A Logical Framework for Scheduling Workflows Under Resource Allocation Constraints *

Pinar Senkul
Middle East Technical University
karagoz@ceng.metu.edu.tr

Michael Kifer
Stony Brook University
kifer@cs.stonybrook.edu

Ismail H. Toroslu
University of Central Florida
toroslu@cs.ucf.edu

Abstract

A *workflow* consists of a collection of coordinated tasks designed to carry out a well-defined complex process, such as catalog ordering, trip planning, or a business process in an enterprise. *Scheduling* of workflows is a problem of finding a *correct* execution sequence for the workflow tasks, *i.e.*, execution that obeys the constraints that embody the business logic of the workflow. Research on workflow scheduling has largely concentrated on temporal constraints, which specify correct ordering of tasks. Another important class of constraints — those that arise from resource allocation — has received relatively little attention in workflow modeling. Since typically resources are not limitless and cannot be shared, scheduling of a workflow execution involves decisions as to which resources to use and when. In this work, we present a framework for workflows whose correctness is given by a set of resource allocation constraints and develop techniques for scheduling such systems. Our framework integrates Concurrent Transaction Logic (CTR) with constraint logic programming (CLP), yielding a new logical formalism, which we call *Concurrent Constraint Transaction Logic*, or *CCTR*.

1 Introduction

A workflow is a coordinated set of activities that act together to achieve a well-defined goal. Typical examples of workflows include multi-agent banking transactions, trip planning, catalog ordering and fulfillment processes, and manufacturing processes in an enterprise. A *workflow management system* (abbr, *WfMS*) provides a model and tools for specification, analysis, and execution of workflows. Surveys of the area can be found in [21, 17, 3].

Scheduling of workflows is a problem of finding a *correct* execution sequence for the workflow tasks, *i.e.*, an execution that obeys the constraints that embody the business logic of the workflow. Research on workflow scheduling has largely concentrated on temporal constraints, which specify correct ordering of tasks [4, 32, 33, 36, 1, 35, 13, 6].

Another important class of constraints — those that arise from resource allocation — has received relatively little attention in workflow modeling. Examples of such resources (sometimes called *agents*) include company personnel and physical objects, like workshop devices that a task might need in order to accomplish its goal. It is also typical to associate costs (time, budget) with execution of a task by an agent. Since typically agents cannot be shared and costs are not limitless, scheduling of a workflow execution involves decisions as to which resources to use and when. Although resource management has been recognized as an important aspect of a WfMS [12, 2, 36], most of the work has focused on *modeling* the various resources [39, 14, 22] with no or little attention devoted to *scheduling* under the constraints associated with such resources. In this paper, we present a framework for workflows whose correctness is specified in terms of a set of resource allocation constraints and develop techniques for scheduling such systems.

To make the distinction clear, it is useful to give some examples of the temporal and causality constraints considered in [4, 32, 13] and resource allocation constraints considered here. A *temporal/causality constraint* is typically of the form, *tasks 1 and 2 must both execute* (with a possible variation that task 2 executes after task 1) or *if task 1 executes then tasks 2 and 3 must execute as well* (with possible variations that task 3 must come after task 2 and both must occur after task 1). A *resource allocation constraint* typically

*This work was supported in part by the Scientific and Technical Research Council of Turkey (TUBITAK) and NSF grants INT-9809945 and IIS-0072927.

Proceedings of the 28th VLDB Conference, Hong Kong, China, 2002

takes the following forms: *If task 1 is executed by some agent, task 2 should be executed by the same agent as well* with a possible variation that the total time or some other aggregate cost function should not exceed certain limit. Another typical resource allocation constraint is that the same agents cannot be assigned to parallel branches of the same workflow (because each branch might require undivided attention from a human agent or a machine might not be used for different tasks simultaneously). We group the resource allocation constraints into two categories: *cost constraints* are constraints on some aggregate cost functions, while *control constraints* are constraints on how to allocate resources. For instance, *if task 1 is executed by some agent, then task 2 should be executed by the same agent* is a control constraint, whereas *total cost should not exceed a given amount* is a cost constraint. This separation turns out to be useful for our formal framework.

Our approach is based on *Concurrent Constraint Transaction Logic* (abbr., *CCTR*), an extension of Concurrent Transaction Logic (abbr., *CTR*) [7] that incorporates the ideas from Constraint Logic Programming [23, 24]. The contributions of this paper include both CCTR itself and the workflow modeling framework based on this logic.

In our framework, the user specifies workflows using CTR, as in [13].[1] In addition, sets of possible agents for each task and their costs are specified. We then provide a transformation algorithm that takes the given workflow specification (in CTR + constraints) and returns a new workflow expressed CCTR. The new specification incorporates resource allocation constraints in such a way that any valid execution of this specification in CCTR is guaranteed to satisfy the original constraints. Finally, we describe a system that produces a workflow schedule for the new workflow specification together with a solution set to the constraints. To the best of our knowledge, this is the first work that defines a formalism for modeling and scheduling workflows under resource allocation constraints, which incorporates Operations Research (OR) and constraint solving techniques.

A number of research areas are related to our work. Workflow scheduling under temporal constraints and resource management have already been mentioned. Other related areas are job shop scheduling [10, 5, 18, 11, 38], planning in Artificial Intelligence (AI) [30, 8, 19, 9, 29, 28] and agent-based workflow systems [37, 34, 25, 26, 27, 20]. We discuss the relationship between these works and ours in Section 9.

This paper is organized as follows. Section 2 presents a concrete example to illustrate the problem. Section 3 briefly sketches the use of CTR for workflow specification. Sections 4 and 5 introduce CCTR and constraint system used in our framework. Sections 6 and 7 develop the framework for modeling resource allocation constraints in workflows. Section 8 illustrates the use of our framework on a concrete application. Section 9 gives the related work and Section 10 concludes the paper.

[1] It should be understood that nobody expects a workflow engineer to learn logic or any other formalism, such as Petri Nets. These formalisms are typically hidden behind graphical user interfaces.

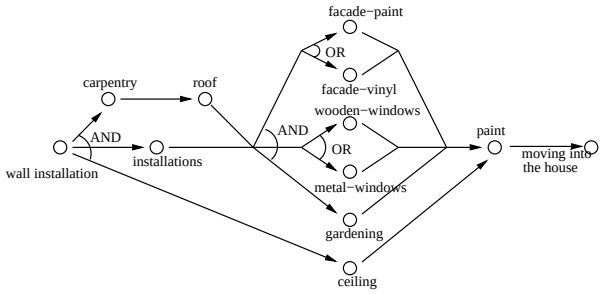

Figure 1: House construction workflow

2 A Motivating Example

The following example, derived from [31], shows a scenario that can be handled by our framework, and we will be using this example throughout the paper.

Example 2.1 *Company A* builds a house, does gardening and moves customer's furniture into the new house. The company subcontracts with other companies for the various subtasks. There can be several candidate subcontractors, or the same company may be qualified to do several subtasks. To satisfy customer's requirements and to maximize its own profit, company A wants to choose the most appropriate companies to subcontract with. The workflow is shown in Figure 1. In the figure, the AND-nodes represent branches of work that can be done in parallel (but all the parallel branches must be finished). OR-nodes represent alternative courses of action. For instance, the facade can be painted or the customer might choose to use vinyl siding. Tasks that must be done in sequence are connected via directed edges.

