximum duration and minimum duration) of such a connection can be computed as follows:

$$D_R((A_i \wedge A_j) \cdot A_k) = \begin{cases} TM(\langle E(A_k), \text{axis}_2 \rangle, \text{axis}_1) - \text{Min}\{S(A_i), S(A_j)\} & \text{in case 1;} \\ E(A_k) - \text{Min}\{S(A_i), TM(\langle S(A_j), \text{axis}_2 \rangle, \text{axis}_1)\} & \text{in case 2;} \\ E(A_k) - \text{Min}\{S(A_i), S(A_j)\} & \text{in case 3.} \end{cases}$$

$$D((A_i \wedge A_j) \cdot A_k) = \begin{cases} \text{Max}\{D(A_i) + D(F_{ik}), D(A_j) + D(F_{jk})\} + D(A_k) - TD & \text{in case 1;} \\ \text{Max}\{D(A_i) + D(F_{ik}), D(A_j) + D(F_{jk}) - TD + D(A_k)\} & \text{in case 2;} \\ \text{Max}\{D(A_i) + D(F_{ik}), D(A_j) + D(F_{jk})\} + D(A_k) & \text{in case 3.} \end{cases}$$

$$d((A_i \wedge A_j) \cdot A_k) = \begin{cases} \text{Max}\{d(A_i) + d(F_{ik}), d(A_j) + d(F_{jk})\} + d(A_k) - TD & \text{in case 1;} \\ \text{Max}\{d(A_i) + d(F_{ik}), d(A_j) + d(F_{jk}) - TD + d(A_k)\} & \text{in case 2;} \\ \text{Max}\{d(A_i) + d(F_{ik}), d(A_j) + d(F_{jk})\} + d(A_k) & \text{in case 3.} \end{cases}$$

Or-join connection is defined as two or more activities workflow branches re-converge into a single thread of control without any synchronisation (WfMC, URL). It is an execution dependence structure that an activity A_k executes after one of two or more activities execution $A_{i1} \vee A_{i2} \vee \dots \vee A_{in}$, denoted as $(A_{i1} \vee A_{i2} \vee \dots \vee A_{in}) \cdot A_k$. We herein just discuss the case that an activity A_k or-join two activities A_i and A_j , which has the three cases like the and-join connection shown in Fig. 3. The run-time duration and build-time duration (maximum and minimum) of the 'or-join' execution of $(A_i \vee A_j) \cdot A_k$ is defined as follows:

$$D_R((A_i \vee A_j) \cdot A_k) = \begin{cases} TM(\langle E(A_k), \text{axis}_2 \rangle, \text{axis}_1) - S(A_i) \text{ or } TM(\langle E(A_k), \text{axis}_2 \rangle, \text{axis}_1) - S(A_j) & \text{in case 1;} \\ E(A_k) - S(A_i) \text{ or } E(A_k) - TM(\langle S(A_j), \text{axis}_2 \rangle, \text{axis}_1) & \text{in case 2;} \\ E(A_k) - S(A_i) \text{ or } E(A_k) - S(A_j) & \text{in case 3.} \end{cases}$$

$$D((A_i \vee A_j) \cdot A_k) = \begin{cases} \text{Max}\{D(A_i) + D(F_{ik}) + D(A_k) - TD + \delta, D(A_j) + D(F_{jk}) + D(A_k) - TD + \delta\} & \text{in case 1;} \\ \text{Max}\{D(A_i) + D(F_{ik}) + D(A_k) + \delta, D(A_j) + D(F_{jk}) + D(A_k) - TD + \delta\} & \text{in case 2;} \\ \text{Max}\{D(A_i) + D(F_{ik}) + D(A_k) + \delta, D(A_j) + D(F_{jk}) + D(A_k) + \delta\}, & \text{in case 3.} \end{cases}$$

$$d((A_i \vee A_j) \cdot A_k) = \begin{cases} \text{Max}\{d(A_i) + d(F_{ik}) + d(A_k) - TD, d(A_j) + d(F_{jk}) + d(A_k) - TD\} & \text{in case 1;} \\ \text{Max}\{d(A_i) + d(F_{ik}) + d(A_k), d(A_j) + d(F_{jk}) + d(A_k) - TD\} & \text{in case 2;} \\ \text{Max}\{d(A_i) + d(F_{ik}) + d(A_k), d(A_j) + d(F_{jk}) + d(A_k)\}, & \text{in case 3.} \end{cases}$$

Duration of any workflow component composed by the above basic workflow components can be similarly computed. For example, $(A_i \cdot A_j) \parallel (A_m \cdot A_n)$ represents the parallel connection of two sequentially connected components $A_i \cdot A_j$ and $A_m \cdot A_n$. We can first compute the durations of $A_i \cdot A_j$ and $A_m \cdot A_n$ by using the duration computing approach for sequential connection then compute the duration of $(A_i \cdot A_j) \parallel (A_m \cdot A_n)$ by using the duration computing approach for parallel connection.

