A Framework for Automatic Improvement of Workflows to Meet Performance Goals

Trent Jaeger, Atul Prakash, and Masayuki Ishikawa
Department of Electrical Engineering and Computer Science
University of Michigan, Ann Arbor, MI 48109-2122.
Email: {jaegert,aprakash}@eecs.umich.edu

Abstract

Business performance improvement is arguably the most important factor in the development or reengineering of a business information system. We present a framework that, given a model of a business information system and a performance goal, helps determine the modifications required in the model in order to meet the performance goal. Our framework includes: (1) an executable system model based on workflows, triggers, and execution resources; (2) an execution environment that measures system performance; (3) improvement operators that can modify the system model; and (4) an automatic improvement mechanism that uses AI search techniques to guide the modification of the system model to meet a performance goal.

Keywords: business reengineering, performance analysis, performance improvement, workflows.

1 Introduction

Adequate performance is a requirement of any information system. If a system's performance is unacceptable, performance analysis is necessary to determine how to modify the system to improve its performance. Three major tasks comprise performance improvement of a system:

1. Specifying the system.
2. Identifying the performance bottleneck(s).
3. Choosing among the possible modifications to resolve the performance bottlenecks.

The first two tasks have been researched extensively. For example, Bodart and Pigneur [2] define a performance analysis model that identifies performance bottlenecks, given a specification of performance parameters, processes, and resources. Macrotec [8] is a graphical modeling and performance measurement system that also uses a resource model. PrM [11] is a process and resource specification language that identifies process bottlenecks using sensitivity analysis. Quartz and IPS-2 [1, 10] analyze the performance of large parallel programs to determine performance bottlenecks using specially-designed performance metrics.

Support for the third task is currently lacking. One system that provides limited support for the third task is the START/ES [3] expert system. START/ES recommends resource (i.e., hardware) changes to improve a computer system's performance. START/ES does not suggest process modifications, which can be important in business process reengineering.

In this paper we provide a framework that suggests both resource and process changes to improve a system's performance. Furthermore, our framework aids reengineers in specifying the types of modifications which should be considered and selecting between multiple applicable modifications using performance estimation. Also, the system's performance goal can include a variety of parameters, such as response time, monetary cost, labor automated, etc.

The structure of the paper is as follows. First, we define the problem: improvement of a business information system to meet a performance goal. Next, we outline our framework. Then, we define an example business information system using our framework and demonstrate the framework's ability to improve that system's performance. Finally, we present our conclusions and point out some open issues.

2 Problem definition

Before we state the problem, we define the major framework concepts:

- **Definition 1**: A *business information system*, S is a triple, $S = (W, T, R)$, where: (1) W is a set of workflows; (2) T is a set of workflow triggers; and 3) R is a set of resources.

- **Definition 2**: A *workflow*, $w \in W$, is a double, $w = (N_w, G)$, where N_w is the workflow's name and $G = (V, E)$ is a directed graph where V is the set of steps and E is the set of precedence constraints between steps. The workflow model we use is similar to the model used in Action Workflows [9].

- **Definition 3**: A *business flow*, bf, is a set of workflows, $w_i \in W$, which are triggered directly or indirectly by an external event (e.g., a customer request). The set of all business flows is BF.

- **Definition 4**: A *resource*, $r \in R$, is a triple, $r = (Sk, A, P)$, where (1) Sk is r's set of skills; (2) A is a sequence of time blocks that indicate when r is available to perform a step; and (s3) P is a set of performance attributes that increment the values of performance parameters when r executes a step.

- **Definition 5**: A *step*, $v \in V$, is a triple, $v = (c, SK, P)$, where: (1) c is a command to be executed by v; (2) SK is a set of resource skills necessary to execute c; and (3) P is a set of performance attributes that increment the values of performance parameters when v is executed. Conditional statements are also steps. The result of evaluating a conditional statement determines the next set of steps to execute in the workflow.

- **Definition 6**: A *trigger*, $t \in T$, is a double, $t = (An, N_w)$, where: (1) An is an antecedent, which is a set of boolean conditions and (2) N_w is a workflow name. If a An is true, then an instance of the workflow N_w is activated.