Resource allocation constraints in this workflow can include:

1. The budget for the construction should not exceed the given amount.

2. The construction should not last longer than the given duration.

3. Different companies must be chosen for parallel tasks (to speed up the construction).

The first and second constraints of the example are defined on total expenditure and total construction duration. For this reason, they are cost constraints. The third one is a control constraint, since it controls how subcontractors are chosen, and does not involve numeric costs.

Once the workflow and the constraints of this example are defined in our framework, the specification passes through a system, which consists of a transformation engine, CTR engine and a constraint solver. This system produces a *valid schedule* (i.e., an execution order for the subtasks) and a solution set for the constraints (subtask-to-subcontractor assignments that obey the constraints).

3 Concurrent Transaction Logic and Workflows

Concurrent transaction logic (CTR) was introduced in [7] and has been shown to be a powerful modeling and reasoning tool. In particular, it is one of the few formalisms that have been successfully applied to modeling and reasoning about workflows [13, 6].

CTR extends classical logic with four new connectives and modalities of which the most important are $\otimes$, the *serial conjunction*, and $|$, the *parallel conjunction*. The semantics of CTR is based on *multipaths*. A *path* is a sequence of one or more *database states* (*e.g.*, $\langle d_1, ..., d_n \rangle$) and a *multipath* (abbr., *m-path*) is a sequence of paths (*i.e.*, $\langle p_1, ..., p_m \rangle$, where each p_i is a path). Informally, a database state can be a collection of facts or even knowledge bases (that consist of facts, rules, etc.), but for this paper we can think of states as simply symbolic identifiers for various collections of facts.

Formulas are viewed as transactions that execute along m-paths and, while doing so they query and change the underlying database state. Execution along an m-path in CTR is tantamount to being true over that m-path. Informally, if ϕ is true on an m-path π_1 (denoted $\pi_1 \models \phi$) and ψ is true on an m-path π_2 ($\pi_2 \models \psi$), then $\phi \otimes \psi$ is true over the concatenated m-path $\pi_1 \bullet \pi_2$. Similarly, $\phi \mid \psi$ is true over the m-path $\pi_1 \parallel \pi_2$, where $\pi_1 \parallel \pi_2$ is an m-path obtained from π_1 and π_2 by some interleaving of the sequences of paths that comprise π_1 and π_2.

To see how CTR can be used to model a workflow, we show a formula that corresponds to one part of the house building workflow in Figure 1:

$$\begin{aligned}&\texttt{wall} \otimes \\ &\quad ((((\texttt{carpentry} \otimes \texttt{roof}) \mid \texttt{installations}) \\ &\qquad \otimes \textit{the-middle-piece}) \mid \texttt{ceiling}) \\ &\quad \otimes \texttt{paint} \otimes \texttt{move}\end{aligned}$$

Each proposition here represents a task and the CTR operators show how to combine the tasks (concurrently or serially). The proposition *the-middle-piece* represents the part of the workflow that did not fit and we present it separately:

$$\begin{aligned}&(\texttt{facade-paint} \vee \texttt{facade-vinyl}) \mid \\ &(\texttt{wooden-windows} \vee \texttt{metal-windows}) \mid \\ &\texttt{gardening}\end{aligned}$$

As can be seen from the last formula, $\vee$ represents alternative executions in a CTR formula. For instance, to execute the above workflow we only need to execute one of the facade-related tasks, not both.

CTR is not only a modeling tool for workflows, but also a scheduler and a reasoner. For instance, [13] shows that a large class of temporal and causality constraints can be represented in CTR and that the proof theory of the logic can be used to perform a number of tasks ranging from consistency checking of a workflow to its scheduling subject to the specified constraints.

4 Concurrent Constraint Transaction Logic (CCTR)

We now develop an extension for CTR, called *Concurrent Constraint Transaction Logic* (abbr., *CCTR*), which integrates CTR with Constraint Logic Programming. Subsequent sections show how CCTR can be used to model and schedule workflows that must obey a wide range of resource allocation constraints.

The syntax of CCTR formulas is the same as in CTR, but the semantics is based on partial schedules instead of m-paths. This change in the semantics will later allow us to add constraints to the logic.

4.1 Semantics

While m-paths are adequate to model serial and concurrent execution in CTR, they are not sufficient to model resource requirements necessary for those executions to succeed. For example, in CCTR we need to be able to distinguish that two m-paths are parts of different concurrent branches of the same execution. To this end, we introduce the notion of a *partial schedule*, which adds certain amount of structure to m-paths.

Partial schedules are defined in terms of two operators: $\bullet_p$ and $\|_p$. The first represents concatenation and is associative; the second does parallel combination of schedules and is both associative and commutative.

Definition 4.1 *A* partial schedule *is defined as follows:*

- *An m-path, π, is a partial schedule*
- *Serial composition of two partial schedules, $\omega_1 \bullet_p \omega_2$, is a partial schedule*
- *Parallel composition of two partial schedules, $\omega_1 \|_p \omega_2$, is a partial schedule*

In addition, we require that a serial composition of m-paths be an m-path: If $\omega = \langle p_1, ..., p_n \rangle$ and $\omega' = \langle p'_1, ..., p'_k \rangle$ are m-paths (*i.e.*, sequences of paths p_1, p_2, etc.) then $\omega \bullet_p \omega'$ is the m-path $\langle p_1, ..., p_n, p'_1, ..., p'_k \rangle$.

4.2 Model Theory of CCTR

The semantics of CCTR is based on *m-path structures* of CTR, which are described in [7]. An m-path structure, M, is a mapping that assigns a regular first-order semantic structure (sometimes called *interpretation*)

to every m-path, ω (*i.e.*, $M(\omega)$ is a first-order semantic structure). For an in-depth exposition of the semantics of CTR and how it is related to execution and transactions we refer the reader to [7]. However, this semantics has a very simple intuitive interpretation: In CTR, a formula is true along an m-path if, in a well-defined sense, it is capable of executing along this path. Similarly, in CCTR, a formula that is true along a partial schedule can be interpreted as being able to execute according to that schedule.

A *CTR goal* is a formula composed of the usual atomic formulas of first-order logic and the connectives $\otimes$, $|$, and $\vee$. We saw an example of such a goal in Section 2. Following [13, 6], CTR goals are used to represent workflows.

Let ω be a partial schedule, M be an m-path structure, and α be a CTR goal. $M, \omega \models \alpha$ (read: α is true in M along the schedule ω) is defined as follows:

- If α is a variable-free atomic formula, then $M, \omega \models \alpha$, if and only if ω is an m-path and $M(\omega) \models_{classic} \alpha$, where $\models_{classic}$ stands for entailment in classic first-order logic (recall that $M(\omega)$ is a first-order semantic structure).
- $M, \omega \models \alpha \otimes \beta$, if and only if $\omega = \omega_1 \bullet_p \omega_2$, $M, \omega_1 \models \alpha$, and $M, \omega_2 \models \beta$.
- $M, \omega \models \alpha \mid \beta$, if and only if $\omega = \omega_1 \parallel_p \omega_2$, $M, \omega_1 \models \alpha$, and $M, \omega_2 \models \beta$.
- $M, \omega \models \alpha \vee \beta$, if and only if either $M, \omega \models \alpha$ or $M, \omega \models \beta$.
- Universal and existential quantification is defined as usual in first-order logic.