4. Timed workflow process and build-time checking

A workflow process instance is called the longest instance (denoted as Wf_L) if and only if for any workflow process instance Wf , $d(Wf) \leq d(Wf_L)$ hold. The workflow process instance is called the shortest instance if and only if for any workflow process instance Wf , $d(Wf_s) \leq d(Wf)$ hold. $D(Wf_L)$ and $d(Wf_s)$ can be used for the build-time consistency checking according to a given duration low-bound and up-bound.

Definition 2. The workflow process instance Wf is called the build-time consistent with the process duration constraint, if and only if low-bound $\leq d(Wf_s)$ and $D(Wf_L) \leq$ up-bound.

The build-time duration of a timed workflow process is determined by the duration of the basic workflow components that compose the workflow process and the temporal order consistency as stated in Definition 1. The maximum duration and the minimum duration of a workflow process (instance) can be computed by a workflow duration algorithm $WfDA(\text{Input}, \text{Output})$. The input is the workflow process definition with the build-time duration of every activity, the build-time duration of every flow, and the time zones where the activities execute. The output is the build-time minimum and maximum duration of both the shortest and the longest instance of the workflow. $WfDA$ can be designed by incorporating the build-time duration definition of the basic timed workflow components into the minimum duration and the maximum duration computation formulation in (Evans and Edward, 1992; Marjanovic and Orłowska, 1999). To avoid repetition, we do not discuss here in detail.

With the duration definition of the basic workflow components and $WfDA(\text{Input}, \text{Output})$, we can answer with the following questions about a workflow process:

1. Given any two activities A_i and A_j of a workflow process, we can estimate the maximum and the minimum build-time duration between them, i.e., computing $D(A_i, A_j)$ and $d(A_i, A_j)$;
2. Compute the build-time duration for any instance of the workflow process;
3. Given a low-bound and an up-bound of the build-time duration between any two activities A_i and A_j of the workflow process, check the build-time duration consistency according to the constraints.

The run-time duration of a workflow process is the period from the start time of the initial activity A_0 to the end time of an activity A_n is defined as follows:

$$D_R(A_0, A_n) = \begin{cases} E(A_n) - S(A_0) & \text{if } A_0 \text{ and } A_n \text{ are on the same axis;} \\ TM(\langle E(A_n), \text{axis}_2 \rangle, \text{axis}_1) - S(A_0) & \text{if } A_0 \text{ and } A_n \text{ are on different axes.} \end{cases}$$

5. Dynamic run-time consistent checking

Run-time consistent checking is to check whether the workflow execution satisfies a set of pre-determined constraints. The build-time duration consistent checking cannot meet the need of the run-time consistent checking. One reason is that the run-time duration of the executed part is a constant value, and the other is that exceptions may happen in run-time. For example, the maximum duration of a flow can be exceeded when an information flow transmission broken is happened. So checkpoints and the related constraints need to be set, usually at the following places: the start time of an activity; the end time of an activity; the start time of a flow; and the end time of a flow. During execution, the activity handler ($WfMC$, URL) is responsible for reporting the exact time at the checkpoints related to the activity. The checkpoints on the other time axis should be mapped onto the reference time axis.

Let C_0 and C_i be two checkpoints and the duration constraint of the workflow execution be $D_R(C_0, C_i) \leq p_i$. The consistency corresponding to the constraint includes the following two cases:

Case 1: $0 \leq i < n$. If the constraint is satisfied at the checkpoint C_i , then the workflow is called stage consistent with the constraint p_i at the checkpoint, otherwise is called stage inconsistent with the constraint at the checkpoint.

Case 2: $i = n$, the constraint becomes $D_R(C_0, C_n) \leq p_n$. If the constraint is satisfied at the checkpoint C_n , then the workflow is called global temporal consistent with the constraint p_n , otherwise is global inconsistent with the constraint.

