Free Schedules for Free Agents in Workflow Systems

Areas: time in multiple agents, communication, and synchronization; temporal constraint reasoning

Claudio Bettini[*] X. Sean Wang[†] Sushil Jajodia[†]

Abstract

This paper investigates workflow systems in which the enactment and completion of activities have to satisfy a set of quantitative temporal constraints. Different activities are usually performed by autonomous agents, and the scheduling of activities by the enactment service has among its goals the minimization of communication and synchronization among the agents. The paper formally defines the notion of a schedule for these workflow systems and it identifies a particularly useful class: *free schedules*. A schedule specifies a time range for the enactment, duration, and completion of each activity in order to satisfy all the temporal constraints in the workflow. In a free schedule, an agent has to start its activity within the range specified in the schedule, but it is free to use any amount of time to finish the activity as long as it is between a minimum and maximum time he has declared when the workflow is designed. No synchronization with other agents is needed. The paper provides a method to characterize all the free-schedules admitted by a workflow specification, and an algorithm to derive them.

1 Introduction

A workflow is a complete or partial automation of a business process, in which participants (humans or machines) involve in a set of activities according to certain procedural rules and constraints. The successful completion of the process often depends on the correct synchronization and scheduling of the activities. The modeling and reasoning tools provided by this paper are intended to address so called *production workflows* with particular emphasis on processes involving loosely coupled, largely distributed information sources where the agents cooperating in a workflow process are essentially independent from each other.

We consider the inclusion in the workflow specification of quantitative temporal constraints on the duration and distances of individual activities. As a simple example, consider an online vendor workflow, including the following activities that must be performed upon the receipt of an order by a customer: (a) order processing, (b) shipping, and (c) payment collection. These activities have certain conditions concerning their timing that may impose temporal distances (possibly involving different time granularities). For instance, the order processing must occur within one *business day* after the order is entered (and the whole workflow process is enacted), the order must be transmitted to the shipping sites within ten *hours* after the end of order processing, and the payment for the merchandise must be made within a time window starting and ending one month before and after delivery, respectively. The payment collection activity has a duration range (i.e., minimum and maximum time) specified in terms of *business days*, e.g., the activity can take as little as one business day and as much as 5 business days. (These requirements are included in the graphical representation of Figure 2 later in the paper.)

The inclusion of temporal constraints naturally leads to questions about how to check the overall consistency of the specification and about how to apply some form of useful temporal reasoning; for example, how can we predict when a certain agent may be asked to perform an activity? However, these questions can be addressed quite easily applying known techniques in constraint reasoning. In this paper, we concentrate on a different issue, related to the enactment service of the workflow system, which has to schedule the different activities in order to guarantee a successful completion of the workflow. This must be done considering all the constraints, and the fact that each activity is per-

[*]DSI, Università di Milano, Italy. bettini@dsi.unimi.it

[†]Dept. of Info.& Software Systems Eng., George Mason University, VA. {xywang, jajodia}@gmu.edu

formed by a relatively autonomous agent, which, in general, may take a different amount of time to complete the same activity.

We assume that each agent in the workflow system declares a time range (the minimum and the maximum amount of time) it usually needs to finish a particular activity. It is then desirable to allow the agents to take any amount of time within the declared time range to finish their work. However, the time each agent actually takes may have some impact on the overall temporal constraints, because there may be constraints relating the ending times of activities. The question is whether there exists a schedule for the activities' enactment such that, no matter when each agent finishes the activity within the declared time range (i.e., using any amount of time between the declared minimum and maximum), the overall temporal constraints are not violated. We call this a *free schedule*.

The main contribution of this paper is a formal characterization of free schedules and an algorithmic method to derive them from the workflow specification .

Little attention has been given in the literature to modeling advanced temporal features for workflow systems. Commercial workflow systems (as reviewed, e.g., in [ADEM97]) are usually limited in specification of temporal conditions for each individual activity or for the global plan and do not provide temporal reasoning. There are two recent papers on related issues. The authors of [MO99] propose a framework for time modeling in production workflows. They also provide an efficient algorithm to check if a constraint (like a deadline or inter-activity constraint) is "implied" by the given activity duration constraints and by the workflow structure. This is similar to checking in our framework that a free schedule retains its "free" property upon the addition of certain constraints. However, it does not give a method to derive a free schedule starting with a global set of intra- and inter-activity constraints. The second paper, [EPR99], is concerned with temporal constraints reasoning and management in the context of workflows. Their temporal constraints form a subclass of those considered in this paper, due to the "well structured" requirement in [EPR99]. Their reasoning about the allowed "buffer time" for parallel activities, is similar to deriving free schedules, even if using completely different methods. Regarding the general scheduling problem, a rich literature exists on the subject (see [SC93] for a good survey). However, our method exploits the particular structure of the constraints, and, for this reason, it has much better computational properties than general purpose scheduling algorithms.