- **Definition 7**: A *performance goal*, G_S, for a business information system, S, is a set of goal elements, $G_S = \{ge_1, ge_2, ..., ge_n\}$. A *goal element*, ge_i, is a quadruple, $ge_i = (bf, p, g, fn)$, where: (1) $bf \in BF$; (2) p is a performance parameter for evaluating the performance of bf; (3) g is the desired goal value of a performance parameter, p; and (4) fn is a predicate over p and g that returns true if the current value of p in bf satisfies the goal.

- **Definition 8**: An *improvement operator*, op, is a quadruple, $op = (S_i, a, s, m)$ where: (1) S_i is the i^{th} version of a business information system; (2) a is an action that op implements; (3) s (a member of either $v \in V$, $r \in R$, or $t \in T$) is an object in S_i that is modified by op; and (4) m contains additional arguments for implementing op. When op is applied to S_i, a new business information system, S_j, results.

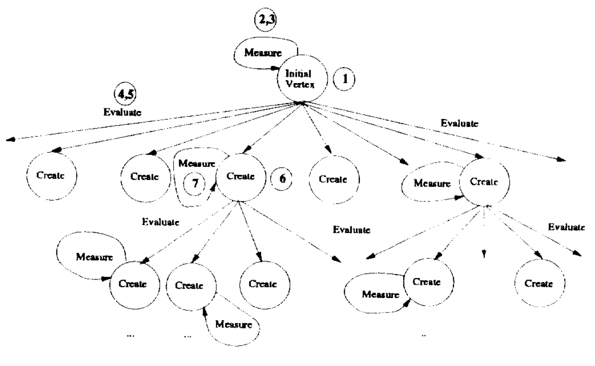

Figure 1: Automatic improvement mechanism and search space

The *automatic improvement problem* is to determine a sequence of improvement operators, $OP = (op_1, op_2, ..., op_n)$, that transform a business information system, $S_{initial}$, into another business information system, S_{goal}, which satisfies a performance goal, G_S.

The automatic improvement problem is complex because: (1) operators interact by modifying the same business information system objects and (2) the number of improvement operators for a large number of steps, triggers, and resources is large. Operators interact, so the solution space must be reevaluated after each operator application. Thus, the solution space forms a graph of business information system specifications created by operators (see Figure 1). The size of this space is exponential in the number of steps, triggers, and resources, so an automatic improvement mechanism must prune the search space to make the problem tractable.

3 Automatic improvement framework

The *automatic improvement framework* represents the automatic improvement problem's search space and uses an automatic improvement mechanism to find a satisficing [1] improvement operator sequence. The *automatic improvement mechanism* implements

[1] We use the term *satisficing* in the sense introduced by Simon in his book *The Sciences of the Artificial* to mean a solution that satisfies some set of requirements but is not necessarily optimal.

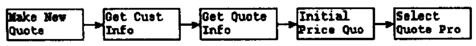

Figure 2: Create quotation workflow

a search as follows (corresponding stages indicated in Figure 1):

1. Define an **initial vertex** in the search space that represents the initial business information system.
2. **Measure** the performance of the initial business information system.
3. Define a **performance goal** for the business information system.
4. Unless the performance goal is met, generate and **evaluate** improvement operators by computing an heuristic estimate of each improvement operator's effect on meeting the performance goal.
5. **Create** n new vertices in the search space by selecting the n best improvement operators. Select a new current vertex using the heuristic estimates.
6. **Measure** the performance of the business information system represented by the new current vertex.
7. Repeat starting at step 4.

4 Framework details

To illustrate the automatic improvement framework, we improve the performance of a sample business flow, called **make quotation**. The **make quotation** business flow writes a quotation to price a list of items for a customer. Five workflows can be executed in the **make quotation** business flow:

- **Create quotation**: Collects a list of the items to be quoted (Figure 2).
- **Standard config quote, low effort config quote, high value config quote**: These three workflows are different techniques for configuring and pricing a quote. One of these three workflows is run based on the cost and complexity of the quote (Figure 3). These workflows have the same step graphs but have different service times.
- **Check inventory**: Estimates the delivery dates of the items in the quotation (Figure 4).