Since the execution duration between any two checkpoints C_i and C_j ($0 \leq i < j \leq n$) can be computed by $D_R(C_i, C_j) = D_R(C_0, C_j) - D_R(C_0, C_i)$, the duration constraint between two checkpoints C_i and C_j can be represented as $D_R(C_i, C_j) \leq p_j - p_i$, which satisfies $D_R(C_0, C_j) \leq p_j$ and $D_R(C_0, C_i) \leq p_i$. If the constraint can be satisfied, the workflow is called duration consistent with the constraint $p_j - p_i$ between the two checkpoints C_i and C_j , otherwise is duration inconsistent with the constraint.

We cannot rigidly conclude that a timed workflow execution is global inconsistent just in terms of the stage inconsistency. Because the workflow can be still globally consistent if the execution of the remaining part of the workflow can be tightened. A run-time workflow consists of two dynamic durations: (1) the exact execution duration from the initial activity to the current checkpoint; and (2) the estimated build-time duration from the current checkpoint to the last checkpoint. The combination of the two durations can dynamically reflect the relationship between an executing workflow process and the related constraints.

Let $d(C_i, C_n)$ be the build-time minimum duration between C_i and C_n . The following two cases need to be considered when the stage inconsistency happens at checkpoint C_i :

1. If $D_R(C_0, C_i) > p_i$, and $D_R(C_0, C_i) + d(C_i, C_n) \leq p_n$, the stage inconsistency happens at checkpoint C_i . But it can still be global consistent if the remaining part can be executed under an adjusted constraint. For example, the constraint p_j at checkpoint C_j ($i < j < n$) can be adjusted by $p'_j = p_j + (p_i - D_R(C_0, C_i))/(n - j)$.
2. If $D_R(C_0, C_i) > p_i$, and $D_R(C_0, C_i) + d(C_i, C_n) > p_n$, the global inconsistency is detected. In this case, the remaining part of the workflow needs to be re-arranged so as to shorten its execution duration. The re-arrangement is constrained by the duration constraint, the process definition, and the resources constraint (if any).

The main part of the consistent checking process is described as follows:

```

INPUT: Sequential pairs of the checkpoint and constraint  $\langle\langle C_0, p_0 \rangle, \dots, \langle C_i, p_i \rangle, \dots, \langle C_n, p_n \rangle\rangle$ ;
OUTPUT: Consistent or inconsistent report;
i = 0 ;           // i is initiated as the first checkpoint
DO WHILE the current checkpoint i ≤ n;
{IF  $D_R(C_0, C_i) \leq p_i$ 
  THEN {Output ‘The workflow is consistent with the constraint at checkpoint  $C'_i$ ’}
  ELSE {
    IF  $D_R(C_0, C_i) > p_i$  and  $D_R(C_0, C_i) + d(C_i, C_n) \leq p_n$ 
      THEN {Output ‘Stage inconsistent with the constraint at the check point  $C'_i$ ’;
        Adjust constraints of the remaining checkpoints as:
           $p'_j = p_j + (p_i - D_R(C_0, C_i))/(n - j)$  for  $j = i + 1, \dots, n$ }
      IF  $D_R(C_0, C_i) > p_i$  and  $D_R(C_0, C_i) + d(C_i, C_n) > p_n$ 
        THEN {// the global inconsistency is detected at checkpoint i
          IF the next activity is not the final activity;
            THEN Call Re-arrangement-process; //re-arrange the remaining part of process
          ELSE Output ‘Global inconsistency’ and Exit with fail }
        IF Re-arrangement is not success,
          THEN Output ‘Global inconsistent’ and Exit with fail }
      i = i + 1;  $C_i$  = next-checkpoint;  $p_i$  = next-constraint;
    }
  Output ‘Global consistent’;
End;
```

6. Process planning

6.1. Time planning

Process planning is to arrange the execution of n activities of a workflow onto m axes (time zones) under the pre-determined duration constraints, dependence constraints, and resource constraints (if any). Time planning usually concerns two issues: (1) arrange n activities onto proper m' ($m' \leq m$) time zones such that the whole workflow duration

is minimised; and (2) arrange n activities onto certain m axes such that the whole workflow duration can be finished within a given time constraint. Since the execution duration of an activity is determined by the internal execution mechanism of the activity, time planning should focus on three variable factors: (1) minimising the flow durations; (2) enabling more activities to execute in parallel under resource constraint (if any); and (3) making full use of time difference among different axes so as to minimise the duration of the whole workflow.

