The Smart Workflow Foundation

MSR-TR-2006-114

Youssef Hamadi* Claude-Guy Quimper†

November 13, 2006

Abstract

This report presents the *Smart Workflow Foundation* (SWF), a new architecture which adds constraint solving capabilities to workflow engines. Thanks to that extension, workflow definitions are freed from low level implementation details and can benefit from smart and robust resource allocation. This architecture represents a radical change over classical engines where the execution of each task or procedural step is either pre-assigned to some entity, *e.g.,*employee, either computed by a former task. The proposed system uses abstract workflow definitions combined with a characterizing of the resources to efficiently match tasks requirements to resources abilities and availabilities. During this process, the possible future steps of a workflow are considered. This mimics the capabilities of human beings, able to infer the consequences of a decision against some foreseeable future. The system is built on top of Windows Workflow Foundation and evaluated through several simulations. Various extensions are also presented in order to improve the scope of the reasoning, automatically drive the execution flow from the result of high level optimization problems, and use the newly proposed abstraction to solve capacity planning scenarios.

1 Introduction

Business workflows organize the activity of an enterprise through the controlled execution of well defined dynamic processes. They combine low level activities through some partial order and control the execution of these activities through high level decisions and constraints. A workflow can be seen as a data-flow and is therefore very similar to a program [ACW06]. For example, a company could use an *interview-process* to ensure that a candidate is handled consistently through a pre-defined interview process. The workflow engine would ensure that each interviewer used the correct online form and successfully gave their personal

*Microsoft Research Ltd., 7 J J Thomson Avenue Cambridge CB3 0FB, United Kingdom, youssefh@microsoft.com

†School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, N2L 3G1, cquimper@math.uwaterloo.ca

feed-back before allowing the process to proceed to the next step and decide to make an offer to the candidate.

More precisely, a *business workflow* describes the tasks, procedural steps, organizations or people involved, required input/output information, and tools needed for each step of a business process.

Business workflows are executed and controlled by *workflow engines* or *workflow schedulers*. These software components take as input workflow definitions and controls their execution through the respect of the specified steps and rules. Their main role is to determine whether the process is ready to move to the next step.

This report presents the *Smart Workflow Foundation* (SWF), a new architecture which adds constraint solving capabilities to workflow engines. Thanks to that extension, workflow definitions are freed from low level implementation details and can automatically benefit from smart and robust resource allocation. This architecture represents a radical change over classical engines where the execution of each task or procedural step is either pre-assigned to some entity, *e.g.,*employee, either computed by a former task. At contrary, SWF uses abstract workflow definitions combined with a characterizing of the resources to efficiently match tasks requirements to resources abilities and availabilities.

In the following, section two presents background material related to Constraint Programming (CP) and to the Windows Workflow Foundation. Section three, describes the general architecture of the SWF. Section four, presents the modeling of tasks and resources. Section five presents the CSP modeling for the resource allocation problem. Section six describes the solving process, and section seven presents the results of our experimental evaluation. Section eight presents possible extensions of the architecture. Finally, before the general conclusion presented in section ten, section 9 describes related works.

2 Background

2.1 Constraint Programming

In the Constraint Programming (CP) or Constraint Satisfaction Problem (CSP) formalism, a combinatorial problem is expressed by a set of constraints applied to its decision variables. More precisely, decision variables are defined with their possible range of values and constraints are applied to these variables to restrict their ranges. The space where variables take their values distinguishes between different flavor of CP (Boolean, Integer, Sets, etc.). A solution is made by a consistent assignment of the variables *i.e.,*a state where each variable gets a value from its range without violating any constraint. Practically, constraint solvers are used to compute solutions (often against some optimality criterion, *a.k.a.*soft-constraints).

2.2 The Windows Workflow Foundation

The Windows Workflow Foundation (WWF) is a Microsoft technology for defining, executing, and managing business processes or workflows [ACW06]. This technology is part of the .NET framework and can be hosted in any CLR application. It has been natively embedded in Windows Vista and back-ported to previous versions of the OS.

In the WWF, a workflow is a collection of tasks structured with connectors allowing their sequential, parallel, conditional, or repetitive execution. The Windows Workflow Foundation scheduler manages the state of each active workflow and launches the tasks according to the structure of the process. A task can either be a computer program or an action executed by an external agent, *e.g.,*employee. Tasks can take seconds or days to be executed depending on their nature.

3 General Architecture

The architecture that we are proposing (see Figure 1) extends the initial work presented in [Ham03a]. It has four main components: the windows workflow foundations, the constraint solver, the resource database, and the policy manager.

Windows Workflow Foundation The WWF allows the efficient design of business processes with workflows (see [ACW06]). In this system, each workflow can be represented as a task which has the possibility to store dedicated information through programmatically defined properties. For example, if a task is assigned to a specific person, a property of the workflow can store the name of that person. Our SWF architecture heavily uses this possibility to integrate decision variables related to the smart allocation process.

Resource Database The resource database contains the information about the resources. This includes the skills of each resource, the tasks that are assigned to the resource, the agenda, and the geographic location.

Constraint Solver See section 2.