The paper is organized as follows: In the next section, we define the workflow model, and in Section 3, we formally characterize free-schedules and provide methods to derive them. We conclude the paper in Section 4.

2 The workflow model

A typical workflow management system provides a formalism and tools to specify workflow activities. We follow the consensus workflow glossary [WfMC99] for the choice of operators to specify the workflow structure. The relation between activities can be specified by sequential routing, as well as through the use of the operators OR-split, AND-split, OR-join, and AND-join. An OR-split identifies a point within the workflow where a single thread of control makes a decision upon which branch to take when encountered with multiple alternative workflow branches, while an AND-split is a point where a single thread of control splits into two or more parallel activities. OR-joins and AND-joins identify points where various branches re-converge to a single path. Note that an OR-join, as opposed to the AND-join, does not require synchronization, since the ancestors activities are not executed in parallel but they are alternatives. In this paper, we do not consider loop operators. Our techniques can still be applied to workflow processes involving loops when loops can be modeled as complex activities, estimating time bounds for their execution by the agents in charge.

A workflow graphical description is shown in Figure 1. An arrow between two activities denotes sequential routing, while arrows going through a circle denote OR-split operators (C in the circle denotes the associated condition). AND-splits are implicit when more than one arrow originates from the same activity. All joins are implicit when more than one arrow lead to the same activity, where an OR-join is intended when the arrows originate from alternative activities, and an AND-join when from parallel activities. The dashed arrows mean that details on other activities in that part of the workflow are omitted.

Our temporal extension allows a workflow designer to include two types of temporal constraints:

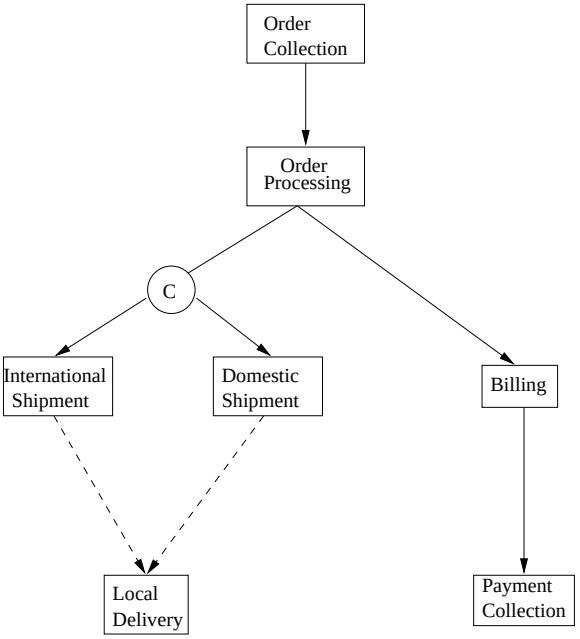

Figure 1: A workflow process description

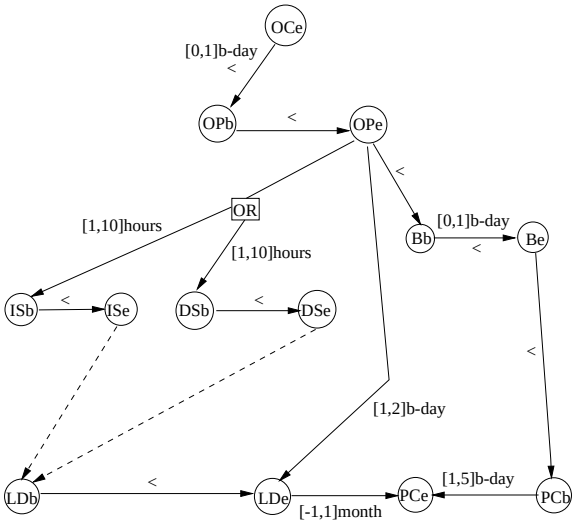

Figure 2: Temporal constraints in a workflow

(a) Constraints in each activity's description, as duration constraints and deadline constraints; (b) Constraints in the workflow process description, as quantitative constraints on the starting/ending of different activities.