Time planning includes two stages: (1) build-time planning, i.e., to design a workflow process satisfying the temporal consistency; and (2) dynamic run-time re-arrangement, i.e., to re-arrange the remaining part of the workflow process when temporal inconsistency is detected. A re-arrangement mechanism needs to estimate the minimum duration that can be used in run-time before carrying out the re-arrangement and to check the exact duration after the re-arrangement. The re-arrangement duration needs to be taken into account in consistent checking of the next checkpoint. The following three heuristic rules are for time planning.

Rule 1. Arrange flows to take the non-working durations as possible under the dependence order constraints of the workflow process.

Rule 2. If an activity cannot be executed within the remaining part of the working duration of its predecessor (A_i), arrange the activity to execute on the working duration of the nearest axis which corresponds to the non-working duration of axis (i) as possible under resource constraints (if any).

Rule 3. Arrange activities to execute in parallel at the same axis as possible under the dependence order constraint and the resource constraint (if any).

6.2. Build-time process definition

The build-time process definition of a timed workflow process consists of the following steps:

1. Design the workflow process at the logical level, i.e., define the traditional workflow process.
2. Determine the constraints then check the reasonability of these constraints.
3. Determine the time axes and the time differences among these axes according to the business requirement, then determine the reference axis.
4. Divide each axis into the working durations and non-working durations according to the working scheme of the sites the activities execute.
5. Assign activities onto proper working durations of proper axes in terms of the execution duration constraints; the dependence order consistency constraint; and resource constraint (if any).
6. Estimate the minimum and the maximum build-time duration for every activity and every flow.
7. Define the flows among activities in terms of the time difference, and define different types of dependence relationship among activities, e.g., execution dependence, information dependence, and material dependence.
8. Compute the minimum and the maximum build-time durations for the workflow process in terms of the build-time durations of activities and flows.
9. Check the build-time consistency of the workflow.

7. Example: timed workflow process for distributed team development

Purpose. To demonstrate the use of the timed workflow process model in distributed team development application.

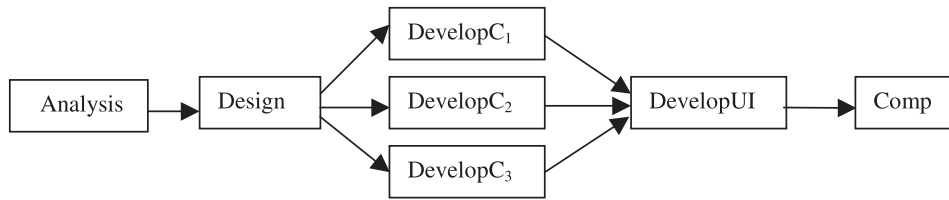
Assumption. (1) the application system consists of four components C_1 , C_2 , C_3 , and an user interface UI; (2) the duration constraint of system development is three working days (assume a working day has eight working hours); (3) the development of each component requires one developer; (4) the total resource constraint is five developers; (5) after the development of a component, the developer resource will be released and can be reused to develop the succeeding component on the same axis; and (6) the build-time duration of these activities are measured by the working hours as follows:

$d(\text{Analysis}) = 6$, $D(\text{Analysis}) = 8$; $d(\text{Design}) = 7$, $D(\text{Design}) = 8$; $d(\text{Develop}C_i) = 6$, $D(\text{Develop}C_i) = 8$; $d(\text{DevelopUI}) = 5$, $D(\text{DevelopUI}) = 8$; and $d(\text{Comp}) = 1$, $D(\text{Comp}) = 3$.