The first item in the above definition states that a transaction named α is true along the m-path ω (recall that m-paths form the base of the definition of a schedule) if the m-path structure M says that α is true along ω in the classical sense. The intuitive meaning of this statement is that α is the name of a transaction that can "execute" along ω. The second item says that the transaction $\alpha \otimes \beta$ can execute along a schedule ω if and only if this schedule is a concatenation of two schedules and α can execute along the prefix-schedule while β can execute along the suffix-schedule. The third item states that a parallel combination of transactions, $\alpha \mid \beta$, can execute along a schedule ω if and only if it is a parallel combination of schedules, $\omega_1 \bullet_p \omega_2$, and α can execute along ω_1 and β along ω_2. The fourth item says that in order to execute $\alpha \vee \beta$ along a schedule it is enough to be able to execute α or β separately.

In addition to the above, a *CTR rule* has the form $head :- body$, where *head* is an atomic formula and *body* is a CTR goal. The semantics of such a rule is analogous to first-order logic: It is satisfied in an m-path structure M if, for every partial schedule ω, whenever $M, \omega \models body$ is true then so is $M, \omega \models head$.

5 Constraint Systems

So far we have not strayed far from the original CTR. Our next step is to define a constraint system, which allows us to talk about resources required for executing workflow activities and constraints on these resources. First, we need to introduce the notion of a resource and associate resources to CTR formulas (*i.e.*, subworkflows). Then we introduce two types of constraints: cost constraints, which involve aggregate functions defined on execution schedules, and control constraints, which restrict the way resources can be allocated to different formulas (subworkflows).

5.1 Basic Definitions

Definition 5.1 *A* resource *is an object with the attributes* token *and* cost.

In workflow modeling, a resource typically represents an execution agent. The attribute *token* then represents this agent's name and the attribute *cost* represents the cost (or multiple costs, if more than one cost factor is used) of using that agent. For notational convenience, we assume that the function *cost_of*( ) returns the value of the cost attribute of the resource and *token_of*( ) returns the value of the token attribute.

Definition 5.2 *A* resource assignment *is a partial mapping from partial schedules to sets of resources. A resource assignment $asg(\omega)$ must satisfy the following conditions:*

$asg(\omega_1 \parallel_p \omega_2) = asg(\omega_1) \cup asg(\omega_2)$,
if both $asg(\omega_1)$ and $asg(\omega_2)$ are defined
$asg(\omega_1 \bullet_p \omega_2) = asg(\omega_1) \cup asg(\omega_2)$,
if both $asg(\omega_1)$ and $asg(\omega_2)$ are defined
$asg(\omega) = S$, *where S is some set of resources,*
if ω is an m-path

Definition 5.3 *A* constraint universe $\mathcal{D}$ *is a set of domains together with predicates associated with each domain. The domains in the constraint universe are*

1. Elementary Domains: *Scalar domain (e.g. integer), goal domain (i.e., the set of all CTR goals, i.e., formulas that represent workflows — see the previous section), the domain of partial schedules, the domain of resource assignments, the domain of resources.*
2. Complex Domains: *Domains that are composed out of elementary domains using various set constructors (e.g., goal $\times$ partial schedule, $2^{resources}$).*

Each domain in $\mathcal{D}$ has a set of predicates associated with that domain.

Example 5.4 Here are some examples of constraints in $\mathcal{D}$:

Definition	Commutative	Dist. over Union
$disjoint(V_1, V_2) = (token\_of(V_1) \cap token\_of(V_2) \equiv \varnothing)$	Yes	Yes
$subset(V_1, V_2) = (token\_of(V_1) \subset token\_of(V_2))$	No	Yes
$subsumes\_c(V_1, V_2) = ((token\_of(V_1) \cap token\_of(V_2)) \subset token\_of(c))$	Yes	Yes

Figure 2: Examples of set constraints and their properties

1. $disjoint(R_1, R_2) \equiv (token\_of(R_1) \cap token\_of(R_2) = \varnothing)$ is a predicate on the domain $(2^{resources} \times 2^{resources})$ where $R_1, R_2 \in$ *resource domain* and $token\_of(R) = \{t_i \mid t_i \in token\_of(i)\}$

2. $less\_than\_c(I) \equiv (I < c)$ is a predicate on the integer domain, where $I \in$ *integer domain* and c is an integer constant.

3. $cost\_constraint(\omega, \rho) \equiv less\_than\_c(f(\omega, \rho))$ is a predicate on the domain (*partial schedule domain* $\times$ *resource assignment domain*), where ω is a schedule, ρ is a resource assignment, $less\_than\_c$ is defined above, and f is a function with the signature (*schedule* $\times$ *resource assignment* $\rightarrow$ *integer.*)

5.2 Definition of Constraint Systems

Definition 5.5 *A* constraint system ζ *is a set of constraint definitions. It consists of two subsystems* ζ_{cost} *and* ζ_{ctrl}*.*

The ζ_{cost} subsystem is used to specify cost constraints (*e.g.*, this task must execute in less than 1 day); the ζ_{ctrl} subsystem is used to specify control constraints (*e.g.*, the copier on the second floor cannot be used by two concurrent tasks).

Definition 5.6 *The constraint subsystem* ζ_{cost} *consists of predicates of the form* $cost\_constraint(\omega, asg)$*, where* ω *is a partial schedule and* asg *is a resource assignment. More specifically,* $cost\_constraint(\omega, asg)$ *has the form* $value\_constraint(cost(\omega, asg))$*, where value_constraint is a predicate over a scalar domain (e.g.,* integer*) and* cost *is a function with the following properties: Let* ω_1 *and* ω_2 *be partial schedules such that both* $cost(\omega_1, asg)$ *and* $cost(\omega_2, asg)$ *are defined. Then:*

$cost(\omega_1 \parallel_p \omega_2, asg) \equiv op_|(cost(\omega_1, asg), cost(\omega_2, asg))$
$cost(\omega_1 \bullet_p \omega_2, asg) \equiv op_\otimes(cost(\omega_1, asg), cost(\omega_2, asg))$
$cost(\omega, asg) \equiv cost\_of(asg(\omega))$*, where* ω *is an m-path and* $cost\_of$ *is defined right after Definition 5.1*

Here $op_\otimes$ *and* $op_|$ *are functions with the signature scalar domain* $\times$ *scalar domain* $\rightarrow$ *scalar domain.*

Definition 5.7 *The constraint subsystem* ζ_{ctrl} *consists of predicates of the form* $ctrl\_constraint(\omega, asg)$*, which satisfy the following conditions. Let* $\omega, \omega_1, \omega_2$ *be partial schedules and* asg *be an assignment such that* $asg(\omega), asg(\omega_1)$ *and* $asg(\omega_2)$ *are defined. Then*

$ctrl\_constraint(\omega_1 \bullet_p \omega_2, asg) \equiv$
$\quad set\_constraint_\otimes(asg(\omega_1), asg(\omega_2)) \wedge$
$\quad ctrl\_constraint(\omega_1, asg) \wedge ctrl\_constraint(\omega_2, asg)$

$ctrl\_constraint(\omega_1 \parallel_p \omega_2, asg) \equiv$
$\quad set\_constraint_|(asg(\omega_1), asg(\omega_2)) \wedge$
$\quad ctrl\_constraint(\omega_1, asg) \wedge ctrl\_constraint(\omega_2, asg)$

$ctrl\_constraint(\omega, asg) \equiv leaf\_constraint(asg(\omega))$,
$\quad$ *where* ω *is an m-path*

Here $set\_constraint_\otimes$ *and* $set\_constraint_|$ *are predicates over the domain* $2^{resources} \times 2^{resources}$*; they express conditions on sets of resources, such as disjointness. The predicate* $leaf\_constraint$ *is defined over the domain* $2^{resources}$*; it constrains executions of individual tasks and can be used to say that, for example, task* t_1 *cannot be executed by agent A.*

To ensure that *cost_constraint* is well-defined, we impose the following restrictions on $op_|$ and $op_\otimes$:

- **Commutativity:** $op_|(X, Y) = op_|(Y, X)$.
- **Associativity:**
  $op_|(op_|(X, Y), Z)) = op_|(X, op_|(Y, Z))$
  $op_\otimes(op_\otimes(X, Y), Z)) = op_\otimes(X, op_\otimes(Y, Z))$.