We represent these constraints as (binary) distance constraints $X_j - X_i \in [m, n]G$ where X_j and X_i are variables representing the instant starting or ending an activity, m and n are either integers or one of the special symbols $-\infty$ and ∞, respectively, and G denotes the time granularity in terms of which the distance is measured. These constraints, known as TCG (Temporal Constraints with Granularity) have been formally studied in [BWJ98, BWJ97]. When G is omitted, the distance is intended in the units of the time domain. A particular case is that of unary constraints, as $X_j \in [m, n]G$, that can force the domain of variables to take values in a specific subset of the time domain.[1] We use $Dom(X_j)$ to denote the domain of X_j.

The constraints informally introduced in Section 1 about our running example, are illustrated using a graph in Figure 2. Each node in the graph is labeled by the initials of the activity's name followed by 'b' for begin or 'e' for end. For example, `OC` stands for the {O}rder {C}ollection activity, `OP` for {O}rder {P}rocessing, etc. The symbol '<' is used as a shortcut of $[1, +\infty]$. More than one constraint can be associated with an arc, with conjunction of them being the semantics. For example, from Bb to Be, $[0, 1]$ `b-day` forces the end of the Billing activity to occur in the same or next business-day as the beginning, while '<' forces the duration of the activity to be positive, since this is not enforced by the first constraint.

Each workflow constraint graph can be easily decomposed according to the OR-split operators in a set of subgraphs each one representing one possible workflow execution thread. In our running example, the graph in Figure 2 is decomposed into a subgraph taking the left branch of the OR-split (i.e., performing ISb and ISe), and another one taking the right branch (i.e., performing DSb and DSe). Note that each subgraph defines a constraint problem commonly known as STP (Simple Temporal Problem). When all constraints are in terms of the same granularity there are efficient algorithms to check consistency and to derive the minimal network [DMP91]. For uniformity with the CSP literature we call these subgraphs *constraint networks*.

3 The generation of enactment schedules

Every time the enactment agent dispatches an activity, it needs to provide a set of constraints on the

[1] We use positive integers as the time domain.

beginning and ending times to the agent in charge of that activity, such that if each agent satisfies the constraints, the whole workflow can be successfully carried out.

To illustrate, consider a refinement of the Domestic Shipment (DS) activity. Upon completion of the order processing, the online vendor asks directly the suppliers of the requested products to ship them to one of its warehouses located in the area of the customer for their final delivery. Obviously, the workflow requires some sort of synchronization to ensure that all the products will be delivered to the customer within a certain time to reduce the need of warehouse space. In Figure 3 we consider the example of two activities S and S' corresponding to the shipments to be made by two suppliers. Each supplier has provided minimum and maximum bounds for the handling and shipping of its products (constraints [3, 7] and [1, 3] respectively). The two activities must start after order processing and not later than 10 time units from it (constraints [1, 10]). Also, the final delivery must begin after all products are available and none of the products must wait more than 5 time units at the warehouse (constraints [1, 5]).

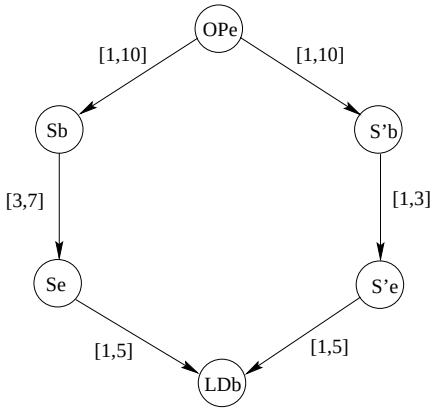

Figure 3: The portion of constraint graph involving the shipping activities S and S'

In this case, the enactment service cannot assign S and S' with the same earliest possible starting times. Indeed, assume we are using granularity *hours* and suppose the order processing completed at 8am, and S and S' are assigned to start at 9am. According to duration constraints, activity S could take 7 hours to complete, while activity S' could take 1 hour only. Hence, the products shipped by S' will have to stand at the warehouse more than the 5 hours allowed by the constraint. To solve this problem, we can delay the starting time for S' to 11am.