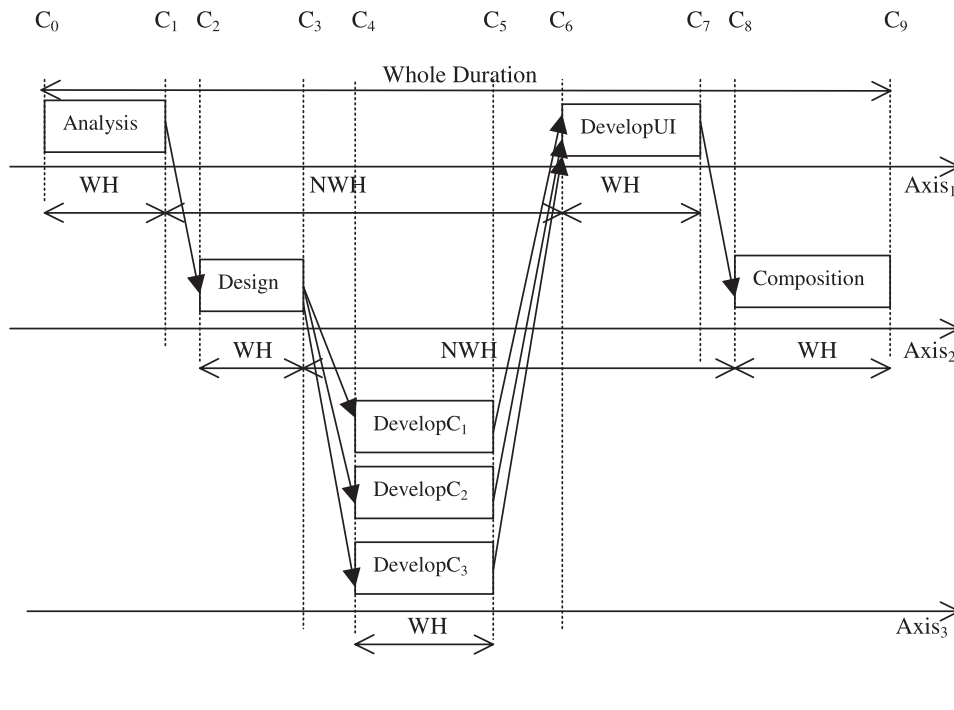
The logical level workflow process of the development process is shown in the upper level of Fig. 4, where the initial activity is ‘analysis’ and the end activity is ‘composition’ (i.e., Comp). If we do not consider the flow duration and the time difference, the build-time duration of the workflow will be computed as follows:

$$d(\text{WF}) = d(\text{Analysis}) + d(\text{Design}) + \text{Max}\{d(\text{Develop}C_1), d(\text{Develop}C_2), d(\text{Develop}C_3)\} + d(\text{DevelopUI}) + d(\text{Comp}) = 6 + 7 + 6 + 5 + 1 = 25,$$

The first level workflow process



The second level workflow process



The third level: WfMS

Fig. 4. Timed workflow process for distributed team development.

$$D(WF) = D(\text{Analysis}) + D(\text{Design}) + \text{Max}\{D(\text{DevelopC}_1), D(\text{DevelopC}_2), D(\text{DevelopC}_3)\} + D(\text{DevelopUI}) + D(\text{Comp}) = 8 + 8 + 8 + 8 + 3 = 35.$$

The result shows that the build-time duration does not satisfy the required duration constraint: 24 working hours. On resource constraint aspect, the resources required by each step do not exceed three developers (a developer can be reused for developing the succeeding component after finishing the current activity, and the parallel development of C_1, C_2 and C_3 just need three developers), so the total resource constraint can be satisfied.

With the timed workflow process model, activities can be arranged onto several time axes. The whole duration of traditional workflow process can be shortened if we make full use of time difference among these time axes. The lower part of Fig. 4 shows an example of arranging the activities onto three time axes, where we assume $TD(\text{axis}_1, \text{axis}_2) = TD(\text{axis}_2, \text{axis}_3) = 8$ h, and WH and NWH denote the working hours and the non-working hours, respectively. According to rule 2 presented in Section 6.1, the activity ‘analysis’ is arranged onto the working hours from 8:00 to 18:00 on axis₁. The activity ‘design’ is arranged onto the working hours of axis₂, which starts after the end

of the working hours on axis₁. According to rule 3, activities ‘DevelopC₁’, ‘DevelopC₂’, and ‘DevelopC₃’ are arranged onto axis₃ in parallel, such that they can be finished before the working hours for performing the activity ‘DevelopUI’ on axis₁. The activity ‘Composition’ is arranged onto the second working hours on axis₂. According to rule 1, we arrange all the flows to take just the non-working hours (assume all the estimation of build-time flow duration are the same, e.g., $d(F) = 1$ min and $D(F) = 30$ min), so the flow duration does not need to take into account when computing the whole duration by working hours in this example. We choose axis₁ as the reference time axis, then the whole build-time duration (working hours) can be computed as follows:

$$d(\text{WF}) = d(\text{Analysis}) + d(\text{DevelopUI}) + d(\text{Composition}) = 6 + 5 + 0 = 11,$$

$$D(\text{WF}) = D(\text{Analysis}) + D(\text{DevelopUI}) + D(\text{Composition}) = 8 + 8 + 0 = 16.$$