Lemma 5.8 *The function "cost" in Definition 5.6 is well-defined.*

Similarly, to ensure that *ctrl_constraint* is well-defined, we impose the following restrictions on $set\_constraint_\otimes$ and $set\_constraint_|$.

- **Commutativity:**

$set\_constraint_|(V_1, V_2) = set\_constraint_|(V_2, V_1)$.

- **Distribution over Union:**

$set\_constraint_\otimes(V_1 \cup V_2, V_3) =$
$\quad set\_constraint_\otimes(V_1, V_3) \wedge set\_constraint_\otimes(V_2, V_3)$

$set\_constraint_|(V_1 \cup V_2, V_3) =$
$\quad set\_constraint_|(V_1, V_3) \wedge set\_constraint_|(V_2, V_3)$.

Lemma 5.9 *ctrl_constraint in Definition 5.7 is well-defined.*

Although there are no restrictions on the use of the attributes *cost* and *token* in resource objects, *cost* is typically used in *cost_constraint* and *token* in *ctrl_constraint*. The functions $op_\otimes$ and $op_|$ are usually aggregates, such as `sum` or `max`, and $set\_constraint_|$ and $set\_constraint_\otimes$ are various set constraints, such as those in Figure 2.

Example 5.10 Let V_1, V_2, c denote sets of resources. Figure 2 lists some Boolean set functions along with their distributivity and commutativity properties. Those that have both properties can be used as $set\_constraint_|$ and those that have only distributivity can be used as $set\_constraint_\otimes$ only.

Example 5.11 Example 2.1 has two cost constraints that would be defined in ζ_{cost} and one control constraint that would be defined in ζ_{ctrl}. Due to space limitation, we give only some of the definitions. For ζ_{cost}, the function $cost(\,)$ of Definition 5.6 should return the costs of the assignment — the construction time and the dollar amount. We can represent this as a list where first element is the time and second is the amount: $cost(\omega, asg) = cost\_of(asg(\omega)) = [V_1, V_2]$

The following constraint can be used to state that total time should not exceed c_1 and budget should not exceed c_2: $value\_constraint([V_1, V_2]) \equiv V_1 < c_1, V_2 < c_2$

The functions $op_|$ and $op_\otimes$ define how the cost of assignment is aggregated. For instance, for parallel executions, maximum of the execution time is used, whereas for dollar costs, payments are added up: $op_|([V_1, V_2], [V_1', V_2']) \equiv [V_1 + V_1', max(V_2, V_2')]$

The following set constraint, which specifies that the agent sets allocated to parallel branches of a schedule must be disjoint, could be part of ζ_{ctrl}:

$$set\_constraint_|(R_1, R_2) \equiv (token\_of(R_1) \cap token\_of(R_2)) = \varnothing)$$

5.3 Logical Entailment with Constraints

Satisfaction in a constraint system is defined as follows. Let ζ be a constraint system, $\mathcal{D}$ be a constraint universe, ω be a partial schedule, asg be a resource assignment, and M be an m-path structure. Then, given a CTR goal ϕ, we write $(M, \zeta, \omega, asg) \models \phi$ if and only if

1. $M, \omega \models \phi$,
2. $\mathcal{D} \models cost\_constraint(\omega, asg)$, and
3. $\mathcal{D} \models ctrl\_constraint(\omega, asg)$.

In other words, ω is a good schedule, *i.e.*, the workflow represented by ϕ can execute along ω and both the cost constraints and control constraints are satisfied by the constraint system.

6 CCTR as a Workflow Scheduler

We now have all the machinery needed for specifying workflows with constraints. In this section, we define a transformation that takes any workflow specification, ϕ, in CTR, a constraint system, ζ, and a constraint universe, $\mathcal{D}$, and returns a new workflow specification, ϕ', and a resource assignment, asg, such that for any m-path structure M:

TRANSFORMATION RULES

$\mathbf{B}(G_1 \vee G_2) \equiv \mathbf{B}(G_1) \vee \mathbf{B}(G_2)$

$\mathbf{B}(G) \equiv$
$\mathbf{R}(G, T) \otimes cost\_constraint(T) \otimes ctrl\_constraint(T)$
if the main connective in G is $\otimes$ or $|$

$\mathbf{R}(A, T) \equiv A \otimes (T = resource\_asg(A, Agents))$
where A is an atomic task, $resource\_asg$ is a resource assignment term for task A and $Agents$ is a new variable or a list of variables

$\mathbf{R}(G_1|G_2, T) \equiv$
$(T = {}'|'(T_1, T_2)) \otimes (\mathbf{R}(G_1, T_1)|\mathbf{R}(G_2, T_2))$
$'|'(T_1, T_2)$ is just a function term; the function symbol $'|'$ is chosen for its resemblance to the parallel composition connective $|$

$\mathbf{R}(G_1 \otimes G_2, T) \equiv$
$(T = {}' \otimes' (T_1, T_2)) \otimes (\mathbf{R}(G_1, T_1) \otimes \mathbf{R}(G_2, T_2))$
$' \otimes' (T_1, T_2)$ is just a function term; the function symbol $' \otimes'$ is chosen for its resemblance to the sequential composition connective $\otimes$

$\mathbf{R}(G_1 \vee G_2, T) \equiv \mathbf{R}(G_1, T) \vee \mathbf{R}(G_2, T)$

Figure 3: Transformation Rules for a Workflow Prescheduler

1. for any partial schedule ω that satisfies the constraints (*i.e.*, such that $\mathcal{D} \models cost\_constraint(\omega, asg) \wedge ctrl\_constraint(\omega, asg)$) we have that $M, \omega \models \phi'$ if and only if $M, \omega \models \phi$; and
2. for *any* schedule ω, $(M, \zeta, \omega, asg) \models \phi'$.

In other words, ϕ and ϕ' are equivalent on schedules that satisfy the constraints, but ϕ' satisfies the constraints "automatically," *i.e.*, *all* of its executions are schedules that satisfy the constraints. Thus, any CCTR interpreter, such as the one described in Section 7, becomes a workflow scheduler.

The aforesaid transformation is the main step in constructing a workflow scheduler. It and the formal CCTR framework presented earlier are two main contributions of our approach.

Transformations that satisfy properties 1 and 2 above are called *correct pre-schedulers*.