Policy Manager The policy manager looks after preferences on the resource allocations. It allows, for instance, to favor resource allocations involving some skill refreshing, leading to a fair distribution of the workload over the employees, or simply optimizing the use of the resources to minimize the make-span of each workflows. The policy manager gives priorities to some preferences based on a weighting system.

We propose to create a CSP whose solution space is equivalent to every possible execution of the workflows. Additional soft-constraints ensure that the resource allocation satisfy the policies. The CSP is created from three sources

of information: the workflow properties, the resource database, and the policies selected by the policy manager.

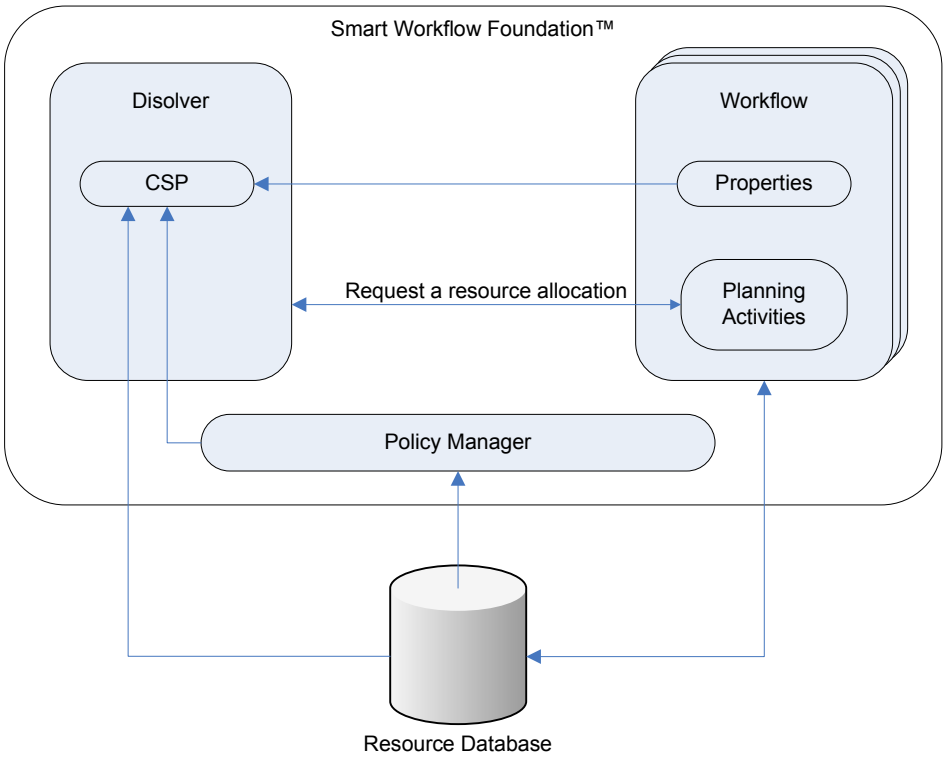

Figure 1: Architecture of the Smart Workflow Foundation.

A task should always be assigned to a resource before being executed. Before executing a task, the WWF should check if the task's resource is allocated. If it is not the case, a CSP is generated on the fly based on the current states of the workflows, the availability of the resources, and the resource allocation policies. The solver finds the best resource allocation for the task based on the policies and assign the task to this resource.

When planning an activity, the system selects a resource based on the current workload of each resource and based on the future actions that require to be planned. Since the workflows might be very long and some activity might not be visited before a long while, the planning only takes into account the activity within a given horizon. This horizon is the number of tasks we look ahead in order to assign a resource to the current task.

4 Workflow Model

4.1 Information about Tasks

We assume that for every task WT in the workflow, the information depicted in Table 1 is available.

Variable Name	Description
$WT.ProcTime$	Expected processing time.
$WT.StartTime$	Starting time (unassigned if the task has not been started)
$WT.EndTime$	Ending time (unassigned if the task has not been completed)
$WT.Skills$	A skill vector indicating, for each skill, the required level to accomplish the task.
$WT.Done$	True if the task is completed, false otherwise.
$WT.Available$	True if the task might eventually be executed, false otherwise.
$WT.Resource$	Resource used to accomplish the task. This property might be unassigned if the task has not been attributed to a resource yet but must be assigned before the execution of the task.

Table 1: Information about tasks stored in each workflow.

4.2 Information about Resources

For each resource R in the database, one can retrieve a skill vector $R.Skills$. Each component of this vector indicates the skill level for a specific skill. For instance, the skill vector of a computer consultant could look like the following one.

.NET	SQL	C++	Networking	Billing
3	3	1	2	0

In addition to the skill levels, a resource R has an agenda of tasks that have been assigned to R. We denote these tasks with $R.Tasks$.

Additional information about resources can be mentioned in the database. For instance, the geographical position $R.Position$ of each employee can be relevant (cf. [DHB05]).

Variable Name	Description	Initial Domain
$T.StartTime$	Estimated starting time	$\{WT.StartTime\}$ if $WT.StartTime$ is assigned. $[0, \infty]$ otherwise.
$T.EndTime$	Estimated ending time	$\{WT.EndTime\}$ if $WT.EndTime$ is assigned, $[0, \infty]$ otherwise.
$T.Available$	1 if the task might eventually be executed, 0 otherwise	$\{0\}$ if not $WT.Available$, $\{1\}$ if $WT.Done$, $\{0, 1\}$ otherwise.
$T.Resource$	Resource that will accomplish the task.	$\{WT.Resource\}$ if $WT.Resource$ is assigned, $\{R \mid R.Skills \geq WT.Skills\} \cup \{Null\}$ otherwise.