The result shows that the duration is consistent with the required constraint: 24 working hours. Since developers are distributed onto three time axes, a developer (resource) can be reused to develop several different components sequentially only on the same axis. The activities on axis₁ and the activities on axis₂ require one developer, respectively. Three developers are required to implement the three parallel activities on axis₃. So the execution of the workflow totally requires five developers. The resource constraint can be satisfied.

For run-time checking, checkpoints C_i ($i = 1, \dots, 9$) are set as shown in Fig. 4. We herein examine the following check points: C_1 , C_5 , C_6 , C_7 and C_9 . Constraint C_i corresponds to a constraint p_i as follows: $p_1: D_R(C_0, C_1) \leq 8$; $p_5: D_R(C_0, C_5) \leq 8$; $p_6: D_R(C_0, C_6) \leq 8$; $p_7: D_R(C_0, C_7) \leq 16$; and $p_9: D_R(C_0, C_9) \leq 24$. Assume the workflow is executing at checkpoint C_5 . We examine the relationship between the stage inconsistency and the global inconsistency in the following three cases:

Case 1. Two hours are delayed at checkpoint C_5 , i.e., $D_R(C_0, C_5) = p_5 + 2 = 10$ (work hours). In this case, stage inconsistent is detected at checkpoint C_5 . Since $d(\text{DevelopUI}) + 2 \leq D(\text{DevelopUI})$, the workflow could still be consistent with the constraint at C_6 .

Case 2. Four hours are delayed at checkpoint C_5 , i.e., $D_R(C_0, C_5) = p_5 + 4 = 12$. In this case we have $D_R(C_0, C_5) > p_5$, $d(C_5, C_9) + (p_5 - D_R(C_0, C_5)) > p_9 - p_5$, and the development of UI cannot be carried out during the working hours on axis₃. In order to keep the global consistency, we need to re-arrange the rest part of the workflow according to the time planning rules. A re-arrangement of the remaining part of the workflow is shown in Fig. 5. Since the activity ‘Composition’ executes during the non-working hours on axis₁, $D(\text{Composition}) = 0$ on axis₁. Hence we have

$$D(\text{WF}) = D(C_0, C_5) + D(\text{DevelopC}_3) + D(\text{Composition}) = 12 + 8 + 0 = 20,$$

and

$$d(\text{WF}) = d(C_0, C_5) + d(\text{DevelopC}_3) + d(\text{Composition}) = 12 + 6 + 0 = 18.$$

After the re-arrangement, the workflow can still keep the global consistency with the constraint.

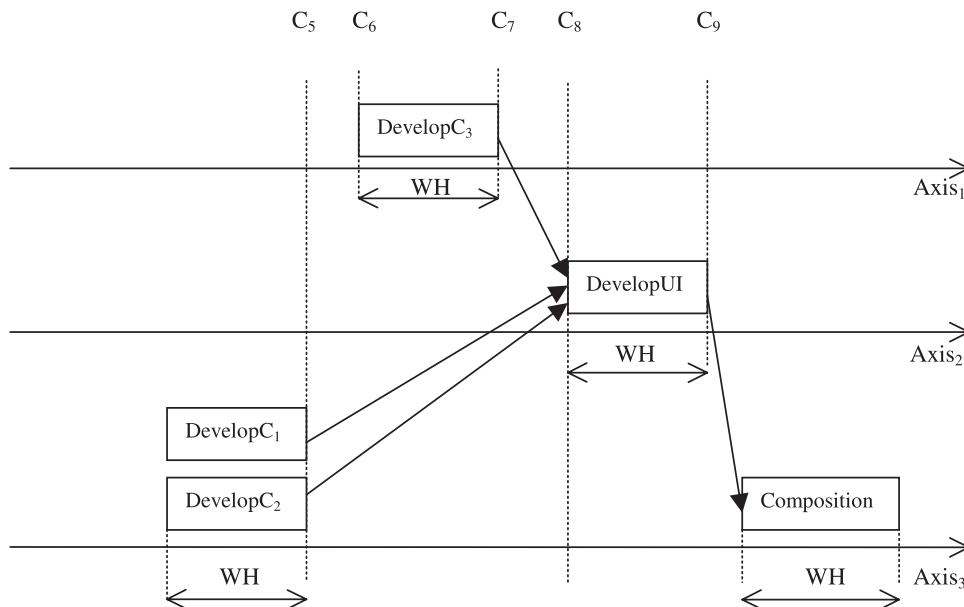


Fig. 5. Re-arrangement of the rest part of the workflow of Fig. 4.