6.1 Building a Workflows Prescheduler

We now begin to develop a correct pre-scheduler for constraint workflows. The transformation itself is defined in Figure 3 by induction on the structure of the workflow. The operator $\mathbf{B}$ transforms the original workflow into a new workflow, which preserves the semantics, but also includes cost and control constraints. In addition, we introduce rule templates, which the user can instantiate in order to describe the constraint

$step_1$	$\mathbf{B}((c \otimes r)\|(i\|g))$
$step_2$	$\mathbf{R}((c \otimes r)\|(i\|g), T) \otimes$ $cost\_constraint(T) \otimes ctrl\_constraint(T)$
$step_3$	$T = '\|'(T_1, T_2) \ \otimes \ (\mathbf{R}(c \otimes r, T_1)\|\mathbf{R}(i\|g, T2))) \otimes$ $cost\_constraint(T) \otimes ctrl\_constraint(T)$
$step_4$	$T = '\|'(T_1, T_2) \otimes$ $(\ (T_1 = '\otimes'(T_3, T_4) \ \otimes \ (\mathbf{R}(c, T_3) \otimes \mathbf{R}(r, T_4))) \ \|$ $\ (T_2 = '\|'(T_5, T_6) \ \otimes \ (\mathbf{R}(i, T_5)\|\mathbf{R}(g, T_6))) \ ) \otimes$ $cost\_constraint(T) \ \otimes \ ctrl\_constraint(T)$
$step_5$	$T = '\|'(T_1, T_2) \otimes$ $(\ (T_1 = '\otimes'(T_3, T_4) \ \otimes$ $\ (((c \ \otimes \ T_3 = rsrc(c, W)) \ \otimes$ $\ (r \ \otimes \ T_4 = rsrc(r, X)))) \ \|$ $\ (T_2 = '\|'(T_5, T_6) \ \otimes$ $\ (((i \ \otimes \ T_5 = rsrc(i, Y)) \ \|$ $\ (g \ \otimes \ T_6 = rsrc(g, Z)))) \ ) \otimes$ $cost\_constraint(T) \ \otimes \ ctrl\_constraint(T)$

Figure 4: Prescheduling Transformation for House Construction Workflow

system appropriate for the application at hand. Theorem 7.2 states that **B** is a correct pre-scheduler.

The operator **B** invokes another operator, **R**, which takes a workflow, G, and creates a *workflow resource allocation term*. The latter looks exactly like the parse tree of G, but the leaves, which are propositions that represent tasks, are replaced with task resource assignments. In our representation, a resource assignment can be any term of the form $resource\_asg(task, Agents)$, where *task* is a constant that represents the task to which resources are assigned and *Agents* is a new variable or a list of new variables.

The purpose of the workflow resource allocation term is that it encodes the workflow structure, and this structure is used in defining the constraint system.[2] Task resource assignments, which are sitting at the leaves of the workflow resource allocation term, have variables that (when instantiated by a constraint solver) represent concrete resource assignments to the workflow tasks. Two kinds of resource allocation constraints are added to the workflow specification by the prescheduler **B**:

- *cost_constraint*: This is a constraint on the aggregate cost of the workflow.
- *ctrl_constraint*: This constraint controls resource allocation to workflow tasks.

Example 6.1 Consider a subset of Example 2.1:

$(\texttt{carpentry} \otimes \texttt{roof}) \ | \ (\texttt{installations} \ | \ \texttt{gardening})$

[2]Recall that the constraint system defined in Section 5.2 has a number of requirements that have to do with the structure of partial schedules. It can be shown that the structure of such a schedule depends on the workflow parse tree. Thus, to be able to construct the constraints properly we must know the parse tree of the corresponding workflow.

The prescheduling transformation of this subworkflow is shown in Figure 4, where the long task names are abbreviated to c, r, i, and g, respectively. The transformed workflow is given in $step_5$.

1. $ctrl\_constraint(\|(T_1, T_2)) \ :-$ $\quad$ **set_constraint**$_\|(T_1, T_2), ctrl\_constraint(T_1),$ $\quad$ $ctrl\_constraint(T_2)$
2. $ctrl\_constraint(\otimes(T_1, T_2)) \ :-$ $\quad$ **set_constraint**$_\otimes(T_1, T_2), ctrl\_constraint(T_1),$ $\quad$ $ctrl\_constraint(T_2)$
3. $ctrl\_constraint(T) \ :- \ leaf(T),$ **leaf_constraint**(T)
4. $cost\_constraint(T) \ :-$ $\quad$ $cost(T, V),$ **value_constraint**(V)
5. $cost(\otimes(T_1, T_2), V) \ :-$ $\quad$ $cost(T_1, V_1), cost(T_2, V_2),$ **op**$_\otimes(V_1, V_2, V)$
6. $cost(\|(T_1, T_2), V) \ :-$ $\quad$ $cost(T_1, V_1), cost(T_2, V_2),$ **op**$_\|(V_1, V_2, V)$
7. $cost(resource\_asg(T, Agents), V) \ :-$ $\quad$ **cost_of**$(resource\_asg(T, Agents), V)$

Figure 5: Template Rules for Constraint Systems

Resource Assignment:
$resource\_asg(T, Agents)$ – placeholder for a user-specified term, which associates resources to a tasks. T denotes an atomic task that can be represented by a single variable, and *Agents* denotes the resource that can be represented by a single variable or a list of variables (in case of multiple resources).

User Predicates (typically defined via user-supplied rules):
cost_of$(resource\_asg(T, Agents), V)$ – placeholder for predicate that tells the costs associated with the resources. V has the data type of the costs attribute of the resource. It can be a single variable or a list of variables.
set_constraint$_\|(T_1, T_2)$ – placeholder for a control constraint for parallel composition of tasks; T_1 and T_2 must have the same user-defined data type.
set_constraint$_\otimes(T_1, T_2)$ – placeholder for a control constraint for sequential composition of tasks; T_1 and T_2 have the same user-defined data type.
leaf_constraint(T) – placeholder for a constraint on individual tasks; T has a user-defined data type.
value_constraint(V) – placeholder for a predicate used to define cost constraints. V has a user-defined data type.
op$_\|(V_1, V_2, V)$ – placeholder for aggregate operator that tells how to compute the cost (V) of a parallel composition of subworkflows from the costs (V_1, V_2) of those subworkflows. Used in the definition of cost constraints. V, V_1, and V_2 must have the same user-defined data type.
op$_\otimes(V_1, V_2, V)$ – similar to **op**$_\|$, but used for serial compositions of subworkflows.

Figure 6: Placeholders for Problem-Specific Predicates and Resource Assignments

6.2 Specifying Constraint Systems

Details of the constraint systems of CCTR introduced in Section 5.2 can vary greatly, but their general properties can be realized as a single set of Prolog rule *templates* shown in Figure 5. In the figure, the boldface predicates are *placeholders* for functions and constraints that the user can specify to adapt the template to a particular application domain. These placeholders are explained in Figure 6. Later we illustrate the use of these templates on a number of nontrivial examples.

Rules 1 to 3 in Figure 5 define *ctrl_constraint* — the control constraint for |-nodes, ⊗-nodes, and leaves of the workflow parse tree, respectively. Rules 4 to 7 define *cost_constraint* — the cost constraint. Again, this is done separately for each type of node in the parse tree.

Lemma 6.2 *The constraint template in Figure 5 defines a constraint system in the sense of Section 5.2, provided that the actual predicates that replace the boldface placeholders have the appropriate associativity and commutativity properties stated in that section.*

Example 6.3 The placeholder definitions for the constraints of Example 2.1 are shown in Figure 7.