3.1 Schedules and Free Schedules

Given the problem description in terms of a set of activities to be enacted, the minimum and maximum durations declared by each agent in charge of an activity, and other temporal constraints involving different activities, we want to provide to the enactment service a *schedule* for these activities. For the sake of simplicity, in the rest of this paper we restrict our technical investigation to the case of graphs where all constraints are or can be converted in terms of a single granularity.

We formally define the notion of a *schedule*.

Definition A *schedule* for a set of activities $A_1, \ldots, A_k$ whose constraints are represented in a network N, is a set of triples of intervals on positive integers $([Ebegin_i, Lbegin_i], [m_i, n_i], [Eend_i, Lend_i])$, one for each activity, such that: if for each activity A_i, a pair (x_i, y_i), with x_i in $[Ebegin_i, Lbegin_i]$, y_i in $[Eend_i, Lend_i]$ and (x_i, y_i) satisfies the constraint $[m_i, n_i]$, is used to instantiate the corresponding pair of beginning and ending nodes for A_i in N, then this assignment can be extended to a solution of N.

The first and last intervals in a schedule identify the allowed beginning and ending instants, respectively (L and E stand for Latest and Earliest, respectively), while the second interval identifies the minimum and maximum duration, called the duration constraint. Hence, if all the involved activities actually begin and end within the given bounds and do not violate the duration constraints, the schedule guarantees that all the constraints in the network are still satisfiable. The motivation for this definition is that the agents responsible for the activities should be allowed to act independently from other activities. Each agent needs only adhere to a "local" constraint and the global constraint should be satisfiable if each agent satisfies the local constraint attached to it.

As an example, assume OPe in Figure 3 happens at time 8. Then the following is a schedule for activities S and S': `⟨([9,9],[3,4],[12,13]),([9,9],[1,2],[10,11])⟩`. Indeed, since S finishes (i.e., Se) either at 12 or at

13, and S' finishes (i.e., $S'e$) either at 10 or 11, in all the possible cases, it's possible to assign LDb a value to satisfy all the constraints in the network.

We say that a schedule is *well-formed* if there are no "redundant" values in each triple. Formally, a triple has no redundant values if for each value in the beginning/ending interval there exists one in the other interval such that the pair satisfies the duration constraint (i.e., the middle interval in the schedules). Moreover, each value in the duration constraint range should be used by one of these pairs. For example, $([9,10],[3,4],[12,13])$ is well-formed since (i) each value in the beginning domain can find a value in the ending domain to satisfy the constraint (e.g., 9 finds 12 and 10 finds 13), (ii) each value in the ending domain can find a value in the beginning domain to satisfy the constraint (e.g., 12 finds 9 and 13 finds 9), and (iii) each duration value (i.e., 3 and 4) can find the beginning and ending pairs to have the exact durations (e.g., 9 and 12 and 9 and 13, respectively).

Note that in the constraint networks and schedules, constraints may involve $+\infty$ or $-\infty$ and domains may be infinite. For simplicity of presentation, we don't explicitly treat ∞ in our exposition. However, it's not difficult to slightly extend the arithmetics to take them into account.

Many different schedules can exist for a given workflow and different criteria can be adopted to identify most interesting ones. When agents for activities operate independently, schedules which do not modify the original duration constraints that these agents declared in the workflow specification, seem to be preferable to those which restrict those duration constraints. We call these schedules *free schedules*. If the restriction is unavoidable, a schedule which restricts the maximum duration, (called *restricted due-time schedule*) is preferable to one forcing a greater minimal duration (called *bounded schedule*). In the following, we formally characterize free schedules, and concentrate on the procedure to generate them. We assume for each activity A_i, the duration constraint from A_ib to A_ie in the network is $[minD_i, maxD_i]$.

Definition A *free schedule* is a schedule such that for each activity, the associated constraint only imposes a time window for the beginning of the activity, while the duration constraint is the original one in the network, and the ending interval is implicit. Formally, a schedule $\langle([Ebegin_1, Lbegin_1],[m_1,n_1],[Eend_1,Lend_1]),\ldots,([Ebegin_k,Lbegin_k],[m_k,n_k],[Eend_k,\ Lend_k])\rangle$ for the activities $A_1,\ldots,A_k$ within a constraint network N, is called *free* if (i) the duration constraint in the schedule is the same as that in N (i.e., $m_i = minD_i$ and $n_i = maxD_i$) for each activity A_i, and (ii) $Eend_i = Ebegin_i + m_i$ and $Lend_i = Lbegin_i + n_i$.

