An architecture for workflow scheduling under resource allocation constraints

Pinar Senkul*, Ismail H. Toroslu

Department of Computer Engineering, Middle East Technical University 06531 Ankara, Turkey

Received 22 November 2002; received in revised form 4 March 2004; accepted 19 March 2004

Abstract

Research on specification and scheduling of workflows has concentrated on temporal and causality constraints, which specify existence and order dependencies among tasks. However, another set of constraints that specify resource allocation is also equally important. The resources in a workflow environment are agents such as person, machine, software, etc. that execute the task. Execution of a task has a cost and this may vary depending on the resources allocated in order to execute that task. Resource allocation constraints define restrictions on how to allocate resources, and scheduling under resource allocation constraints provide proper resource allocation to tasks. In this work, we provide an architecture to specify and to schedule workflows under resource allocation constraints as well as under the temporal and causality constraints. A specification language with the ability to express resources and resource allocation constraints and a scheduler module that contains a constraint solver in order to find correct resource assignments are core and novel parts of this architecture.

© 2004 Elsevier Ltd. All rights reserved.

Keywords: Workflow; Scheduling; Resource; Resource allocation constraints; Constraint programming

1. Introduction

Workflow is a collection of tasks organized to accomplish some business process. It also defines the order of task invocation or conditions under which task must be invoked, task synchronization, and information flow [1]. Patient treatment in a hospital, banking services, manufacturing and catalog ordering processes and many more activities in today’s business life can be modeled as workflow. The nature of the tasks in a workflow may vary considerably. These tasks may be manual jobs or computer programs that execute on different platforms. A workflow management system (WFMS) provides a model and tools for specification, analysis, and execution of workflows. It forms a framework to capture the relations among the tasks (i.e., the business logic of the workflow), to gather the tasks with different nature and to automate the execution...
in such a way that it obeys the business logic. The correct definition and execution of the workflow means more quality and less cost for an establishment. For this reason, workflows have been an attractive research area. Surveys on the subject can be found in [1–6,49].

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. Approaches in this area are typically based on temporal logic and specialized algebras [7–9], Petri nets [10,11], and concurrent transaction logic and other logics for representing actions [12–16]. Research on workflow scheduling has largely concentrated on temporal and causality constraints, which specify existence and order dependencies among the tasks. For instance, task _1_ must execute before task _2_ or if task _1_ executes, task _2_ must execute as well (also known as Klein’s existence constraint [17]) are examples of temporal/causality constraints considered in [7,9,15].

However, another set of constraints that specify resource allocation are equally important. Examples of resources in a workflow are agents such as person, machine, software, etc. that execute the task. Resource allocation constraints define restrictions on the resources of the workflow. Execution of a task has a cost, in terms of money, time etc., and this may vary depending on the resources allocated in order to execute the task. Although resource management has been recognized as an important aspect of a WFMS, most of the work has focused on modeling issues [2,10,18] or agent behavior in workflow systems [19–21] with little attention devoted to scheduling. Scheduling under resource allocation constraints include decisions on which resources to allocate and when to allocate them and thus provide proper resource allocation. It seems natural to include resource allocation constraints into the workflow specifications in the areas that require efficient scheduling in order to obtain fast responds to user’s demands on the process’ budget, duration, etc. as in catalog ordering, manufacturing. Davulcu et al. [22] discusses about constraints that involve execution cost of tasks and that are used for modeling and reasoning of virtual enterprises modeled as workflows. These constraints may be seen as early version for resource allocation constraints.

Simple resource allocation constraints are constraints like if task _1_ is executed by some agent, task _2_ should be executed by the same agent as well. Moreover, total cost < 100 is an example of resource allocation constraints in a system where resource allocation to execute tasks has some associated cost values. Cost may consist of one or more of dimensions such as financial expenditure to fulfill the task by a given agent, the execution time, or the energy consumption, etc. Several resources may be available and qualified to perform the given task with different cost values. It is possible to group resources under roles and define hierarchical role structures.

Most of the previous work on workflows concentrated on issues related to run-time check of the constraints. If a workflow is executed before it is verified, its constraints may be checked and a schedule might be obtained incrementally during the execution. However, a workflow specification might be unexecutable because of its incorrect design or conflicting constraints. At some point of the execution, if it is detected that the workflow cannot be completed because of design errors, some rollback operations might be needed before abandoning the execution of the workflow. Obviously, such cases cause waste of resources. Therefore, it is important to verify the given workflow specification, and this verification can be done by obtaining a feasible workflow schedule providing proper resource allocation.

In this work, we present a workflow management system architecture that provides modules to model resources and resource allocation constraints and to find schedules fulfilling these constraints. In order to find schedules, constraint programming approach is used. That is, scheduler incorporates an off-the-shelf constraint solver to obtain a feasible schedule for the workflow satisfying both resource allocation and temporal/causality constraints. Operations research (OR) and constraint programming have been successfully used for problems such as job-shop scheduling, in which proper machine (i.e., resource) allocation constitutes an important part of the solution. (For more information on constraint programming, reader may refer to [23–26].) However, contrary to most OR problems, our goal is to find a feasible resource allocation rather than finding the optimal solution. A typical constraint solver is not
sufficient for finding optimal solution. General algorithms for finding optimal solutions for problems which are specified in terms of constraints are usually not very efficient, and customized algorithms are needed for different classes of problems. This work can be extended to produce optimal solutions by incorporating such algorithms into the proposed system. However, for efficient scheduling, we consider finding any feasible solution that meets the constraints. The study of Senkul et al. [27] is another work that uses constraint programming for workflow scheduling under resource allocation constraint. In [27], a formal model for the problem is presented. In this work, we demonstrate the practicality of the ideas presented in [27], by a system that has a script language which constructs to model resources and resource allocation allocation constraints and that uses constraint solver as the core of the scheduler.

This paper presents a system to schedule workflows under resource allocation constraints. Since the concentration of the paper is on pre-scheduling of workflows, concurrent workflows have not been considered. Indeed, at the very basic level the concurrency can be defined in our model by using the AND blocks. However, our approach does not support the scheduling of general concurrent workflows.

The main contributions of this paper may be summarized as follows:

- As a part of the architecture, we have developed a specification language that can model cost information for resources and specify resource allocation constraints, which is not provided by previous workflow definition languages.
- The enactment service of the architecture contains a scheduler module, which incorporates a constraint solver. This module gets the workflow specification, which is translated into constraint language, and produces a schedule satisfying resource allocation constraints.