Let the placeholder $resource\_asg(T, Agents)$ be of the form $rsrc(T, A)$, where T represents task and A the agent

$\mathbf{cost\_of}(rsrc(T, A), [V, U]) \ :-$
$\qquad duration(T, A, V),\ price(T, A, U)$
$\mathbf{value\_constraint}([V, U]) \ :- \ V < c_1,\ U < c_2$
$\mathbf{set\_constraint}_{|}(T_1, T_2) \ :- \ disjoint(T_1, T_2)$
$\mathbf{set\_constraint}_{\otimes}(T_1, T_2) \ :- \ true$
$\mathbf{leaf\_constraint}(T) \ :- \ true$
$\mathbf{op}_{|}([V_1, U_1], [V_2, U_2], [V, U]) \ :-$
$\qquad V \text{ is } max(V_1, V_2),\ U \text{ is } U_1 + U_2$
$\mathbf{op}_{\otimes}([V_1, U_1], [V_2, U_2], [V, U]) \ :-$
$\qquad V \text{ is } V_1 + V_2,\ U \text{ is } U_1 + U_2$

Figure 7: Placeholders for House Construction Example

7 The Big Picture

The scheduling process under resource allocation constraints is depicted in Figure 8. The process has three main components:

- the transformation rules and the templates defined in Figures 3 and 5
- the inference system of CCTR (not presented here due to space limitation)
- an off-the-shelf constraint solver

The steps in this process are as follows:

Step 1. A given workflow definition, G, is transformed into a new workflow definition $G' := \mathbf{B}(G)$ using the prescheduler $\mathbf{B}$ defined in Section 6.

Step 2. A partial schedule is found for the workflow. In this step, resource allocation constraints are collected as a set of atomic constraints.

Step 3. Under the partial schedule of the previous step, solutions to the resource allocation constraints are found using the off-the-shelf constraint solver.

The systems returns all schedules and solutions to the workflow constraints. Any constraint solver that can handle all of our constraints can be used. A prototype of this architecture has been implemented with the help of the CTR interpreter available at `www.cs.toronto.edu/~bonner/ctr/index.html`.

The prototype has a graphical interface for the user to define the workflow control and the placeholders for the constraint system. The template rules are already defined in the system and basic guidelines on how to fill in the bodies of the user-specified predicates are provided to the user, as well. Pressing the *transform* button feeds the workflow specification to the prescheduler $\mathbf{B}$, which is realized as a Prolog program. The prescheduler returns a transformed workflow, which is ready to be run and scheduled by the CTR interpreter.

The CTR interpreter, together with the prescheduler, simulates the behavior of CCTR inference system. It takes the transformed workflow (the output of $\mathbf{B}$) together with constraint definitions and produces a schedule that obeys the specified control flow and a set of constraints that must be satisfied in order for the schedule to be valid.

The constraint set is then sent to the constraint solver, which returns a solution to the resource allocation constraints. Through backtracking, the system can return all valid schedules and resource allocations. Our implementation uses the constraint solver provided by XSB,[3] since the CTR interpreter is realized as an XSB application.

Example 7.1 The pre-scheduled House construction workflow specification of Example 6.1, the associated constraints of Example 6.3, and the template rules are fed to the CTR interpreter as shown in Figure 8. The interpreter returns a schedule and a set of constraints as shown in Figure 9. Next, the constraint set is sent to the constraint solver, which yields a set of valid resource allocations. If the solution set is empty, it means that there is no valid schedule satisfying the constraints.

[3]XSB is a high-performance deductive database and Prolog system available at `http://xsb.sourceforge.net/`

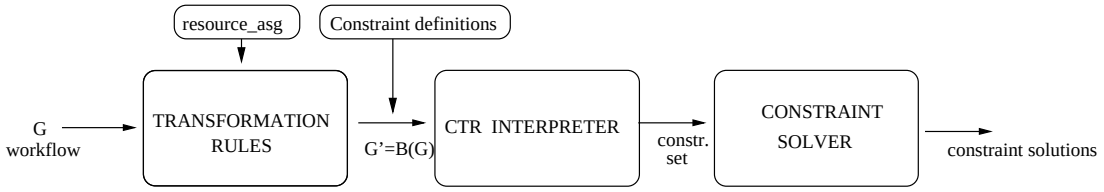

Figure 8: The Big Picture

base predicates	$duration(c, W, V_1),\ price(c, W, U_1),$ $duration(r, X, V_2),\ price(r, X, U_2)$ $duration(i, Y, V_4),\ price(i, Y, U_4),$ $duration(g, Z, V_5),\ price(g, Z, U_5)$
cost constraints	V_3 is $V_1 + V_2,\ U_3$ is $U_1 + U_2,$ V_6 is $max(V_4, V_5),\ U_6$ is $U_4 + U_5,$ V is $max(V_3, V_6),\ U$ is $U_3 + U_6,$ $V < c_1,\ U < c_2$
ctrl constraints	$Y \neq Z,\ W \neq Y,\ W \neq Z,$ $X \neq Y,\ X \neq Z$

Figure 9: Constraints Computed for House Construction Workflow

Theorem 7.2 *Let ζ be a constraint system and $\mathcal{D}$ be a constraint domain, which can be represented using the template rules in Figure 5 (plus the additional definitions for the boldface placeholders). Then, the transformation* $\mathbf{B}$ *in Figure 3 is a correct prescheduler, i.e., it satisfies conditions 1 and 2 given at the beginning of Section 6.*

8 Applications

The framework presented in this paper fits a large number of applications. We were able to apply it successfully to job shop scheduling, travel planning, scheduling of conference presentations, and others.

In this section we present two applications of workflow scheduling under resource allocation constraints.

8.1 Job Shop Scheduling

We illustrate how a simple job shop scheduling problem can be represented as a workflow with resource allocation constraints. There are four jobs: a, b, c and d, and two precedence relations: job a must be completed before job b starts, and job c must be completed

$resource\_asg(T, Agents) : rsrc(T, A)$
$\mathbf{cost\_of}(rsrc(T, A), V)\ :-\ duration(T, A, V)$
$\mathbf{value\_constraint}(T)\ :-\ T < c.$
$\mathbf{set\_constraint}_{|}(T_1, T_2)\ :-\ disjoint(T_1, T_2).$
$\mathbf{set\_constraint}_{\otimes}(T_1, T_2)\ :-\ true.$
$\mathbf{op}_{|}(V_1, V_2, V)\ :-\ V$ is $max(V_1, V_2).$
$\mathbf{op}_{\otimes}(V_1, V_2, V)\ :-\ V$ is $V_1 + V_2.$
$\mathbf{leaf\_constraint}(T)\ :-\ true.$

Figure 10: Placeholders for Job Shop Scheduling with One Cost Constraint

$step_1$	$\mathbf{B}((a \otimes b)\|(c \otimes d))$
$step_2$	$\mathbf{R}((a \otimes b)\|(c \otimes d), T) \otimes$ $cost\_constraint(T) \otimes ctrl\_constraint(T).$
$step_3$	$((T = '\|'(T_1, T_2)) \otimes$ $(\mathbf{R}((a \otimes b), T_1)\|(\mathbf{R}(c \otimes d), T2))$ $\otimes cost\_constraint(T) \otimes ctrl\_constraint(T).$
$step_4$	$((T = '\|'(T_1, T_2)) \otimes$ $(((T_1 = ' \otimes' (T_3, T_4)) \otimes (\mathbf{R}(a, T_3) \otimes \mathbf{R}(b, T_4)))\|$ $((T_2 = ' \otimes' (T_5, T_6)) \otimes (\mathbf{R}(c, T_5) \otimes \mathbf{R}(d, T_6))) \otimes$ $cost\_constraint(T) \otimes ctrl\_constraint(T).$
$step_5$	$((T = '\|'(T_1, T_2)) \otimes$ $((T_1 = ' \otimes' (T_3, T_4)) \otimes$ $((a \otimes T_3 = rsrc(a, W)) \otimes (b \otimes (T_4 = rsrc(b, X)))))\|$ $((T_2 = ' \otimes' (T_5, T_6)) \otimes$ $(c \otimes T_5 = rsrc(c, Y)) \otimes (d \otimes T_6 = rsrc(d, Z)))) \otimes$ $cost\_constraint(T) \otimes ctrl\_constraint(T)$