4.3 Information about the Workflows

Using properties, it is possible to store information about the structure of the workflows. For instance, in an *If-Else* statement, it is possible to store the probability that the workflow branches on the *if* statement and therefore, the probability that it branches on the *else* statement. These probabilities can even be computed from the history of past executions of the workflow saved in the WWF database. The probabilities are used to better predict the execution of the workflow. If the probabilities are unknown, a uniform distribution over the different choices can always be used.

5 CSP Model

Following [Ham03a] and [ST05], we present a CSP model whose solution space corresponds to all possible walks through the workflows. The solution space is given by hard constraints on which we add soft constraints for optimization purposes. These soft constraints, when violated, only deteriorate the objective value. The feasibility of the solution is not compromised.

5.1 Variables

For every workflow task WT, we declare a task T in the CSP whose members are the following constrained variables.

Figure 2: Single activity

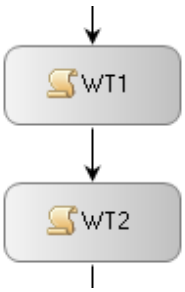

Figure 3: Sequential Activities

5.2 Hard Constraints

5.2.1 Structural Constraints

We present in this section hard constraints based on the structure of the workflow. For every single activity (see Figure 2), we have the following constraint.

$$T_1.EndTime = T_1.StartTime + T_1.Available \times WT_1.ProcTime \quad (1)$$

For any two activities forming a sequence (see Figure 3), we have the following constraints.

$$T_1.Available = T_2.Available \quad (2)$$
$$T_2.StartTime \geq T_1.EndTime \quad (3)$$

For activities executed in parallel (see Figure 4, we have the following constraints.

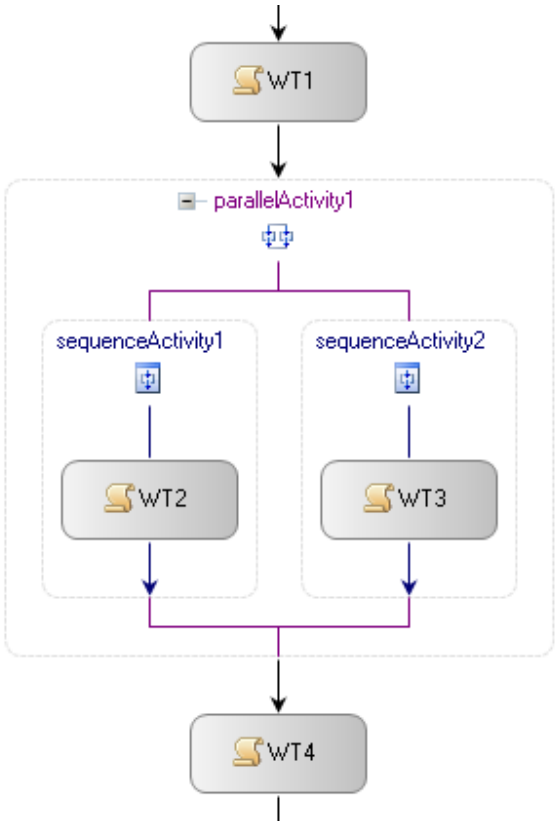

Figure 4: Parallel Activities

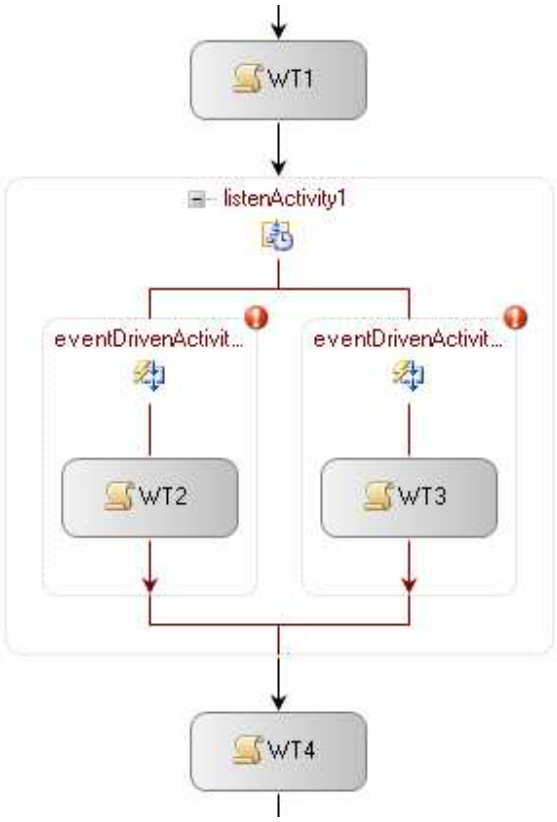

Figure 5: *Listen-Activity* block

$$T_1.Available = T_2.Available = T_3.Available = T_4.Available \quad (4)$$

$$T_2.StartTime \geq T_1.EndTime \quad (5)$$

$$T_3.StartTime \geq T_1.EndTime \quad (6)$$

$$T_4.StartTime \geq max(T_2.EndTime, T_3.EndTime) \quad (7)$$

A workflow might have to branch on a specific activity depending on the event it receives. This is modeled with a *Listen-Activity* block in the Windows Workflow Foundation (see Figure 5). The following constraints apply to the activities in this block.