Case 3. Nine hours are delayed at checkpoint C_5 , i.e., $D_R(C_0, C_5) = p_5 + 9$. In this case we have $D_R(C_0, C_5) > p_5$, $d(C_5, C_9) + (p_5 - D_R(C_0, C_5)) > p_9 - p_5$, and the remaining part cannot be re-arranged within the rest duration. Hence, the global inconsistency happens in this case.

8. Comparison and discussion

Comparing with the previous approaches (especially, Eder et al., 1999; Marjanovic and Orłowska, 1999), the major differences consist of three aspects. First, flow duration is taken into account in the proposed approach, while it is neglected in the other approaches. This extension is significant because it enables the proposed approach to exactly measure the duration of a geographically distributed workflow process. This extension will cause some semantic adaptation of traditional workflow model. Second, time difference factor is taken into account in the proposed approach, while it is neglected in the other approaches. This extension can avoid time inconsistency when activities are globally distributed, and can be used for reducing the duration of globally distributed workflow processes. This advantage is explained through an example of distributed team development. Third, the proposed consistency checking incorporates the exact run-time duration and the estimated build-time duration (the maximum and minimum durations). This enables the checking approach to judge the global consistency based on the stage consistency checking. The result of the stage inconsistency and the global inconsistency checking provide the process monitor the evidence for making the decision of whether to continue the process or not when stage inconsistency happens. We also provide the approach for adapting a workflow process to be consistent when a global inconsistency has been detected. Besides, the proposed approach includes the super-constraints on time constraints and the process planning approach. The previous approaches do not have these functions.

The proposed approach can be regarded as a refinement of the traditional workflow process. A workflow system development can work at three levels. The first level (top) is the logical workflow process, where the execution dependence relationships among activities are defined. The second level is the conceptual workflow process definition by using the proposed timed workflow model. It is a refinement of the first level. At the second level we need to consider the site-related activity distribution, time constraints, resource constraint (if any), and process planning. The third level is the WfMS, which is responsible for executing and managing the conceptual level workflow process, and inter-operating with the applications and the human users. To support the execution of the second level workflow process, traditional WfMS needs to be slightly extended. First, we need to extend the data fields for recording the time factors and the constraints about the activities and flows to the data structure interface between the build-time and the run-time of WfMS. The time-related fields include the minimum duration, the maximum duration, the start time, and the end time. Second, we need to add the build-time consistency checking approach to the workflow definition mechanism, and to add the run-time consistency checking approach to the workflow enactment mechanism.

9. Summary

This paper presents a timed workflow process model to meet the needs of the globalization of business applications. The model consists of the time-related definitions about the activity and flow, the build-time duration and the run-time duration of a set of basic workflow components, and the approach for temporal consistent checking and process planning. An example demonstrates the proposed approach.

The main contribution of this work concerns three aspects. First, we incorporate the flow duration into the duration computation of workflow process in both the build-time and the run-time. This enables the proposed approach to exactly measure the duration of business process in both the build-time and the run-time, especially in globally distributed business applications. Second, we incorporate multiple time axes into the workflow process, where activities can be distributed at different sites belonging to different time zones. This extension enables the proposed approach to be a planning tool for shortening the execution duration of global workflow applications. Third, we proposed a consistent checking approach based on the combination of the exact run-time duration and the estimated build-time duration. Previous approaches do not have the above three functions. The proposed model is useful in modeling the globally distributed applications, where activities are geographically distributed and flow transmissions are time consuming.

The ongoing work is to incorporate the resource sharing and cost evaluation of both the activities and flows in a workflow process and to apply the proposed model to real e-commerce problems, e.g., distributed inter-organization supply chain management.

Acknowledgements

Research was supported by the Government of Hong Kong under Grant CERG 1072/00E and partially supported by the National Science Foundation of China and Singapore NSTB Grant RP960349.