In this case the agent of the activity is just told the set of time instants at which it can begin and it simply has to meet the duration constraints as declared in the workflow specification. Referring to our running example and assuming the time for OPe is 8, we find one of the free schedules is the one assigning 9 and 11 to the beginning of S and S', respectively. The schedule can be represented as ⟨([9,9],[3,7],[12,16]), ([11,11],[1,3],[12, 14])⟩. This is clearly a schedule by definition. It is also clear that this is a free one since the duration constraints $[3,7]$ and $[1,3]$ are as declared in the constraint network, and condition (ii) is easily verified to be true.

3.2 Finding free schedules

We propose a general algorithm for free schedule generation (FSG algorithm from now on) which consists of three main steps:

1. decompose the constraint graph according to OR-split operators and derive the minimal network for each resulting constraint network;

2. for each network characterize the set of free schedules for the given set of activities to be scheduled;

3. derive a particular free schedule from the result of Step 2 above, according to some criteria.

Deriving the minimal network in Step 1 can be easily done by applying a path-consistency algorithm since each constraint network defines an STP. The minimal network guarantees that none of its constraints can be tightened without loosing a possible solution. If the original duration constraints for the activities to be scheduled are tightened during Step 1, no free schedule can exist and the algorithm terminates. Indeed, this will mean that one of the minimum (or maximum) duration declared by an agent cannot be used as part of any solution, i.e., if the activity really uses the minimum (or maximum) amount of time as declared,

then the whole network is not satisfiable, and some constraints sooner or later will be violated.

When the FSG algorithm is called for a schedule of the activities S and S' in our example, Step 1 derives the implicit constraints and domains depicted in Figure 4.[2] In this case, neither durations were tightened, i.e., they maintained the original values. This means that the use of the minimum or maximum durations are not entirely ruled out by the network, and we can still hope to derive free schedules.

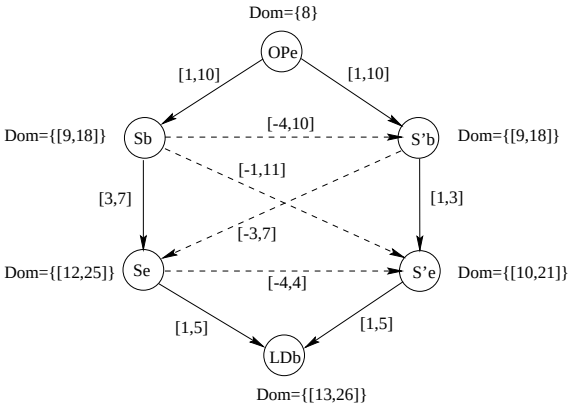

Figure 4: The minimal network derived by path-consistency

Step 2 is nontrivial and it is supported by a theoretical result which needs some preliminary observations.

Definition A pair of intervals I and I' is called *sync-free* with respect to a constraint on the distance between elements of I and elements of I', if each pair (x, y) with $x \in I$ and $y \in I'$ satisfies the constraint.

For example, let $I = [9, 10]$ and $I' = [11, 13]$. Then I and I' are sync-free with respect to the constraint $[1, 4]$, but not sync-free with respect to $[2, 4]$, nor $[1, 3]$.

A simple test for sync-freeness is given by Lemma le:sync.

Lemma 1 *A pair of intervals $I = [minI, maxI]$, $I' = [minI', maxI']$ is sync-free with respect to constraint $[m, n]$ from I to I', if and only if: $(minI' - maxI) \geq m$ and $(maxI' - minI) \leq n$*

[2]For the sake of clarity, we do not depict the implicit constraints originating from the first and last node since they won't be used in the algorithms.

We now want to understand the properties of schedules in terms of sync-free intervals.

Lemma 2 *Let $\mathcal{S}$ be a well-formed schedule for activities $A_1, \ldots, A_n$ in a network N, A_i and A_j be any two of these activities, $I(A_i)$ the beginning or ending interval in $\mathcal{S}$ for A_i, and similarly $I(A_j)$ for A_j. Then, the intervals $I(A_i)$ and $I(A_j)$ are sync-free with respect to the temporal constraint between the corresponding nodes in N.*

Conversely, given a minimal network N including the constraints among n activities, any set of triples $([Ebegin_i, Lbegin_i], [m_i, n_i], [Eend_i, Lend_i])$ for $i = 1 \ldots n$ with the values in the first and last intervals included in the domain of the corresponding node in N, and the second included in the corresponding constraint in N, is a schedule if the above sync-free property holds on each pair of beginning/ending intervals of different activities with respect to the corresponding constraint in N.