This paper is organized as follows: In Section 2, the rationale and motivation behind scheduling workflows under resource allocation constraints are given through examples. In Section 3, workflow structure considered in this paper is explained. In Section 4, resource allocation constraints for workflows are explained. In Section 5, the architecture and the workflow specification in the proposed system is presented. In Section 6, the semantics of the specification language in terms of constraint language is given. Section 7 discusses the related work. Finally, a summary is given in the conclusion.

2. Motivating examples

In this section, first a typical workflow example is given and it is shown that how this workflow can be extended with some possible resource allocation constraints. As the second example, we give a more complex workflow with various temporal/causality and resource allocation constraints.

Example 2.1 (Classical Example). Fulfilling customer service requests in a telecommunications company is a typical workflow example that is used by previous work on workflows, as in [1]. In this workflow, customer request is obtained via telephone or a service request web page. After customer and request information are checked and recorded, and circuits are checked, the installation starts. As installation proceeds, telephone directory is updated. When the installation is completed, customer is notified. The tasks of this workflow and relations between them are shown in Fig. 1. Typically, basic ordering relations among the tasks are represented by control flow graphs, as in Fig. 1. AND show the concurrent branches and XOR shows alternative tasks. Labelings on the arrows may be used to denote the pre-conditions for the tasks that the arrow points to.

\(^1\)Block types are explained in Section 3.
In this example, computerized and manual tasks together constitute the workflow. Resources for manual tasks are human participants and software/hardware are the resources for computerized tasks. There are different levels of costs and qualities of tasks that a service can provide. If the user wants all installation equipments to be renewed, then the installation becomes high quality, but also with a high cost. On the other hand, if the user demands an economical service, among alternative tasks, the one with the least cost must be chosen. Therefore, obtaining only the correct ordering of tasks is not enough for such cases.

In order to obtain a feasible schedule that meets the demands and the constraints, this workflow can be extended with the following resource allocation constraints:

- If installation quality is the primary consideration of the customer, instead of using existing installation, new installation must be done.
- If the customer wants a low budget installation, existing lines and slots must be used as much as possible.
- If the customer wants the service as soon as possible, the most available technician must be chosen, rather than waiting for the most experienced one to be available.

**Example 2.2** (Complex Example). A more complex example is house construction workflow. This workflow, which is partially from [28], is shown in Fig. 2 and is composed of construction and installation tasks for parts of the house, gardening and moving furniture into the house. Typically, although a single company is responsible for the house construction process as a whole, this company subcontract with smaller companies or handymen for individual tasks. This workflow is a simple case of virtual enterprises, which is a temporary consortium of autonomous organizations to meet short-termed objectives. Today's
information technology supports defining and constructing such consortium as an e-business process. Reducing startup cost is desirable for such workflows and resource allocation constraints can be used to find a schema with feasible cost to establish consortium.

For the house construction workflow, in order to meet customer requirements and to maximize the profit, the construction company wants to choose the proper subcontractors and for this reason, the following resource allocation constraints are defined:

1. The budget of the house construction must be less than $20,000.
2. If carpentry is done by company \( x \), roof construction should be done by company \( x \) as well.
3. If plumbing and plumbing checking are done by the same company, there is a discount of $700.
4. If wall construction cost is more than $1,000, do not choose facade vinyl plate covering.

In addition to resource allocation constraints, the following temporal/causality constraints are defined:

1. Electricity installation construction must be completed before ceiling starts.
2. If facade painting is done, wooden framed window installation should be also done.

Note that, since decision as to whether to execute a task or not affect the total cost and resource allocation, temporal/causality constraints and resource allocation constraints may be interwoven.

In the rest of the paper, we will use the house construction workflow as the running example of the paper. Although workflow processes may vary in their structure and they appear in very different areas from administrative contexts to manufacturing, we chose this example for it is a compact one illustrating a rich set of features.

3. Workflow structure

In this paper, following the process definition standards defined by Workflow Management Coalition (WFMC) in [18], workflow specifications consist of basic block structures. Workflow patterns of van der Aalst et al. [29] contains more complete and comprehensive set of structures than [18]. Our workflow structure supports all basic and advanced patterns (two of the advanced patterns are not directly supported, but can be defined in terms of other patterns), and cycles of van der Aalst et al. [29]. Since the concentration of this paper is on pre-scheduling of workflows, the patterns that are related to executional issues, such as termination, activity cancellation or multiple instances, are not supported by our system. The block structures that are defined by WFMC and supported in our workflow structure are shown in Fig. 3. These block structures are as follows:

- The simplest block being a single task, blocks may have recursive structures.
- In an \textit{AND block}, all sub-blocks are executed and the block successfully finishes execution when all its sub-blocks are completed successfully. Depending on the resources and the constraints, some or all sub-blocks can execute concurrently.
- \textit{OR block} is completed successfully when at least one of the sub-blocks successfully completes execution.
- In an \textit{XOR block (eXclusive Or)} only one of the sub-blocks can be completed. The block successfully finishes execution when at most and at least one of the sub-blocks is completed successfully.
- In a \textit{Sequential block} sub-blocks are executed sequentially in the order of definition.
- \textit{Iteration block} may run more than once repetitively until a given condition holds. Thus, several instances of the iteration block may execute sequentially.
The basic component for workflow is a task. In addition to tasks, conditions are also necessary for workflow specification. *Condition* is a logical expression. It is a part of iteration block or it can be used as a precondition for other blocks. If the condition is met, the block following the condition can be executed as shown in Fig. 3(f).

The above structures capture most of the order and existence dependencies among tasks. However, there may be other order and existence dependencies that cannot be defined using these structures, such as the dependencies between two tasks in different blocks. The typical examples of such temporal/causality constraints may be listed as follows:

- Task $t_i$ must execute.
- Task $t_i$ must not execute.
- If $t_i$ executes, $t_j$ must execute as well. (Klein’s existence constraint in [17]).
- If $t_i$ and $t_j$ both execute, $t_i$ must execute before $t_j$ (Klein’s order constraint in [17]).

4. Resource allocation constraints in workflows

On the contrary to previous workflow scheduling works that deal with temporal/causality constraints, we specify resource allocation constraints as well as temporal/causality constraints and find schedules satisfying all of the given constraints. In this section, we have a closer look at resource allocation constraints.
Resource allocation constraints define restrictions on which resources to allocate and when to allocate them. Some of the constraints directly involve the total execution cost, whereas others define restrictions on resources and relations between resources. From this point of view, we may categorize the resource allocation constraints as cost and control constraints.

- **Cost constraints** are defined on resource allocation cost. For instance, total cost < 40 is a cost constraint.
- **Control constraints** are defined directly on the resource. If task $t_2$ is done by resource $x$, then task $t_5$ must be done by the same resource as well is an example for control constraints.