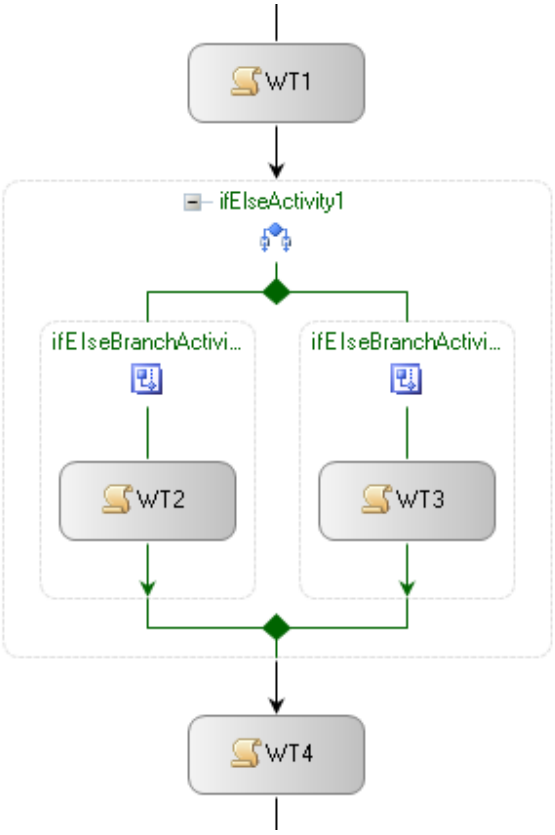

Figure 6: If-else statement block

$$T_1.Available = T_2.Available \quad + \quad T_3.Available = T_4.Available \tag{8}$$

$$T_2.StartTime \quad \geq \quad T_1.EndTime \tag{9}$$

$$T_3.StartTime \quad \geq \quad T_1.EndTime \tag{10}$$

$$T_4.StartTime \quad \geq \quad max(T_2.EndTime, T_3.EndTime) \tag{11}$$

A workflow can also branch according to an *if* statement (see Figure 6 based on a given condition C. The following constraints apply.

$$T_1.Available = T_2.Available \quad + \quad T_3.Available = T_4.Available \tag{12}$$
$$T_2.StartTime \quad \geq \quad T_1.EndTime \tag{13}$$
$$T_3.StartTime \quad \geq \quad T_1.EndTime \tag{14}$$
$$T_4.StartTime \quad \geq \quad max(T_2.EndTime, T_3.EndTime) \tag{15}$$
$$C \quad \iff \quad T_2.Available = 1 \tag{16}$$

The WWF supports composite activities. These activities are built from other activities that form a sub-workflow. We construct the CSP by replacing all composite activities by a decomposition into atomic activities. If a composite activity contains other composite activities, we construct the CSP model by recursively replacing composite activities by atomic activities.

The WWF also supports loops (see Figure 7). The *while* loop tests a condition before executing a sub-workflow and keeps executing this sub-workflow until the condition becomes false. The number of times the loop will execute is unknown but one has to be ready to this eventuality.

5.2.2 Resource Constraints

We present some constraints that model the use of the resources. Notice that according to the initial domain of $T_i.Resource$, only the resources with the proper skills can be affected to a task. There is also a special resource called the $Null$ resource. The resource is allocated to tasks that are not executed. The following constraint models the use of the $Null$ resource.

$$T_i.Resource = Null \iff T_i.Available = 0 \tag{17}$$

When sharing the same resource, two tasks cannot be executed at the same time. This is modeled with the following constraint[1].

$$\begin{aligned} T_i.Resource = T_j.Resource \longrightarrow & T_i.EndTime \leq T_j.StartTime \\ & \vee T_j.EndTime \leq T_i.StartTime \end{aligned} \tag{18}$$

We assume that a list of tasks was previously assigned to each resource. This list is denoted by $R.Tasks$. We assume that each resource executes the tasks using a FIFO policy (first in first out). Therefore, if a task is assigned to a resource R, the task will not be executed until all other tasks in $R.Tasks$ are completed. This is expressed using the following constraint. Notice that in this constraint, $T_i.Resource$ and $T_i.StartTime$ are the two only variables. All other terms are constants.

[1] This will not preclude a human resource to balance its time between multiple assignation and this constraint is only used to report the cumulative use of the resources.