References

- Baekgaard, L., Godskenen, J.C., 1998. Real-time event control in active databases. *J. Syst. Software* 42, 263–271.
- Bestougeff, H., Ligozat, G., 1992. *Logical Tools for Temporal Knowledge Representation*. Ellis Horwood, England.
- Botti, V., Barber, F., Crespo, A., 1998. Towards a temporal coherence management in real-time knowledge-based systems. *Data Knowledge Eng.* 25, 247–266.
- Casati, F., Fugini, M., Mirbel, I., 1999. An environment for designing exceptions in workflows. *Inform. Syst.* 24, 255–273.
- Dechter, R., Meiri, I., Pearl, J., 1991. Temporal constraint networks. *Arti. Intell.* 49, 61–95.
- Evans, J., Edward, M., 1992. *Optimisation Algorithms for Networks and Graphs*. Marcel Dekker, New York.
- Eder, J., Panagos, E., Pozewaunig, H., Rabinovich, M., 1999. In: *Time Constraints in Workflow Systems, CAiSE'99*. Heidelberg, Germany, June 1999.
- Georgakopoulos, D., Hornick, M., 1995. An overview of workflow management: from process modeling to workflow automation infrastructure. *Distrib. Parallel Databases* 3, 119–153.
- Geppert, A., Tombros, D., Dittrich, K.R., 1998. Defining the semantics of reactive components in event-driven workflow execution with event histories. *Inform. Syst.* 23, 235–252.
- Jensen, C.S., Snodgrass, R.A., 1999. Temporal data management. *IEEE Trans. Knowledge Data Eng.* 11, 36–44.
- Lawrence, P. (Ed.), 1997. *Workflow Handbook 1997*. Wiley, New York.
- Lee, J.Y., Elmasri, R., Won, J., 1998. An integrated temporal data model incorporating time series concept. *Data Knowledge Eng.* 24, 257–276.
- Leymann, F., Roller, D., 1997. Workflow-based applications. *IBM Syst. J.* 36, 102–122.
- Liu, Z., 1998. Performance analysis of stochastic timed petri nets using linear programming approach. *IEEE Trans. Software Eng.* 24, 1014–1030.
- Marjanovic, O., Orlowska, M.E., 1999. On modeling and verification of temporal constraints in production workflows. *Knowledge Inform. Syst.* 1, 157–192.
- Puustjarvi, J., 1997. Reusability and modularity in transactional workflows. *Inform. Syst.* 22, 101–120.
- Rose, A., 1998. Visual assessment of engineering processes in virtual enterprises. *Commun. ACM* 41, 45–52.
- Tsai, J.J.P., Yang, S.J., Chang, Y.H., 1995. Aiming constraint petri nets and their application to schedulability analysis of real-time system specification. *IEEE Trans. Software Eng.* 21, 32–48.
- WfMC, The Workflow Reference Model, Document Number AC00-1003, <http://www.wfmc.org/>.
- Zhuce, H., Pung, H.K., Cheung, T.Y., 2000. Timed workflow model: concept, model, and method. In: *First International Conference on Web Information Systems Engineering*, Hong Kong, June 2000, pp. 166–172.

Hai Zhuge received Ph.D. in computer science from Zhejiang University, China. He was the postdoctoral research fellow, the associate research professor, and the senior fellow at the Institute of Software, Chinese Academy of Sciences. He has been the guest professor of an institute in China, and the principal investigators of three national grants. He has been a visiting research fellow of the National University of Singapore and the City University of Hong Kong several times in the past five years. His current research interests include: multi-agent co-operation systems, problem-oriented component repository, cognitive-based software process model, workflow model and application, and inter-operation model for group decision. His publications mainly appear in: *IEEE Transactions on Systems, Man, and Cybernetics*; *Decision Support Systems*; *Journal of Systems and Software*; *Knowledge-based Systems*; *Information and Software Technology*; and *Chinese Journal of Advanced Software Research*.

To-Yat Cheung got his Ph.D from the University of Wisconsin. He has had almost 30 years of experience in industry, governments and academics in North America and Hong Kong. He is currently the Chair Professor of the Department of Computer Science, City University of Hong Kong, and honorary professor of several other universities. His current research interests are in verification and testing in component-based software engineering, application of Petri nets, object-oriented software design, workflow systems, distributed computing and Internet applications.

Hung-Keng Pung is an associate professor of the Department of Computer Science, National University of Singapore, Singapore.