Another categorization orthogonal to the above is as follows: as in the resource allocation constraints discussed in Example 2.2, satisfaction of some constraints is necessary in order to obtain a solution, whereas satisfaction of some others is not obligatory for the solution, but they rather define rules that facilitate the satisfaction of the constraint set. According to this property, we group resource allocation constraints as hard resource allocation constraints and business rules:

1. **Hard resource allocation constraints**: The satisfaction of these constraints is necessary for the solution. For instance, sum of costs of tasks $t_1$ and $t_2$ must be less than 10 is in this category. Similarly, if task $t_2$ is done by resource $x$, then task $t_5$ must be done by the same resource as well is also an hard constraint. Majority of the resource allocation constraints defined for workflows are in this category.

2. **Business rules**: Sometimes, satisfaction of a certain resource allocation constraint facilitates the satisfaction of total cost constraint. For instance, if tasks $t_1$ and $t_3$ are done by the same resource, then there is a discount of $300 is such a constraint. Discount in this example is discount by the given amount in the total expenditure. Another constraint may be if $t_1$ and $t_2$ are performed on the same computer, execution time of $t_2$ is shortened by 2 min, due to gain in communication and data transfer. Note that discount and reduction in execution time are not actually constraints themselves. For this reason, these constraints are different from conditional constraints.

Sometimes, temporal/causality constraints work together with resource allocation constraints. For instance, a temporal/causality constraint may take part in a resource allocation constraint such as if task $t_5$ costs more than $1000, then do not execute task $t_6$. Similarly, a business rule may contain a temporal/causality constraint as in if task $t_4$ is completed before task $t_6$ begins, then there is a discount of $400. Thus, it is possible to make the following generalization: for hard resource allocation constraints in the form of if $p$ then $q$, $p$ or $q$ may be a temporal/causality constraint. For business rules if $p$ then $q$, $p$ may be a temporal/causality constraint.

5. **System architecture and specification language**

5.1. **System architecture**

Resource specification and management are crucial for workflows. For this reason, we propose an architecture to model and solve resource allocation constraints. This architecture is shown in Fig. 4. In order to specify workflow blocks, temporal/causality constraints, resources and resource allocation constraints, we have developed a specification language, *Workflow Specification Language* (WSL). In our system, the user directly specifies the workflow in WSL. It is also possible to develop a graphical interface tool that may be incorporated into the system. By using this tool, a graphical representation of the

---

2Note that, if both $p$ and $q$ are temporal/causality constraints, if $p$ then $q$ is a temporal/causality constraint as well.
A workflow can be defined by the user and then it is translated into WSL specification. Generally, the changes on the resource definitions are independent of the workflow definitions. Therefore, we keep resource definitions as a separate set of definitions from the workflow definitions as in many proposed workflow systems, such as [30–32]. The resource specification supports role definitions as well. Although there are different definitions for the concept of role in the literature, in this work “role is an abstraction to define the relationship between a set of resources and the capabilities of the resources” [30,31,33]. Although we model flat role structures in our examples, it is possible to extend our definition for more complex hierarchical organizational structures.

The workflow specification in WSL is sent to the enactment service module. The core of this module is the scheduler which contains a constraint solver. Therefore, the workflow and resource definitions in WSL are converted to a set of constraint statements to be processed by the constraint solver. The product of the scheduler is an execution order and the corresponding resource assignments.

Once the execution order and the resource assignments are produced, enactment engine and resource manager may receive the ordering and the resource assignment, respectively, from the scheduler, and executes the workflow. Since, in this work, the emphasis is on specification and scheduling part, we have not implemented the resource manager and the enactment engine. However, since the schedule is already produced, enactment engine and resource manager may simply follow the given execution order and resource assignment, to activate the tasks and resources. The interface between the enactment service and tasks/resources may follow the WfMC definitions [34].

5.2. Workflow specification language: WSL

We have created a script language, WSL, in order to specify basic workflow blocks, defined by WfMC in [18] and summarized in Section 3, temporal/causality constraints, resource information and resource allocation constraints. This subsection defines the syntax of the structures in WSL. The semantics of these
structures are similar to the semantics of the terms used in the syntax, which are also given informally in [18]. In this paper, we will specify the semantics in terms of constraints in the following section.

5.2.1. Workflow blocks

The syntax of the basic workflow blocks defined in Section 3 in WSL are as follows:

- A task is represented as \(<task\_name>(<parameterlist>)\), optionally followed by by \(<role>\) to specify the role of the resource for the given task. The information flow between the task is through parameters. However, in this paper, we do not explain the details of parameter passing and information flow mechanism. Since, in this work, the emphasis is on scheduling, the tasks given in the examples are defined without parameters:
  - And Block: \(\text{AND}\{\text{< List of Block Definitions >}\}\)
  - Or Block: \(\text{OR}\{\text{< List of Block Definitions >}\}\)
  - Xor Block: \(\text{XOR}\{\text{< List of Block Definitions >}\}\)
  - Sequential Block: \(\text{SEQ}\{\text{< List of Block Definitions >}\}\)
  - Iteration Block: \(\text{WHILE}\{\text{<condition>}{\text{< Block Definition >}}\}\)
  - Condition Block: \(\text{IF}\{\text{<condition>}{\text{< Block Definition >}}\}\text{ELSE}\{\text{< Block Definition >}\}\)

An example workflow specification in WSL is given in Fig. 5.

5.2.2. Temporal and causality constraints

In most of the script languages (such as given in [35,36]), temporal/causality constraints cannot be specified. On contrary, WSL provides the constructs to define temporal and causality constraints. These structures have a simple and usual semantics. They can be listed as follows:

1. \(\text{Execute}(<task>)\).
2. \(\text{NotExecute}(<task>)\).
3. \(<task_1>\text{BEFORE}<task_2>\).
4. \(\text{IF } X \text{ THEN } Y\), where \(X\) and \(Y\) are either temporal/causality constraints as defined above or a set of such constraints.

By using the above syntax, Klein’s existence constraint is represented as:

\(\text{IF Execute}(t_i) \text{ THEN Execute}(t_j)\).

Similarly, Klein’s order constraint is

\(\text{IF Execute}(t_i), \text{Execute}(t_j) \text{ THEN } (t_i \text{ BEFORE } t_j)\).

Note that, if part contains two constraints separated by a comma denoting conjunction. For instance, the constraint “if facade painting is done, wooden framed window installation must be done as well”, given in Example 2.2 is represented as

\(\text{IF Execute}(\text{paint}_\text{facade}) \text{ THEN Execute}(\text{install}_\text{wooden}_\text{frame})\).