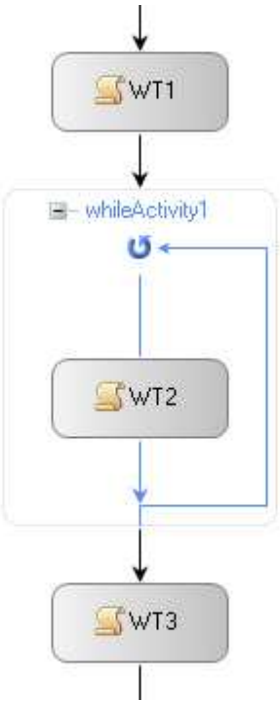

Figure 7: While loop block

$$T_i.Resource = R \longrightarrow T_i.StartTime \geq \sum_{T_j \in R.Tasks} T_j.ProcTime + CurrentTime - \min_{T_j \in R.Tasks} T_j.StartTime \tag{19}$$

5.3 Soft Constraints

In this section, we present constraints expressing preferences on the solution we would like to obtain. These constraints generally map a property of the solution to an integer variable on which we try to minimize (or maximize) the value.

Remark that our modeling directly filters-out non-properly qualified resources (see section 5).

5.3.1 Distributing the Workload

A good resource allocation solution spreads the workload between the different resources. For instance, we want to avoid overloading a resource A while resource B is idle. We define the workload $W(R)$ of a resource R to be the processing time of the tasks assigned to this resource. More formally, we have.

$$W(R) = \sum_{T_i \in R.Tasks} T_i.ProcTime + \sum_{T_i.Resource=R} T_i.ProcTime \tag{20}$$

Two different techniques can be used to spread the workload over the resources. The simplest one is to minimize the maximum workload. We therefore solve the following optimization problem.

$$\min \quad M \tag{21}$$

$$M \quad \geq \quad W(R_i) \ \ \forall R_i \tag{22}$$

This solution is simple as it only involves standard binary constraints. Unfortunately, the workload vectors for three resources $[10, 8, 6]$ and $[10, 7, 7]$ are equivalent since the maximum workload is 10 in both cases. Clearly, the vector $[10, 7, 7]$ is a better solution since it better spreads the workload over the second and the third resource.

Pesant and Régin [PR05] solved this issue by introducing the spread constraint. The expression $\textsc{Spread}([X_1, \ldots, X_n], E, \sigma)$ is satisfied if E is the mean and σ the standard deviation of the sample $X_1, \ldots, X_n$. The workload can be spread over the resources using the following constraints.

$$\min \sigma \tag{23}$$

$$\textsc{Spread}([W(R_1), \ldots, W(R_n)], E, \sigma) \tag{24}$$

$$0 \leq E < \infty \tag{25}$$

We showed two different solutions for distributing the workload over the different resources. Many other solutions might exist. The architecture we present in this document is flexible enough to support new or enhanced models that can better address the needs of a particular organization.

5.3.2 Skill Refreshing

Skill refreshing consists of assigning tasks to resources that have not used a required skill for a *long* time [DHB05]. We show how to compute the resource allocation that maximizes skill refreshing.

Let $f(R, T)$ be a function that returns the skill refreshment gain if task T is assigned to resource R. The total skill refreshing is represented by S which we want to maximize.

$$\max \quad S \tag{26}$$

$$S \quad = \quad \sum_{T_i} f(T_i.Resource, T_i) \tag{27}$$

5.3.3 Avoiding Over-qualified Allocations

A resource must satisfy the required skills in order to accomplish a task. Although, it is undesirable to assign an over-qualified resource to a task. It is a better solution to keep this resource available for more demanding tasks. We define a variable Q evaluating the degree of over-qualified allocations in an assignment. We want to minimize Q.

$$\min Q \tag{28}$$

$$Q \quad = \quad \sum_{T} \sum_{i} T.Resource.Skills[i] - T.Skills[i] \tag{29}$$

5.3.4 Other Policies

Our architecture can handle many other policies. For instance, one might want to minimize the traveled distance of a team of consultants that need to move to accomplish tasks (*cf.*[DHB05]). This could be done by affecting a start-up cost between each pair (Task, Resource). In this example, we simply want to minimize the sum of the start-up costs for every pair of tasks and resources.

All policies can be encoded with soft constraints that map the quality of a solution to a variable called the *cost variable*. We find the best resource allocation subject to multiple policies by minimizing (maximizing) a weighted sum over all cost variables. The user provides these weights dynamically according to the importance given to each policy.

6 Solving the Resource Allocation Problem

We show how to use the model developed in Section 5 to solve the resource allocation problem in workflows. Workflow optimization is a complex problem. It might involve many tasks to schedule with multiple resources. Moreover, the processing time given for each task is only an estimate and therefore scheduling on a long term basis becomes inaccurate. The number of tasks to schedule and the inaccuracy for long term prediction is the first challenge we need to address.

Uncertainty in workflows represents the second challenge. Some activities are conditional to events that cannot be predicted and therefore prevent to derive any precise schedule. It is the case for the *Listen-Activity* blocks, *if-else* statements, and *while* loops. We cannot predict which event will occur first, if the condition will be true or not, or how many times the while loop will be executed. We will show how one can find the best resource allocation despite this uncertainty.

6.1 Horizon

In order to reduce the combinatorial search space, we propose to reduce the total number of tasks by considering a horizon. The tasks beyond a given horizon h from the tasks that are currently being executed are temporarily ignored. Their corresponding variables are not included in the CSP.