Figure 11: Prescheduling Transformation for Job Shop Scheduling Problem

base predicates	$duration(a, W, V_1),\ duration(b, X, V_2),$ $duration(c, Y, V_3),\ duration(d, Z, V_4)$
cost constraints	V_5 is $V_1 + V_2,\ V_6$ is $V_3 + V_4,$ V is $max(V_5, V_6), V < c$
ctrl const.	$W \neq Y,\ W \neq Z,\ X \neq Y,\ X \neq Z$

Figure 12: Constraints Computed for Job Shop Scheduling with One Constraint

before job d starts. The ordering relations among tasks can be captured by a workflow G with the definition $((a \otimes b) \mid (c \otimes d))$. Suppose that there are three machines, each capable of doing each of the four jobs, but they take different time to complete these jobs. The resource allocation constraints are as follows:

1. The total duration of the workflow should be less than a given value.

2. The set of agents assigned to concurrent branches should be disjoint.

The problem-specific placeholders to be used in the constraint template (Figure 5) for this example are given in Figure 10. Here *duration* is a base predicate that specifies the duration of various tasks and the *disjoint* constraint is defined as in the house construction example.

The transformation $\mathbf{B}$ yields a workflow definition G' shown in Figure 11 and constraints used in G' are in Figure 12.

$resource\_asg(T, Agents) : rsrc(T, A)$
cost_of$(rsrc(T, A), [V, U])$ $: -$
$duration(T, A, V),\ quality(T, A, U)$
value_constraint$([V, U])$ $: - V < c_1,\ U > c_2$
set_constraint$_{|}(T_1, T_2)$ $: - disjoint(T_1, T_2)$
set_constraint$_{\otimes}(T_1, T_2)$ $: - true$
leaf_constraint(T) $: - true$
op$_{|}([V_1, U_1], [V_2, U_2], [V, U])$ $: - V$ is $max(V_1, V_2)$,
U is $U_1 + U_2$
op$_{\otimes}([V_1, U_1], [V_2, U_2], [V, U])$ $: - V$ is $V_1 + V_2$,
U is $U_1 + U_2$

Figure 13: Placeholders for Job Shop Scheduling with Two Cost Constraints

base predicates	$duration(a, W, V_1),\ quality(a, W, U_1),$ $duration(b, X, V_2),\ quality(b, X, U_2),$ $duration(c, Y, V_4),\ quality(c, Y, U_4),$ $duration(d, Z, V_5),\ quality(d, Z, U_5)$
cost constraints	V_3 is $V_1 + V_2,\ U_3$ is $U_1 + U_2,$ V_6 is $V_4 + V_5,\ U_6$ is $U_4 + U_5,$ V is $max(V_3, V_6),\ U$ is $U_3 + U_6,$ $V < c_1, U > c_2$
ctrl constraints	$W \neq Y,\ W \neq Z,$ $X \neq Y,\ X \neq Z$

Figure 14: Constraint Computed for Job Shop Scheduling with Two Cost Constraints

8.2 Job Shop Scheduling with Multiple Cost Constraints

We now modify the above example to allow more constraints. Let, for example, the additional constraint be *the quality measure must exceed certain given value*. To handle this, the placeholders are modified by adding an extra variable for the quality measure (see Figure 13). The transformation **B** in this case is similar to the case of one constraint, but the resulting set of constraints is slightly different, as shown in Figure 14.

9 Related Work

A number of works have dealt with workflow scheduling, resource management in workflows, and scheduling using constraint satisfaction techniques. However, to the best of our knowledge, none combines these disparate techniques to provide a framework for workflow scheduling under resource allocation constraints. In the following, we discuss these related works.

Workflow scheduling: Research on workflow scheduling has largely concentrated on temporal constraints. In [4, 32, 33], temporal logic and specialized algebras are used for scheduling. [13, 6] are based on CTR and [35] presents a scheduling method based on an action description logic. Besides these logic-based approaches, Petri nets are also used for workflow scheduling [1, 36]. In all of mentioned works, *temporal and causality constraints* determine the correctness of the schedule. In contrast, we deal with *resource allocation constraints* — a problem that requires substantially different machinery. In [15, 16], deadline constraints and how to avoid deadline violations are discussed. However, it does not provide any formal framework for solving these problems.

Resource management in workflows: Resource management has been recognized as an important issue in a WfMS [12, 2]. However, most of the work in this area have focused on modeling of resources with little attention devoted to scheduling. In [39], a framework for the representation of resources and an organizational model for workflow system is given. [14] describes an architecture of a workflow resource manager that can integrate external resources. [22] proposes a method to handle policies (i.e., general guidelines for resource allocation) in a workflow resource manager. These policies are defined independently of the workflow schedule. While such policies can serve as guidelines on how resources can be allocated during execution, they do *not* constitute a scheduling algorithm. Our approach differs from this prior work in that it deals with resource allocation and scheduling *together*.

Job-shop scheduling: Operations research (OR) has developed a number of successful algorithms for solving many scheduling problems [10, 5]. Among them, job-shop scheduling is most relevant to workflow scheduling. Constraint logic programming (CLP), which integrates logic-based and OR techniques, has also been successfully used to deal with job-shop scheduling problems. The following works is just a tip of an iceberg of the vast research on the subject [18, 11, 38]. Both of the job-shop scheduling and workflow scheduling problems incorporate resource allocation constraints. However, a workflow can be much more complex than a job-shop. Furthermore, the focus of our work is orthogonal to the works on constraint solving — we *use* constraint solvers in Stage 3 of our scheduling process.

Planning in AI: As in the scheduling problem in OR, constraint programming is used for planning in AI [8, 30], and there also are works on planning under resource constraints [19, 9]. However, only a small number of works in this area propose planning as a workflow scheduling technique [28, 29]. However, those that do deal with scheduling do not address this problem under resource allocation constraints. Instead, planning techniques are used to schedule dynamically changing workflows.

Agent-based workflow systems: Using agent technology for workflow systems is another related research area [37, 34, 25, 26, 27, 20]. In agent-based workflow systems, execution decisions are based on the communication events that occur when one agent requests services of another. Research on this area has largely concentrated on intelligent agent modeling and communication issues. Only a few [26] briefly mention

the issue of scheduling under resource allocation constraints, which they propose to do through negotiations among the agents. However, to the best of our knowledge, details of such techniques have not been worked out.

10 Conclusion and Future Work

We presented a logical framework for scheduling workflows under resource allocation constraints. The framework is based on Concurrent Constraint Transaction Logic (CCTR) and integrates Concurrent Transaction Logic [7] with Constraint Logic Programming [23, 24]. We presented an algorithm that takes the initial workflow specification and a set of resource allocation constraints and returns a new workflow and a resource assignment, such that every execution of that workflow is guaranteed to satisfy the constraints. The system was prototyped with the help of the CTR interpreter available at `www.cs.toronto.edu/~bonner/ctr/index.html`.

Although not discussed here, the proposed framework can be extended with temporal and causality constraints of the kind that is discussed in [13].