Another constraint that is common for workflows is \(\text{IF Execute}(t_i) \text{ THEN NotExecute}(t_j)\), meaning either \(t_i\) or \(t_j\) must execute exclusively.

5.2.3. Resources

In addition to the basic blocks, WSL provides structures to define resources in the workflow system. Resource definitions include cost information, which is used for the evaluation of cost constraints. The
role–resources association is modeled as follows:

\[ \text{Resource Name} \text{ BELONGS TO role } \{ \langle \text{cost term}_1 \rangle, \langle \text{cost}_1 \rangle \} \ldots \langle \langle \text{cost term}_n \rangle, \langle \text{cost}_n \rangle \} \].

It is possible to define associations directly between tasks and resources as follows:

\[ \text{Resource Name} \text{ CAPABLE OF task } \{ \langle \text{cost term}_1 \rangle, \langle \text{cost}_1 \rangle \} \ldots \langle \langle \text{cost term}_n \rangle, \langle \text{cost}_n \rangle \} \].

For instance, resource \( r_1 \) that can execute task \( t_2 \) having cost dimensions of money, time and energy with certain values defined as

\[ r_1 \text{ CAPABLE OF } t_2 \text{ WITH } \{ (\text{money}, 10), (\text{time}, 5), (\text{energy}, 2) \} \].

5.2.4. Resource allocation constraints

One of the basic issues of this work is to specify resource allocation constraint for workflows. For this purpose, our specification language, \( WSL \), provides constructs to model resource allocation constraints, fig. 5. House construction workflow in WSL.

\[ \text{wall installation. money } + \text{ carpentry. money } + \ldots + \text{ moving. money } < 200 \]
IF carpentry. AssignedResource = \( r_1 \) THEN roof. AssignedResource = \( r_1 \)
IF plumbing. AssignedResource = check plumbing. AssignedResource THEN Discount(money, 7)

电 \( \text{electricity installation} \) \text{ BEFORE} \ ceiling
IF Execute(facade painting) THEN Execute(wooden frame installation)

Fig. 5. House construction workflow in WSL.
which is not provided by previous workflow specification languages. As explained in Section 4, we group the resource allocation constraints as hard resource allocation constraints and business rules. Both types of constraints are modeled in this language as follows:

1. Hard resource allocation constraints:
   
i. \( \langle \text{task} \rangle . \text{AssignedResource} = \langle \text{task} \rangle . \text{AssignedResource} \)
   e.g. construct_wall.AssignedResource = construct_ceiling.AssignedResource.
   
ii. \( \langle \text{task} \rangle . \text{AssignedResource} \neq \langle \text{task} \rangle . \text{AssignedResource} \)
   e.g. install_frame.AssignedResource \neq facade.AssignedResource.
   
iii. \( \langle \text{task} \rangle . \text{AssignedResource} = \langle \text{resource} \rangle \)
   e.g. construct_wall.AssignedResource = r_1.
   
iv. \( \langle \text{task} \rangle . \text{AssignedResource} \neq \langle \text{resource} \rangle \)
   e.g. install_frame.AssignedResource \neq r_1.
   
v. \( f(\langle \text{task} \rangle . \langle \text{costterm} \rangle, \ldots, \langle \text{task} \rangle . \langle \text{costterm} \rangle) \langle \text{op} \rangle f'(\langle \text{task} \rangle . \langle \text{costterm} \rangle, \ldots, \langle \text{task} \rangle . \langle \text{costterm} \rangle) \)
   e.g. max((carpentry.Duration + roof.Duration), (elec_inst.Duration + plumbing.Duration)) \leq 7\text{days}
   
vi. \text{IF } X \text{ THEN } Y, \text{ where } X \text{ and } Y \text{ are hard resource constraints. It is also possible that } X \text{ or } Y \text{ (but not both) is a temporal constraint.}
   e.g. \text{IF } gardening.Cost > 50 \text{ THEN } moving.Cost < 80

2. Business rules:
   \( \text{IF } \langle \text{hard constraint} \rangle \text{ THEN } \text{Discount}(\langle \text{cost term} \rangle, \langle \text{amount} \rangle) \)
   e.g. \text{IF } carpentry.assignedresource = roof.assignedresource \text{ THEN } \text{Discount}(\text{totalCost}, 100).
   As explained in Section 4, hard constraint of the constraint may be replaced by a temporal constraint.

5.2.5. Example—house construction

The WSL representation of the house construction workflow is given in Fig. 5. The first part of the WSL representation defines the blocks constituting the workflow. The second part shows the resource allocation constraints and temporal/causeality constraints of Example 2.2 specified in WSL. Fig. 6 presents the resource definitions. Due to space limitation, we give only a part of the resource definitions.

6. Defining workflow using constraint programming

6.1. Overview of constraint programming

Constraint programming is an interdisciplinary area combining computer science and operations research. In constraint programming, a problem is defined declaratively as a set of constraints and solved
by using efficient constraint programming algorithms from mathematics, artificial intelligence and operations research. Thus, constraint programming provides a framework for both defining and solving problems and it is effectively used in many areas such as production planning or scheduling [25,26,37].

Constraint logic programming (CLP) is a branch of constraint programming that combines logic programming and constraint satisfaction techniques. Constraints are integrated into classical logic programming languages (such as Prolog) as new semantic structures and control mechanism are supplemented by a constraint solver to handle the new structures. Jaffar and Maher [23] and Fages [24] gives a more detailed overview on CLP.

Among several domains used in CLP, finite domain is the one that is most suitable for workflow specification and satisfaction under resource allocation constraints. Finite domain for a constraint variable is a finite set of values [23,38,39]. Typically, it is realized as a finite set of integers. This realization sets a limit to the complexity of the constraint satisfaction operations. For finite domain, the basic constraint solving technique is local propagation. Local propagation technique tries to remove the set of inconsistent values from the domain of constraint variables as much as possible [38,40]. However, after local propagation, inconsistent combinations of values may still remain. For this reason, this technique is generally accompanied by enumeration technique, referred as distribution, where the domain is further pruned by a search process for consistent values [28,38,40].

Resources of a workflow are typically a limited set and the cost values of resources are as integer values in a limited range. Thus, the resource information and resource allocation constraints can be modeled as objects in finite domain. In addition to this, execution of the workflow along a time scale can be represented in finite domain and similar to resource allocation constraints, temporal/causality constraints are defined as finite domain objects.