6.2 Scenarios

There exist different ways to visit a workflow. For instance, there are two ways to walk through a *if-else* statement: by visiting the *if* branch or the *else* branch. Consider the binary vector $S = [T_1.Available, \ldots, T_n.Available]$. Any such binary vector that satisfies the structural constraints represent a valid walk in the workflow. We call these walks *scenarios*.

Scenarios depend on branching activities: the *Listen-Activity* blocks, the *If-Else* statements, and the loops. We assign a probability on each of these activity branches. For instance, in the case of an *If-Else* statement, we assign a probability p that the condition is true and therefore a probability $1-p$ that the condition is false. Based on these probabilities, we can compute the probability $p(S)$ that a scenario S occurs.

Assume, without loss of generality, that we want to find the best resource allocation for task T_1. Let C_i^j be the cost of the best solution for scenario S_i such that $T_1.Resource = R_j$. We then allocate T_1 to the resource R_j that minimizes the following expression.

$$\sum_{S_i} p(S_i) C_i^j \tag{30}$$

Notice that this solution implies to solve $s \times r$ different CSPs where s is the numbers of scenarios and r the number of resources available for task T_1.

7 Evaluation

7.1 Methodology

We generated 5 random workflows, each having ten tasks. We used different control structures to connect these tasks together. Some workflows are highly probabilistic with many nested *if-else* statements. Other workflows execute many tasks in parallel. Other workflows have *while* loops that increase or decrease the number of resource allocations during a workflow execution.

The duration of each task is precomputed for each instance. A duration is assigned to each combination of a resource and a task. We therefore model the case where some resources require more time to execute a task than others. Each resource is given two or three skills out of four. Each task requires one or two skills to be executed. The skills of the resources remain the same throughout the simulation while the skills associated to each task vary for each instance of a workflow.

Each branch in an *if-else* statement is associated to a probability of branching to this branch. During the execution of the workflow, the system generates pseudo-random numbers to branch in the workflow according to the probabilities. Note that each instance can have different probabilities. The same principle applies for the *while* loops. Each loop has a probability p that the condition is satisfied. The simulator generates pseudo-random numbers before each iteration in order to test if the loop should be executed or not.

A simulation consists of the execution of 10 workflows whose starting time is uniformly spread over an interval of T seconds. Throughout the simulation, the scheduler uses a predefined resource allocation policy. We compare different metrics against the allocation policies and the length of interval T. We label the random resource allocation policy which allocates one compatible resource to a task at random with Rd, the minimization of the largest resource workload with MSW, the minimization of the resource workload variance with MV, the skill refreshing policy with SR, and the overskill minimization policy with OS.

The Disolver constraint solver was used in all our tests [Ham03b].

7.2 Results

Number of Resource Allocations We present in this table the number of resource allocations that occurred in each simulation. This number should roughly be the same for all simulations having the same time span T but can vary since the *if-else* statements as well as the *while* loops are non-deterministic.

Average Scheduling Time We report the average time for allocating a resource using different resource allocation policies.

As expected, the random resource allocation policy is the fastest since it only requires to generate a pseudo-random number associated to a resource. The minimization of the resource workload variance (MV) is the policy requiring, by far, the most computation time.

T	Rd	MSW	MV	SR	OS
40	74	78	70	63	69
60	69	45	84	61	57
80	69	86	74	68	71
120	71	75	81	69	69

Table 2: Number of tasks execution during the simulation

T	Rd	MSW	MV	SR	OS
40	0	49	285	72	65
60	0	105	255	112	132
80	0	179	396	71	75
120	0	60	396	76	70

Table 3: Average scheduling time for finding a resource allocation. All times are in milliseconds.

Workflow Execution Time The workflow execution time is the total amount of time spent in executing a task. Since many tasks can be executed at the same time, the workflow execution time might be greater than the workflow completion time. The policies can affect the workflow execution time by choosing resources that are faster or slower at executing a particular task.

T	Rd	MSW	MV	SR	OS
40	37.2	25.6	23.3	21.4	48.9
60	22.0	12.8	24.5	18.0	19.2
80	25.8	28.5	23.5	21.7	29.5
120	20.9	19.5	26.9	22.5	23.1

Table 4: Workflow execution time in seconds.

The different policies are comparable in this case since none of them aim at optimizing the workflow execution time.

Workflow Idle Time Resources execute tasks using a FIFO policy. While a task is in a resource's queue, we say that this task is *idle* since it is waiting for its resource to be freed. We compute the average idle time of the workflows.

One would expect the minimization of the largest resource workload policy (MSW) or the minimization of the resource workload variance policy (MV) to be the best policies in this context since they spread the workload among the resources. Surprisingly, the skill refreshing policy (SR) is the policy offering the best performances in terms of idle time.

T	Rd	MSW	MV	SR	OS
40	16.5	6.8	6.9	5.9	30.7
60	3.2	1.0	2.4	1.3	3.2
80	9.2	5.9	4.5	1.5	12.9
120	2.2	1.2	4.0	1.8	3.5

Table 5: Workflow idle time in seconds.

Skill Freshness We compare the freshness of the skills at the end of the simulation, *i.e.,*the average number of seconds since the last time the skill has been used. The smaller the values are, the fresher the skills are.