If we look at the example of activities S and S', and we consider free schedules having a single starting value, from Lemma 1 we see that the following conditions must be satisfied:
$x_{S'} - x_S \geq Min(TC(Sb, S'b))$
$x_{S'} - x_S \leq Max(TC(Sb, S'b))$
$x_{S'} + minD_{S'} - x_S \geq Min(TC(Sb, S'e))$
$x_{S'} + maxD_{S'} - x_S \leq Max(TC(Sb, S'e))$
$x_S + minD_S - x_{S'} \geq Min(TC(S'b, Se))$
$x_S + maxD_{S'} - x_{S'} \leq Max(TC(S'b, Se))$
$x_{S'} + minD_{S'} - (x_S + maxD_S) \geq Min(TC(Se, S'e))$
$x_{S'} + maxD_{S'} - (x_S + minD_S) \leq Max(TC(Se, S'e))$
where x_S and $x_{S'}$ are the starting values, $TC(node_1, node_2)$ represents the {T}emporal {C}onstraint assigned to the arc from $node_1$ to $node_2$, and $Min(TC(node_1, node_2))$ and $Max(TC(node_1, node_2))$ represent the lower and upper bounds of the constraint, respectively.

A simple elaboration of these equations and its generalization to k activities provides the basis for Theorem 1.

Theorem 1 *Let $A_1, \ldots, A_k$ be activities to be enacted, and N a minimal network of constraints on these and possibly other activities endpoints. The set of all free schedules for $A_1, \ldots, A_k$ in N having a single value as the beginning interval is the set of schedules of the form $([x_i, x_i], [minD_i, maxD_i], [x_i + minD_i, x_i + maxD_i])$ for each activity A_i, where $minD_i$ and $maxD_i$ are the duration bounds for A_i in N, and the values $x_1, \ldots, x_k$ satisfy the conditions:*

- *for each activity* A_j, $Min(Dom(A_jb)) \leq x_j \leq Max(Dom(A_jb))$

- *for each pair of activities* $\langle A_i, A_j \rangle$ *with* $i < j$, $x_i + k_{ij} \leq x_j \leq x_i + K_{ij}$, *where the constants* k_{ij} *and* K_{ij} *are derived from the following:* $k_{ij} = Max((maxD_i - minD_j + Min(TC(A_ie, A_je))), Min(TC(A_ib, A_jb)), (Min(TC(A_ib, A_je)) - minD_j), (maxD_i - Max(TC(A_jb, A_ie))))$, *and* $K_{ij} = Min((minD_i - maxD_j + Max(TC(A_ie, A_je))), Max(TC(A_ib, A_jb)), (Max(TC(A_ib, A_je)) - maxD_j), (minD_i - Min(TC(A_jb, A_ie))))$.

The first condition in the theorem ensures that starting and ending values are included in their corresponding domains. The second is derived from Lemma 1 as shown above, and it ensures that the constraints of the network will be satisfied independently from the specific amount of time taken by each activity within its duration constraint.

Note that all values (included $minD_i, maxD_i$) used in the calculations of the constant expressions are given by the minimal network computed in Step 1. Hence, the theorem characterizes all the single-beginning-point free schedules in N, the minimized network. As mentioned earlier, if the value $minD_i$ or $maxD_i$ in N are changed by the consistency algorithm, there does not exist any free schedule. The theorem, however, is still useful since if the system of inequations has any solution, these solutions will give restricted due-time or bounded schedules, depending on whether the consistency algorithm has tightened only maximum duration bounds or some of the minimum ones.

The theorem also provides an efficient way to find a free schedule. Indeed, the inequations given by Theorem 1 can be expressed in a new constraint network with n nodes, with domains of variables bound by the first inequations and constraints given by the second ones. We apply the consistency algorithm to this network obtaining a minimal network representing the schedule solution space. If we take the minimal value from each domain, the resulting set of values is guaranteed to be a solution [DMP91], and, in terms of our problem, this solution provides the earliest free schedule. Technically, "earliest" here means that this solution identifies the point closest to the origin in the n-dimensional space representing all solutions. Intuitively, it is a schedule with the earliest enactment times.