In this work, a multi-paradigm programming language Oz [28,41] is used as constraint programming environment. Oz has logic programming and concurrent programming features as well as its constraint specification and solving capabilities. Oz exhibits logic programming and constraint programming features through its logic variables, disjunctive constructs and programmable search strategies [28,42]. In addition to this, Oz has concurrent programming features, which enable users to dynamically create any number of threads that can interact with each other [43]. Using this feature, workflows with concurrent branches can easily be executed through an Oz-based enactment engine.

### 6.2. Translation from WSL to constraint language (WfToCon)

One of the most important contributions of this paper is the translator (WfToCon) module of the architecture, which is defined in Section 5.1. WfToCon translates workflow and constraint specifications into a constraint program in Oz. In general, any constraint language can be used as target language. We defined a mapping from each block and constraint-type specification in WSL to a constraint structure. Therefore, this translation may also be viewed as the specification of the semantics of WSL structures in terms of set of constraints. By using this mapping, WfToCon produces a constraint set equivalent to WSL specification.

In order to capture the inter- and intra-block dependencies, we defined start, end and duration attributes for blocks, denoting beginning, end and duration of block’s execution.\(^3\) \(\langle\text{block}\rangle.\text{start} + \langle\text{block}\rangle.\text{duration} = \langle\text{block}\rangle.\text{end}\) is a default constraint for each block. As discussed in the previous section, these attributes are defined as finite-domain objects.

In the rest of this section, the mapping from WSL to constraint language is presented. We used a classical mathematical notation for the representation of constraints. This notation is converted to Oz’s constraint specification notation in the implementation.

\(^3\)Since a task is the simplest block, these attributes are defined also for tasks.
6.2.1. Sequential block

Sequential block $P$ with the sub-blocks $b_1, \ldots, b_i, \ldots, b_n$ is defined as

\[ P \text{. start} = b_1 \text{. start} \]
\[ P \text{. end} = b_n \text{. end} \]
\[ \forall i, \ 1 \leq i \leq n - 1, \ b_i \text{. end} \leq b_{i+1} \text{. start} \]

6.2.2. AND block

AND block $P$ with the sub-blocks $b_1, \ldots, b_i, \ldots, b_n$ is defined as

\[ \forall i, \ 1 \leq i \leq n, P \text{. start} \leq b_i \text{. start} \]
\[ \forall i, \ 1 \leq i \leq n, b_i \text{. end} \leq P \text{. end} \]

6.2.3. OR block

OR block $P$ with the sub-blocks $b_1, \ldots, b_i, \ldots, b_n$ is defined as follows:

\[ (\forall i, \ 1 \leq i \leq n, \bigvee (P \text{. start} \leq b_i \text{. start} \land P \text{. end} \geq b_i \text{. end})). \]

In an OR block, at least one sub-block is executed. This is modeled through disjunction. Typically, if a block is executed, its start time is an integer value within the execution time scale. In our representation, a task is executed if its start time is within the execution time range of the block in which it is defined.

6.2.4. XOR block

XOR block $P$ with the sub-blocks $b_1, \ldots, b_i, \ldots, b_n$ is defined as

\[ \forall i, \ 1 \leq i \leq n, \bigvee ((P \text{. start} = b_i \text{. start} \land P \text{. end} = b_i \text{. end}) \land \forall j, \ 1 \leq j \leq n, j \neq i \ (b_j \text{. start} = \text{OutOfRange})). \]

In an XOR block, only one of the sub-blocks is executed. The first part of the constraint shows that a sub-block is chosen for execution and the second part guarantees that once a sub-block is executed, the others are not executed. We represent that a sub-block (or task) is not to be executed by setting a start value for it that is out of the normal execution range (given as outOfRange above). This may be a negative value or a value that is higher than the end of scale. The sub-block is chosen non-deterministically and this is represented by disjunction.

If the start time of a block is set to outOfRange, this should be reflected to all sub-blocks of this block. Hence, the following constraints are defined for subtasks of OR and XOR blocks:

\[ \forall P, P \text{ is OR/XOR block}, \forall Q, Q \in P, P \text{. start} = \text{OutOfRange} \rightarrow Q \text{. start} = \text{OutOfRange}. \]

6.2.5. Iteration block

To represent an iteration block as a set of constraints, the tasks in the block must be renamed for each instance of iteration. These instances are modeled as sub-blocks of a sequential block. In order to determine the number of iterations, a close guess may be possible from the previous executions or we may overestimate this number.

If the iteration will take place $n$ times, then we rename $n$ instances of the iteration block $P$ as $P_1, \ldots, P_n$ and model the iteration as follows:

\[ \forall i, \ 1 \leq i \leq n - 1, (P_i \text{. end} \leq P_{i+1} \text{. start}). \]

In this model, the sub-blocks in $P_i$’s must also be renamed to differentiate the instances. Multiple copies of the constraints involving the iteration block must be defined as many as the number of the iteration. Additionally, iteration block or its sub-blocks referred in these multiple copies must be renamed in consistency with instance naming.
6.2.6. Condition

A precondition for a block \( P \) is defined as

\[
\text{Pre}\_\text{Cond} \_P \rightarrow \langle P \rangle ,
\]

where \( \langle P \rangle \) is the set of constraints defining block \( P \). Post conditions can also be defined similarly. If execution of a block \( Q \) can make a condition true, then this is represented as

\[
\langle Q \rangle \rightarrow \text{Cond} \_Q .
\]

6.2.7. Temporal and causality constraints

In order to represent temporal/causality constraints, as in OR/XOR block definitions, \( outOfRange \) is used as the starting time value in order to denote that the block is not executed. The temporal/causality constraint definitions of WSL are translated into constraint language as follows:

- **Execute** (*) \( t_i \) is represented as \( (t_i, \text{start} \neq outOfRange) \).
- **NotExecute** (*) \( t_i \) is represented as \( (t_i, \text{start} = outOfRange) \).
- \( t_j \) BEFORE \( t_i \) is represented as \( (t_i, \text{end} \leq t_j, \text{start}) \).
- **IF** \( X \) THEN \( Y \) is represented as \( X \rightarrow Y \).