T	Rd	MSW	MV	SR	OS
40	22.2	10.0	11.2	12.2	48.0
60	20.0	19.7	34.0	17.2	53.5
80	28.0	29.5	25.0	20.6	41.0
120	12.0	13.3	17.4	11.2	52.0

Table 6: Average time in seconds since the last time the skill has been used.

As expected the skill refreshing policy (SR) is the most efficient. Notice that the overskill minimization policy gives the worst results. Indeed, the overskill minimization policy tends to assign the same resources to the same tasks and prevents the refreshment of the skills.

Average Workload Let $W(R,t)$ be the workload of resource R at time t *i.e.,*the amount of work in the to-do list of the resource R expressed in seconds. Let E be the completion time of the simulation. The following formula defines the average workload A.

$$A = \frac{1}{|R|E} \sum_{R} \int_{0}^{E} W(R,t)\, dt \tag{31}$$

T	Rd	MSW	MV	SR	OS
40	1.38	0.78	0.79	0.87	1.34
60	0.60	0.39	0.43	0.56	0.40
80	0.98	0.59	0.45	0.56	0.77
120	0.41	0.36	0.35	0.46	0.41

Table 7: Average workload in seconds.

Both the minimization of the largest resource workload policy (MSW) and the minimization of the resource workload variance policy (MV) perform well

as they tend to minimize the average workload. Moreover, the overskill minimization policy (OS) seems comparable to the random allocation policy.

Variance of Average Workloads Let $W(R,t)$ be the workload of resource R at time t *i.e.,*the amount of work in the to-do list of the resource R expressed in seconds. The average workload of a resource R over a period of time E is given by the following expression.

$$AverageWorkload(R) \quad = \quad \frac{1}{E}\int_0^E W(R,t)\,dt \tag{32}$$

We compare the variance of $AverageWorkload(R)$ among all resources. A high variance indicates some imbalance in the resource allocations *i.e.,*some resources had a peek period more intense than others.

T	Rd	MSW	MV	SR	OS
40	2.23	0.11	0.12	0.18	4.70
60	0.09	0.03	0.02	0.07	0.09
80	0.96	0.06	0.06	0.00	0.92
120	0.02	0.00	0.02	0.03	0.12

Table 8: Variance of average workload.

Clearly the minimization of the largest resource workload policy (MSW) and the minimization of the resource workload variance policy (MV) perform the best as they aim at minimizing the variance. However, notice that the MV policy is not significantly better than the MSW despite its high computation time (see Section 7.2). Once more, we observe that the skill refreshing policy (SR) spreads the workload among the resources while the overskill minimization policy (OS) tends to assign the tasks to the same people.

Average Overskill Indicator The overskill indicator indicates how overskilled is a resource R to perform a task T. Let $R.Skills$ be the resource's skill vector and $T.Skills$ be the task's skill vector. The overskill indicator is given by the square of the differences between the resource's skills and the task's skills.

$$OverSkillIndicator(R,T) \quad = \quad \sum_s (R.Skills[s] - T.Skills[s])^2 \tag{33}$$

Clearly, the overskill minimization policy (OS) is the only one that minimizes the overskill indicator. All other policies are equivalent for this measure.

T	Rd	MSW	MV	SR	OS
40	8.5	8.4	8.5	8.7	6.3
60	9.0	8.6	8.7	8.7	6.1
80	8.6	8.6	8.7	8.7	6.4
120	9.1	8.6	8.9	9.1	6.4

Table 9: Average overskill indicator.

8 Extensions

This section explores many interesting ideas which could easily be integrated in our original SWF architecture.

8.1 Future scenarios

In the current architecture, we decided to allocate the best compatible resource for a task while considering the possible future steps of the workflow. Since resources are shared by the different workflows running at the same time, the evaluation of the cost of allocation could be performed against all the future steps of the *other* workflows. The problem would become more complex but priority levels between business processes could be used to limit the scope.

8.2 Scheduling of different workflows

In our architecture, the scheduling of the tasks is completely reactive. The engine follows the progress of the workflows and decides to allocate them resources on a first come, first served basis. A more informed scheduling might help in order to give priority to important processes. By doing this the tasks of critical business processes would be scheduled before the others and benefit form larger/better combinations of resources. This could be implemented by an ordered queuing of the tasks.

8.3 Cross-workflows synchronization

Business processes are usually cross-functional and involve the simultaneous or reciprocal flow of information between several functional areas. For example, the order-fulfilment process needs input from sales, logistics, manufacturing and finance as it progresses from sales order through production and cash payment [DKKS04]. Today, these dependencies are well addressed by workflows, as long as they correctly define each required input. However, situations which need the loose coordination of different processes are not easily defined. For example, in Microsoft, the activity of Microsoft Consulting Services (MCS) highly depends on the company product releases, and any information on release dates can greatly improve MCS's work.

One obvious solution here would integrate MCS's and product groups' workflows into a large and complicated workflow. Unfortunately, this would introduce an artificial complexity for the managers and will go against the accepted separation between functional areas.

To successfully address these loose dependencies, we propose to extend the SWF with a new cross-workflow synchronization mechanism. It uses a client/server architecture where workflows register their dependency to other workflows' tasks. The server is there to maintain these dependencies[2] and to trigger registered clients each time external tasks make progress.