As a future work, our framework can be extended with special-purpose constraint solvers that are optimized for our framework. In addition, scheduling under resource allocation constraints for dynamic workflows and scheduling concurrent workflow instances under resource allocation constraints are also interesting topics for future work.

Acknowledgements. We would like to thank Hasan Davulcu and C.R. Ramakrishnan for the suggestions that helped improve this paper.

References

[1] N. Adam, V. Atluri, and W. Huang. Modeling and analysis of workflows using Petri nets. *Journal of Intelligent Information Systems*, 10(2):131–158, March 1998.

[2] G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, R. Günthör, and C. Mohan. Advanced transaction models in workflow contexts. In *International Conference on Data Engineering*, New Orleans, Louisiana, February 1996.

[3] G. Alonso, D. Agrawal, A. El Abbadi, and C. Mohan. Functionality and limitations of current workflow management systems. In *IEEE-Expert. Special issue on Cooperative Information Systems*, 1997.

[4] P. Attie, M. Singh, A. Sheth, and M. Rusinkiewicz. Specifying and enforcing intertask dependencies. In *Int'l Conference on Very Large Data Bases*, Dublin, Ireland, August 1993.

[5] J. Blazewicz, K. H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz. *Scheduling Computer and Manufacturing Processes*. Springer-Verlag, 1996.

[6] A.J. Bonner. Workflow, transactions, and datalog. In *ACM Symposium on Principles of Database Systems*, Philadelphia, PA, May/June 1999. Long version available at `http://www.cs.toronto.edu/~bonner/papers.html#transaction-logic`.

[7] A.J. Bonner and M. Kifer. Concurrency and communication in transaction logic. In *Joint Int'l Conference and Symposium on Logic Programming*, pages 142–156, Bonn, Germany, September 1996. MIT Press.

[8] P. Van Breek and X. Chen. CPlan: A constraint programming approach to planning. In *Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 585–590, 1999.

[9] M. Brenner. A formal model for planning with time and resources in concurrent domains. In *IJCAI Workshop on Planning with Resources*, Seattle, USA, August 2001.

[10] P. Brucker. *Scheduling Algorithms*. Springer-Verlag, 1995.

[11] Y. Caseau and F. Laburthe. Improved CLP scheduling with task intervals. In *Int'l Conference on Logic Programming*, 1994.

[12] Workflow Management Coalition. Terminology and glossary ver 3.0. Technical Report (WFMC-TC-1011), Workflow Management Coalition, Brussels, February 1999.

[13] H. Davulcu, M. Kifer, C.R. Ramakrishnan, and I.V. Ramakrishnan. Logic based modeling and analysis of workflows. In *ACM Symposium on Principles of Database Systems*, pages 25–33, Seattle, Washington, June 1998.

[14] W. Du, j. Davis, Y. Huang, and M. Shan. Enterprise workflow resource management. In *Int'l Workshop on Research Issues in Data Engineering*, pages 108–115, Sydney, Australia, 1999.

[15] J. Eder, E. Panagos, H. Pezewaunig, and M. Rabinovich. Time management in workflow systems. In *Int. Conf. on Business Infomation Systems*, pages 265–280, Poznan, Poland, 1999.

[16] J. Eder, E. Panagos, and M. Rabinovich. Time constraints in workflow systems. In *Conference on Advanced Information Systems Engineering*, pages 286–300, Germany, 1999.

[17] D. Georgakopoulos, M. Hornick, and A. Sheth. An overview of workflow management: From process modeling to infrastructure for automation. *Journal on Distributed and Parallel Database Systems*, 3(2):119–153, April 1995.

[18] H. Goltz and U. John. Methods for solving practical problems of job-shop scheduling modelled in clp(fd). In *Conf. on Practical Application of Constraint Technology*, London, April 1996.

[19] P. Halsum and H. Geffner. Heuristic planning with time and resource. In *IJCAI Workshop on Planning with Resources*, Seattle, USA, August 2001.

[20] M. Hannebauer. From formal workflow models to intelligent agents. In *AAAI-99 Workshop on Agent Based Systems in the Business Context*, pages 19–24, Orlando, USA, 1999.

[21] Special issue on workflow systems. *Bulletin of the Technical Committee on Data Engineering (IEEE Computer Society)*, 18(1), March 1995.

[22] Y. Huang and M. Shan. Policies in a resource manager of workflow systems: Modeling,enforcement and management. In *International Conference on Data Engineering*, 1999.

[23] J. Jaffar and J.-L. Lassez. Constraint logic programming. In *ACM Symposium on Principles of Programming Languages*, Munich, W. Germany, 1987.

[24] J. Jaffar and M. J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20, May/July 1994.

[25] N.R. Jennings, P. Faratin, M.J. Johnson, and P. O'Brien ad M.E. Wiegand. Using intelligent agents to manage business processes. In *Int'l Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, pages 345–360, London, 1996.

[26] N.R. Jennings, T.J. Norman, and P. Faratin. Adept: An agent-based approach to business process management. *ACM SIGMOD Record*, 27(4):32–39, 1998.

[27] N.R. Jennings, T.J. Norman, P. Faratin, P. O'Brien, and B. Odgers. Auotonomous agents for business process management. *Int'l Journal of Applied Artificial Intelligence*, 14(2):145–189, 2000.

[28] K.L. Myers and P. M. Berry. At the boundary of workflow and ai. In *AAAI-99 Workshop on Agent Based Systems in the Business Context*, Orlando, USA, 1999.

[29] A. Nareyek. Applying local search to structural constraint satisfaction. In *IJCAI Workshop on Intelligent Workflow and Process Management: The New Frontier fo AI in Business*, Stockholm, Sweden, August 1999.

[30] A. Nareyek. AI planning in a constraint programming framework. In G. Hommel, editor, *Communication-Based Systems*, pages 163–178. Kluwer Academic Publishers, 2000.

[31] C. Schulte and G. Smolka. Finite domain constraint programming in oz. a tutorial. Version 1.1.0, February 2000.

[32] M.P. Singh. Semantical considerations on workflows: An algebra for intertask dependencies. In *Proceedings of the International Workshop on Database Programming Languages*, Gubbio, Umbria, Italy, September 6–8 1995.

[33] M.P. Singh. Synthesizing distributed constrained events from transactional workflow specifications. In *Proceedings of 12-th IEEE Int'l Conference on Data Engineering*, pages 616–623, New Orleans, LA, February 1996.

[34] M.P. Singh and M. H. Huhns. Multiagent systems for workflow. *Int'l Journal of Intelligent Systems in Accounting, Finance and Management*, 8:105–117, 1999.

[35] G. Trajcevski, C. Baral, and J. Lobo. Formalizing (and reasoning about) the specifications of workflows. In *Proceedings of the International Conference on Cooperative Information Systems (CoopIS)*, September 1996.

[36] W. M. P. van der Aalst. The application of petri nets to workflow management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[37] F. Wan, K. Rustogi, J. Xing, and M.P. Singh. Multiagent workflow management. In *IJCAI Workshop on Intelligent Workflow and Process Management: The New Frontier for AI in Business*, Stockholm, Sweden, August 1999.

[38] J. Wurtz. Oz scheduler: A workbench for scheduling problems. In *Proceedings of International Conference on Tools with Artificial Intelligence (ICTAI)*, 1996.

[39] M. zur Muhlen. Resource modeling in workflow applications. In *Workflow Management Conference*, pages 137–153, Muenster, Germany, November 1999.