6.2.8. Defining resources

The resource information given in \( WSL \) as

\[
\langle \text{Resource.Name} \rangle \text{CAPABLE.} \_\text{OF} \langle \text{task} \rangle \\
\text{WITH} [(\langle \text{cost}\_\text{term}_1 \rangle, \langle \text{cost}_1 \rangle) \ldots (\langle \text{cost}\_\text{term}_n \rangle, \langle \text{cost}_n \rangle)]
\]

is represented as follows:

\[
\langle \text{task} \rangle.\text{resource} = \langle \text{resource} \rangle \rightarrow \langle \text{task} \rangle.\langle \text{cost}_1 \rangle \ldots \langle \text{cost}_n \rangle
\]

\[
= \langle \text{cost}_1 \rangle, \ldots, \langle \text{task} \rangle.\langle \text{cost}_n \rangle = \langle \text{cost}_n \rangle .
\]

The role–resource associations are converted to task–resource associations. For example, if a role \( r_i \) is associated with a resource \( rsr_j \) as follows:

\[
rsr_j \text{ BELONGS.} \_\text{TO} \ r_i \text{ WITH} [\text{(time, 5), (money, 5)}]
\]

for each task \( t_k \) that is mapped to role \( r_i \),

we define an association between \( rsr_j \) and task \( t_k \) as follows:

\[
t_k.\text{resource} = rsr_j \rightarrow \text{task.time} = 5, \text{task.money} = 5 .
\]

It is possible that some tasks in OR and XOR blocks are never executed. We have to show that no resource selection is done for these tasks and resource allocation cost is 0. For this purpose, we make additional definitions with empty resource, as follows:

\[
\langle \text{task} \rangle.\text{resource} = \text{Empty} \rightarrow \langle \text{task} \rangle.\langle \text{cost}_1 \rangle 0, \ldots, \langle \text{task} \rangle.\langle \text{cost}_n \rangle 0 .
\]

In addition to this, we have the following constraints to show that no resource allocation is performed for tasks with start value \( outOfRange \):

\[
\langle \text{task} \rangle.\text{start} = outOfRange \rightarrow \langle \text{task} \rangle.\text{resource} = \text{Empty} .
\]

6.2.9. Resource allocation constraints

The specification of resource allocation constraints in \( WSL \) and constraint language are similar with minor changes in the notation. For instance,

\[
\langle \text{task}_1 \rangle.\text{AssignedResource} = \langle \text{task}_j \rangle.\text{AssignedResource}
\]
of \( WSL \) is represented as
\[
\langle task_i \rangle .resource = \langle task_j \rangle .resource
\]
in constraint language. As another example, in constraint language, business rules are modeled as follows:
\[
\langle \text{hard constraint} \rangle \rightarrow Discount. \langle \text{cost term} \rangle = \langle \text{value} \rangle.
\]

6.2.10. Example
In this example, we present the constraint language representation of a subworkflow of house construction workflow. We assume that there are two subcontractors (i.e., resources), \( x \) and \( y \) that are capable of performing the tasks of the subworkflow given in Fig. 7 with the financial costs given in Fig. 8. In order to simplify the example, we assume that the execution time is 1 unit for all tasks with any resource. The resource allocation constraint is \( \text{total cost} < 150 \) (which has a smaller budget than the one given Example 2.2, since we use just a sub-workflow in this example). The constraint language representation of the workflow blocks are given in Fig. 9, constraints in Fig. 10 and resource information in Fig. 11. The number of iterations for \text{check plumbing} \) is estimated as 3 and three instances of this task is created. As seen from the example, most of the constraints are defined on attributes of the tasks, not the blocks. The constraints on block attributes can be rewritten in terms of task attributes. This reduces the redundancy in constraint set. Although not given in the example, domain declarations are also a part of the specification.

A feasible solution provided by Oz to the house construction workflow of Fig. 7 is summarized in Fig. 12. The solution contains resource allocations, costs of resource allocations and start times for each task. The content of the solution may be extended or reduced depending on the number of cost dimension attributes

![Diagram](image-url)

Fig. 7. Subworkflow for house construction example.

<table>
<thead>
<tr>
<th>tasks</th>
<th>( x )</th>
<th>( y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>wall</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>elec</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>plumb</td>
<td>20</td>
<td>15</td>
</tr>
<tr>
<td>checkPlumb</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>facadePaint</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>facadeVinyl</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>ceiling</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>fireplace</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>sauna</td>
<td>50</td>
<td>60</td>
</tr>
</tbody>
</table>

Fig. 8. Resource allocation cost table.
**Duration information**

\[
\text{wall.duration} = 1, \text{electricity.duration} = 1, \ldots, \text{sauna.duration} = 1
\]

**Block Definitions**

\[
\text{wall.start} + \text{wall.duration} \leq \text{electricity.start}
\]

\[
\text{electricity.start} + \text{electricity.duration} \leq \text{plumbing.start}
\]

\[
\text{plumbing.start} + \text{plumbing.duration} \leq \text{checkPlumbing1.start}
\]

\[
\text{checkPlumbing1.start} + \text{checkPlumbing1.duration} \leq \text{checkPlumbing2.start}
\]

\[
\text{checkPlumbing2.start} + \text{checkPlumbing2.duration} \leq \text{checkPlumbing3.start}
\]

\[
\text{checkPlumbing3.start} + \text{checkPlumbing3.duration} \leq \text{facade.start}
\]

\[
((\text{facade.start} = \text{facadePaint.start}) \land
\quad \text{facade.end} = \text{facadePaint.start} + \text{facadePaint.duration}) \lor
\quad \text{facadeVinyl.start} = \text{outOfRange}) \lor
\]

\[
((\text{facade.start} = \text{facadeVinyl.start}) \land
\quad \text{facade.end} = \text{facadeVinyl.start} + \text{facadeVinyl.duration}) \land
\quad \text{facadePaint.start} = \text{outOfRange})
\]

\[
\text{wall.start} + \text{wall.duration} \leq \text{ceiling.start}
\]

\[
\text{ceiling.start} + \text{ceiling.duration} \leq \text{fireplaceOrSauna.start}
\]

\[
((\text{fireplaceOrSauna.start} = \text{fireplace.start}) \land
\quad \text{fireplaceOrSauna.end} = \text{fireplace.start} + \text{fireplace.duration}) \lor
\quad \text{fireplaceOrSauna.start} = \text{sauna.start} \land
\quad \text{fireplaceOrSauna.end} = \text{sauna.start} + \text{sauna.duration})
\]

---

**Fig. 9. Blocks of house constraint example.**

---

**Constraints**

\[
\text{wall.cost} + \text{electricity.cost} + \text{plumbing.cost} + \\
\quad \text{checkPlumbing1.cost} + \text{checkPlumbing2.cost} + \text{checkPlumbing3.cost} + \text{ceiling.cost} + \text{facadePaint.cost} + \text{facadeVinyl.cost} + \text{firePlace.cost} + \text{sauna.cost} - \text{discount} = \text{total.cost}
\]

\[
\text{total.cost} < 150
\]

\[
\text{electricity.end} \leq \text{ceiling.start}
\]

\[
\text{plumbing.resource} = \text{check.plumbing1.resource} \land
\quad \text{check.plumbing1.resource} = \text{check.plumbing2.resource} \land
\quad \text{check.plumbing2.resource} = \text{check.plumbing3.resource} \rightarrow \text{discount} = 7
\]