8.4 Online and batch optimization

Balancing choices between different opportunities is part of the daily life of an organization. For instance, in a typical back-ordering problem, a company whose inventory is less that its cumulated orders must decide which customer will immediately receive the totality of their order and which customer will immediately receive part of their order and later receive the balance. These decisions may have a financial cost, especially in advanced B2B scenarios where discounts are agreed when back-ordering takes place. Constraint optimization can be used to take the optimal decisions, *e.g.,*to decide when to back-order a client.

The designer of a workflow declares a vector $\vec{q}$ of decision variables relative to a workflow. In the case of the back-ordering problem, the decision variables are the quantity that is immediately shipped to the customer and the quantity that is back-ordered. The sum of these two quantities must equal the command. This constraint is also expressed at the level of the workflow instance.

At the workflow class level, one has access to the matrix Q whose columns are the vectors of decision variables of the workflow instances. Constraints can be posted on this matrix of variables. For instance, the total quantity of delivered goods must be less or equal to the inventory.

Finally, at the system level, one can think about a collection of workflow classes. Constraints can be posted over the matrices of variables associated to each workflow class.

8.5 Offline capacity planning

Intuitively, the robustness of a workflow represents its chances of being efficiently executed. In B2B frameworks, efficiency may have different meanings. It could be the time for completion but it could also involve qualitative criteria like the ability of the resources, etc. In any case, these criteria are all related to available resources. For example, consider an organization where only one person can perform a particular task. If this resource becomes unavailable for some period, all the processes involving her unique skills could experience difficulties.

[2]Checking for consistency, *i.e.,*avoiding circular references.

The abstraction introduced by the SWF allows us to connect the previous intuitive definition to solution space. The more robust a workflow is, the bigger is its solution set.

We propose to use this relation to define an offline capacity planning able to reason on an organization's workflows. The same constraint solver used for the daily resource allocation can be used for this [Ham03b].

Such a tool could be used on a regular basis by business analysts and managers. It would use the following input:

- For each resource, the cost of acquiring some level in a new skill or a better level in an existing skill.
- A training budget limit or an objective in robustness increase.

The goal is to automatically compute a training plan made of tuples $\{(resource, skill, level)\}$ which associate an existing resource to some missing skill/level. It can respect the training budget and maximize the overall increase in robustness or meet the robustness criterion and minimize the training cost. Remark that weights could be automatically computed assuming that the past allocation are representatives of the future ones or added by analysts according to their own predictions. They would bias the reasoning toward critical business processes and improve their capacity.

The previous could easily be generalized to non-existing resources, *i.e.,*to compute hiring plans which will really have a high impact on a company's business.

9 Related Work

Senkul and Toroslu [ST05] use constraint programming for scheduling resource allocations in workflows. They focus on the satisfiability of resource allocation problems. More precisely, they study offline scheduling problems. Their model, similar to the one presented in Section 5, allows the optimization of a cost function without explicitly mentioning some resource allocation policies.

10 Conclusion

We have presented the Smart Workflow Foundation, a resource allocation system integrated to a workflow engine. The SWF abstracts business processes and company's resources to perform dynamic task to resource allocation while enforcing high level policies.

In an effort to emulate the advanced reasoning capabilities of human decision making, a form of forward-reasoning which considers the cost of the allocation against the possible future steps of a workflow has been defined and implemented. Our system plans ahead considering the upcoming tasks in the workflow and optionally considering the upcoming tasks in the other workflows running at the same time.

Many interesting work has yet to be done. For instance, as drafted in section 8, cross-workflows optimization could be performed on the fly and would allow to compromise decisions against some global utility function, and capacity planning could be easily integrated. Overall, we think that the SWF represents the right abstraction level for a new category of important software, workflow engines.

References

[ACW06] Paul Andrew, James Conard, and Scott Woodgate. *Presenting Windows Workflow Foundation*. Sams Publishing, 2006.

[DHB05] S. Dickson, Y. Hamadi, and L. Bordeaux. Optimizing resource and capacity management in Microsoft enterprise services: towards business agility. Technical Report MSR-TR-2005-188, Microsoft Research, Cambridge, September 2005.

[DKKS04] Nikunj P. Dalal, Manjunath Kamath, William J. Kolarik, and Eswar Sivaraman. Toward an integrated framework for modeling enterprise processes. *Commun. ACM*, 47(3):83–87, 2004.

[Ham03a] Y. Hamadi. Constrained workflows: Challenges and opportunities. Technical Report MSR-TR-2003-101, Microsoft Research, Cambridge, 2003.

[Ham03b] Y. Hamadi. Disolver : A Distributed Constraint Solver. Technical Report MSR-TR-2003-91, Microsoft Research, Dec 2003.

[PR05] Gilles Pesant and Jean-Charles Régin. Spread: A balancing constraint based on statistics. In Peter van Beek, editor, *CP*, volume 3709 of *Lecture Notes in Computer Science*, pages 460–474. Springer, 2005.

[ST05] Pinar Senkul and Ismail H. Toroslu. An architecture for workflow scheduling under resource allocation constraints. *Inf. Syst.*, 30(5):399–422, 2005.