Consider our running example, and, in particular, the minimal network as shown in Figure 4. Applying Theorem 1, we obtain from the first condition $x_S \in [9, 18]$, $x_{S'} \in [9, 18]$, and, from the second condition, $2 \leq x_{S'} - x_S \leq 4$. These constraints define a new network having only two nodes for S and S' with the corresponding domains and a single arc from S to S' labeled by the constraint $[2, 4]$. When the propagation algorithm is applied, the resulting minimal network is identical except that the domain of S' has been tightened to [11,18]. Indeed, the value 9 and 10 cannot be part of any solution. According to the procedure illustrated above, the earliest single-valued free schedule is obtained taking as starting instant x_S and $x_{S'}$, the values 9 and 11, respectively, as they are the minimal values of the minimal domains in the network representing the solution space. It is easily checked that it is a free schedule. Note that this is exactly the example given at the end of Section 3.1.

3.3 Optimizing free schedules

The schedule we have identified in the previous section is not necessarily "maximal", in the sense that we may extend some intervals in a free schedule to still retain the property of freeness. Intuitively, the larger the intervals, the more relaxed the constraints are on activities. An interesting problem is to find maximal free schedules. To do that, we only need to observe that any (high-dimensional) rectangular region in the solution space given in Theorem 1 yields a free schedule, and to find a maximal free schedule is to find the largest rectangular region. It is easily seen that in some situations, there are many different maximal free schedules. In certain cases, we may be interested in finding an "optimal" free schedule. In these cases, we may rely on standard optimization algorithms to find the "optimal" rectangular regions.

4 Conclusion

In this paper, we investigated the enactment scheduling problem in the context of workflow systems with autonomous agents and temporal constraints among the workflow activities. The notion of *free schedule* we have introduced is particularly powerful in this context, since, if such a schedule exists, the agents performing the workflow activities are essentially free from the need of communicating with each other in order to successfully com-

plete the workflow, satisfying the global temporal constraints. Our scheduling algorithm should be integrated with the ones used by the workflow enactment service at runtime, and it should be run each time the workflow execution reaches an AND-split node, i.e., when parallel activities should be enacted.

We are extending this work in several directions. One of them considers situations where a free schedule does not exist; intuitively, this is the case when duration bounds provided by agents have a wide range and/or when synchronization among activities is quite strict. In these cases, the most reasonable schedules become *restricted due-time schedules* which impose a due-time to the agents in charge of the activities, by restricting the maximum duration. The algorithm presented in the previous section must be extended to find these schedules.

Another direction considers the derivation of schedules in the case of constraints in terms of different time granularities. This becomes necessary when this kind of constraints are given in the specification, and the approximation introduced by a conversion into a common time unit is not acceptable by the workflow application. The results in [BWJ97, BWJ98] give some insight on the problems involved in the extension.

References

[ADEM97] G. Alonso, D. Agrawal, A. El Abbadi, and C. Mohan. Functionalities and Limitations of Current Workflow Management Systems. In *IEEE Expert (Special Issue on Cooperative Information Systems)*, 1997.

[BWJ98] C. Bettini, X. Wang, and S. Jajodia. A General Framework for Time Granularity and its Application to Temporal Reasoning. *Annals of Mathematics and Artificial Intelligence*, 22(1,2), 1998.

[BWJ97] C. Bettini, X. Wang, and S. Jajodia. Satisfiability of Quantitative Temporal Constraints with Multiple Granularities. In *Proc. of 3rd Int.l Conf. on Principles and Practice of Constraint Programming*, Springer-Verlag LNCS 1330, 1997.

[DMP91] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49, 1991.

[EPR99] J. Eder, E. Panagos, and M. Rabinovich. Time constraints in workflow systems. In *Proc. of 11th Int.l Conf. on Advanced Information Systems Enigneering*, 1999.

[MO99] O. Marjanovic and M.E. Orlowska. On Modeling and verification of Temporal constraints in Production Workflows. *Knowledge And Information Systems*, 1(2), 1999.

[SC93] V. Suresh and D. Chaudhuri. Dynamic scheduling: a survey of research. *International Journal of Production Economics*, 32(1), 1993.

[WfMC99] Workflow Management Coalition. Terminology & Glossary. Document Number WFMC-TC-1011. 1999. `http://www.aiim.org/wfmc`