---

**Fig. 10. Constraints for house constraint example.**
Resource definitions:

\[
\begin{align*}
\text{wall.resource} &= x \rightarrow \text{wall.cost} = 10 \\
\text{wall.resource} &= y \rightarrow \text{wall.cost} = 20 \\
\text{electricity.resource} &= x \rightarrow \text{electricity.cost} = 20 \\
\text{electricity.resource} &= y \rightarrow \text{electricity.cost} = 10 \\
\text{plumbing.resource} &= x \rightarrow \text{plumbing.cost} = 20 \\
\text{plumbing.resource} &= y \rightarrow \text{plumbing.cost} = 15 \\
\text{checkPlumbing1.resource} &= x \rightarrow \text{checkPlumbing1.cost} = 15 \\
\text{checkPlumbing1.resource} &= y \rightarrow \text{checkPlumbing1.cost} = 10 \\
\text{checkPlumbing2.resource} &= x \rightarrow \text{checkPlumbing2.cost} = 15 \\
\text{checkPlumbing2.resource} &= y \rightarrow \text{checkPlumbing2.cost} = 10 \\
\text{checkPlumbing3.resource} &= x \rightarrow \text{checkPlumbing3.cost} = 15 \\
\text{checkPlumbing3.resource} &= y \rightarrow \text{checkPlumbing3.cost} = 10 \\
\text{facadePaint.resource} &= x \rightarrow \text{facadePaint.cost} = 20 \\
\text{facadePaint.resource} &= y \rightarrow \text{facadePaint.cost} = 40 \\
\text{facadePaint.resource} &= \text{Empty} \rightarrow \text{facadePaint.cost} = 0 \\
\text{facadeVinyl.resource} &= x \rightarrow \text{facadeVinyl.cost} = 10 \\
\text{facadeVinyl.resource} &= y \rightarrow \text{facadeVinyl.cost} = 20 \\
\text{facadeVinyl.resource} &= \text{Empty} \rightarrow \text{facadeVinyl.cost} = 0 \\
\text{ceiling.resource} &= x \rightarrow \text{ceiling.cost} = 10 \\
\text{ceiling.resource} &= y \rightarrow \text{ceiling.cost} = 15 \\
\text{fireplace.resource} &= x \rightarrow \text{fireplace.cost} = 30 \\
\text{fireplace.resource} &= y \rightarrow \text{fireplace.cost} = 20 \\
\text{fireplace.resource} &= \text{Empty} \rightarrow \text{fireplace.cost} = 0 \\
\text{sauna.resource} &= x \rightarrow \text{sauna.cost} = 50 \\
\text{sauna.resource} &= y \rightarrow \text{sauna.cost} = 60 \\
\text{sauna.resource} &= \text{Empty} \rightarrow \text{sauna.cost} = 0 \\
\text{facadePaint.start} &= \text{out OfRange} \rightarrow \text{facadePaint.resource} = \text{Empty} \\
\text{facadeVinyl.start} &= \text{out OfRange} \rightarrow \text{facadeVinyl.resource} = \text{Empty} \\
\text{fireplace.start} &= \text{out OfRange} \rightarrow \text{fireplace.resource} = \text{Empty} \\
\text{sauna.start} &= \text{out OfRange} \rightarrow \text{sauna.resource} = \text{Empty}
\end{align*}
\]

Fig. 11. Resource definitions for house construction example.

of resource allocation. Oz also computes the total budget and amount of discount. For this example, the total budget of the constructed schedule is 133 with a discount of 7.

In order to activate the workflow according to the provided results, the start times can be sent to the enactment engine and resource assignment information is sent to the resource manager. The enactment engine activates the tasks according to the schedule. Just as a task is about to be activated, the enactment engine notifies the resource manager and therefore resource manager allocates the scheduled resource for
the task. As an alternative, enactment engine may only consider ordering with considering the exact start times for tasks. Although our architecture is designed to provide feasible solution, it can be tuned to find all solutions or the optimal solution. However, as expected, in this case execution time will increase since the whole search space needs to be explored.

6.3. Experiments and efficiency issues

Although workflow processes are generally long-term activities, finding the schedule prior to the enactment of the process must be efficient. For most of the constraint problems, the aim is to find the optimal solution which requires search throughout the whole search space. However, we aimed to obtain a feasible solution to reduce the complexity of the problem when compared to conventional scheduling problems that requires optimal solution.

In order to check the feasibility of constraint programming approach for workflow scheduling under resource allocation constraints and to see the effect of different workflow structures on the execution times of constraint resolution process, we have conducted some experiments. In the first set of experiments, we tested the effect of increase in constraint set on four types of workflows that include the same fixed number of tasks. The first workflow consists of a single sequential block, second one has a single AND block, third one has a single OR block and fourth one consists of a single XOR block. We have not used iteration blocks, since it is modeled using sequential blocks. The experiment is conducted with constraints sets including 2, 4, 8, 16 and 32 constraints. The constraint sets include a cost constraint on the total cost and

<table>
<thead>
<tr>
<th>tasks</th>
<th>resource</th>
<th>cost</th>
<th>start</th>
</tr>
</thead>
<tbody>
<tr>
<td>wall</td>
<td>x</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>elec</td>
<td>y</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>plumb</td>
<td>y</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>checkPlumb1</td>
<td>y</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>checkPlumb2</td>
<td>y</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>checkPlumb3</td>
<td>y</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>facadePaint</td>
<td>empty</td>
<td>0</td>
<td>do not execute</td>
</tr>
<tr>
<td>facadeVinyl</td>
<td>x</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>ceiling</td>
<td>y</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>fireplace</td>
<td>empty</td>
<td>0</td>
<td>do not execute</td>
</tr>
<tr>
<td>sauna</td>
<td>x</td>
<td>50</td>
<td>3</td>
</tr>
</tbody>
</table>

Fig. 12. Schedule for house construction example.
control constraints on equality/inequality of resources to each other and to given scalar values. The number of second type of constraints is increased for each test case. The result of this first set of experiments is presented in Fig. 13. The main results of this experiment are as follows:

- The increase in the number of constraints cause a little increase in the execution time of the constraint resolution process.
- The amount of increase is nearly the same for all four types of workflows.
- The ratio between the constraint resolution execution time of XOR/OR blocks vs. SEQ/AND blocks is almost constant and it is related to the number of tasks.

The second set of experiments is conducted to check the effect of increase in the number of tasks. Again we have used four types of workflows including only a sequential block, AND block, OR block and XOR block, respectively. The experiment is repeated with exponentially increasing number of tasks in blocks—using 2, 4, 8, 16 and 32 tasks—under the same constraint set with five constraints (a cost constraint on the total cost, two constraints on the equality/inequality of the costs of individual tasks to given scalar values, two constraints on the equality/inequality of the resources of the individual tasks). The result is given in Fig. 14. The main results of this experiment are as follows:

- The increase in the number of tasks leads to more increase in constraint resolution time compared to the increase in the size of the constraint set. This result is expected due to the fact that each task is represented with a constraint variable and therefore its associated domain and each new task means a new domain to be reduced.
- The increase in the execution time of the constraint resolution process for workflows consisting of OR block and XOR block is higher compared to workflows consisting of SEQ and AND blocks. This is due to the disjunction in the semantics of OR and XOR blocks.
- When we compare OR and XOR blocks, the execution time of the constraint resolution for XOR block is slightly longer than that of OR block. The reason for this result is that OR blocks are easier to satisfy. The same set of constraints may fail to find a solution for XOR block.
- The increase in of the constraint resolution time is all polynomial.

Usually, workflows consist of mostly sequential and AND blocks. XOR and OR blocks appear only occasionally and thus the constraint resolution time is expected to be below the OR/XOR curve. Fig. 15
presents the constraint resolution time for some examples from the literature. We have added resource definitions and resource allocation constraints (a cost constraint on total cost, and three control constraints on equality/inequality of resources for some tasks) in the specification of telephone line installation [1] and hospital workflow [44] examples, which did not include any resource information or resource allocation constraints originally. The workflow structure of the first and second examples were already given in Section 2. Fig. 16(a) and (b) show the block structure for the third and fourth examples of Fig. 15, respectively.
7. Related work

This work is related to workflow scheduling, resource management in workflows and constraint programming. Below we compare our work with previous works on each one of these areas.

**Workflow scheduling:** Our paper is basically on workflow scheduling problem. However, the difference between our work and other workflow scheduling works is that most of the previous work on this problem deals with *temporal/causality* constraints, but our work also allows *resource allocation constraints*. The study by Senkul et al. [27] is the only previous workflow scheduling work that deals with resource allocation constraints. In [27], a formal model for the problem is presented. In this work, we demonstrate the practicality of the ideas presented in [27], by a system that has a script language with constructs to model resources and resource allocation allocation constraints and that uses constraint solver as the core of the scheduler. Davulcu et al. [22] discusses about constraints that involve execution cost of tasks and that are used for modeling and reasoning of virtual enterprises modeled as workflows. These constraints may be seen as early version for resource allocation constraints. Different approaches have been used in order to schedule workflows under temporal/causality constraints. In [7–9], temporal logic and specialized algebras are used for scheduling. The study of Davulcu et al. [15] and Bonner [16] are based on CTR and [12] presents a scheduling method based on an action description logic. The approach followed in [14] is based on situation calculus and concurrent logic programming language. The study by Koksal et al. [13] is another work that models workflows by using action description logic in order to obtain schedules. In [13], workflow blocks and available resource information are translated into action description logic C. Specification and satisfaction of resource allocation constraints are not discussed. Since the action description language C does not have a proof theory, scheduling of the workflow specification is obtained from SAT solvers. Therefore, this approach is not efficient beyond workflows with simple structures such as sequential and AND blocks including few tasks. Besides these logic-based approaches, Petri nets are also used for workflow scheduling [10,11]. There are few works on time management in workflows, which is represented as time cost in our framework. In [45,46] execution time is considered for deadline violation issues independent from resource allocation.

**Resource management in workflow scheduling:** Resource management has been recognized as an important issue in a WFMS [2,18]. However, most of the works in this area have focused on modeling of resources with little attention devoted to scheduling. In [31], a framework for the representation of resources and an organizational model for workflow system is given. Du et al. [32] describes an architecture of a workflow resource manager that can integrate external resources. Huang and Shan [47] 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. Another work on policy management is [30]. While such policies can serve as guidelines on how resources can be allocated during execution, they do not constitute a scheduling algorithm. Our approach considers resource allocation and scheduling together and can be considered orthogonal to these works. The proposed system can be extended with additional policy rules and policy manager to cooperate with our constraint-solver-based scheduler.

**Constraint programming:** Operations research (OR) has developed a number of successful algorithms for solving many scheduling problems [25,26]. Among them, job-shop scheduling is the most relevant to workflow scheduling. Constraint logic programming, which integrates logic programming and OR techniques, has also been successfully used to deal with job-shop scheduling problems [37,48]. 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. For example, iterative blocks of workflows do not exists in job-shop problem. Furthermore, in this work, the emphasis is on developing constraint-programming-based framework for workflow specification and scheduling. For this reason, the works mentioned above are orthogonal to our work in such a way that some of the constraint solving techniques proposed may be adapted for workflows as well.
8. Conclusion

This paper presents an architecture to model and schedule workflows with resource allocation constraints as well as with the traditional temporal/causality constraints. Current approaches for scheduling tasks in a workflow provide no mechanism to reason about the relative costs of schedules. Scheduling under resource allocation constraints provide decisions on the assignment of resources to tasks, as well as correct execution sequence of tasks. Hence, workflow schedules with desired execution cost level can be obtained.

We use constraint programming to schedule workflows with resource allocation constraints. Workflow specification together with resource information and constraints, including both resource allocation and temporal/causality constraints, are translated to finite-domain constraints. For the implementation, constraint programming language Oz is used.

The main contribution of this work is the proposed architecture which provides a specification language that can model resource information and resource allocation constraints, and a scheduler model that incorporates a constraint solver in order to find proper resource assignments. Although workflow processes are generally long-term activities, finding the schedule prior to the enactment of the process must be efficient. For most of the constraint problems, the aim is to find the optimal solution which requires search throughout the whole search space. However, our aim is to obtain a feasible solution to reduce the complexity of the problem. As a future work, the architecture can be extended with a special-purpose constraint solver in order to find solutions for our framework more efficiently.

In our work scheduling of single workflow instances are handled. Pre-scheduling of concurrent workflows is also very interesting problem. For this purpose, first, the issue on how to specify the starting times of workflows, must be resolved. Then, our approach can be extended to determine the schedule of concurrent workflows with given starting times by maintaining additional temporal information in order to make the decisions for starting the executions of workflows. The schedule obtained must satisfy all the constraints and obey given starting times of concurrently executing workflows.

References