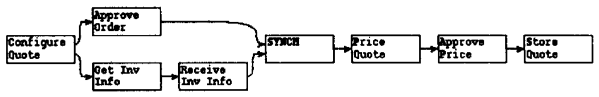

Figure 3: Standard config quote, low effort config quote, and high value config quote workflows

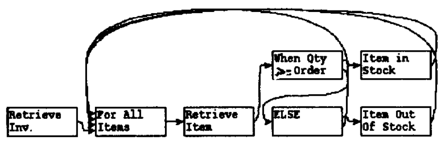

Figure 4: Check inventory workflow

The automatic improvement problem is to modify a business information system containing the above workflows so as to meet a performance goal for the **make quotation** business flow. Note that, in general, the business information system may contain other workflows, and all those workflows must be considered by the improvement framework because the various workflows may share the same resources or may affect each other via triggers. In this paper, due to lack of space, we will only consider the five workflows above.

The framework is implemented in the Common LISP Object System (CLOS) [14]. The interface to the system is constructed in Tcl/Tk [12]. The following subsections detail the implementation of each stage of the automatic improvement mechanism.

4.1 Define the business system

A business information system is defined by specifying its workflows, execution resources, and triggers.

Workflow definition A workflow definition specifies a process and its performance requirements. Examples of the workflow graphs for the five **make quotation** workflows are shown in Figures 2, 3, and 4. Workflow performance is a combination of the performance each of its steps. Step performance is represented by the step's performance attributes: (1) average service time ($1/\mu_{avg}$); (2) resource options to automate labor (*auto. by*); (3) value added; and (4) skills required. The performance attribute values for each step with non-zero service time in the five **make quotation** workflows are listed in Table 1.

Step	$1/\mu_{avg}$ *min.*	*Auto. By*	*Value Added*	*Skills*
Make New Quote	1	N/A	\$0	Pricing
Get Cust Info	2	N/A	\$0	Pricing
Get Quote Info	10	N/A	\$0	Pricing
Initial Price	1	N/A	\$0	Pricing
Configure Quote	10/10 /60	Config Support	\$200	Pricing, Prob Solv
Approve Order	5/10 /20	N/A	\$20	Manager
Price Quote	5/30 /30	DB	\$30	Pricing
Approve Quote	5/20 /20	N/A	\$20	Manager
Store Quote	2/2 /2	DB	\$10	Pricing
Ret. Quote	4	DB	\$0	Stock
Ret. Item	4	DB	\$0	Stock
Item In Stock	3	N/A	\$10	Stock
Item Out Of Stock	3	N/A	\$10	Stock

Table 1: Step Performance attributes for the `create quotation`, `low effort config quote/high value config quote/standard config quote`, and `check inventory` workflows

Resource definition Resources supply the skills to perform steps. For the `make new quote` step to be executed, resources with the pricing skill must be found (see Table 1). The resource definitions in Table 2 show that the `inside mgr`, `sales engr`, and `sales assocs` have the pricing skill. The `cost per hr` performance attribute specifies the resource cost of executing a step.

Trigger definition The business information system chooses among the `low effort config quote`, `high value config quote`, and `standard config quote` workflows using triggers. An example of the trigger that activates the `high value config quote` workflow is shown in Figure 5. The `step sizes` of the trigger are used by the automatic improvement mechanism to modify the domain in which the trigger applies. For example, a `step size` of 50 means that a trigger operator can change the *quotation->total-cost* antecedent value by ±50.

Name (count)	*Cost/Hr*	*Skills*
Inside Mgr (1)	\$100	Mgr, Problem-solving, Pricing
Sales Mgr (1)	\$120	Mgr, Problem-solving, Sales
Operations Mgr (1)	\$80	Mgr, Stock, Deliver
Sales Engr (1)	\$80	Problem-solving Pricing, Phone
Sales Assoc (3)	\$40	Pricing, Phone, Stock
Sales Person (5)	\$70	Sales, Phone
Stock Person (2)	\$30	Stock, Deliver
DB (1)	\$250	Specific Steps
Config (1) Support	\$200	Specific Steps
Mgmt (1) Support	\$100	Specific Steps

Table 2: Resource definitions (includes automation)

```
(define-trigger HIGH-VALUE-TRIGGER
  :antecedents
      (> quotation->total-cost 100)
      (<= quotation->no-of-items 5)
  :workflows high-value
  :step-sizes (quotation->total-cost 50)
      (quotation->no-of-items 1))
```

Figure 5: `High value config quote trigger`

4.2 Measure performance

Performance is measured by executing workflows using the *business flow simulator*. The business flow simulator collects data using two types of performance parameters: (1) user performance parameters and (2) performance metrics.

User performance parameters represent user-level performance information about a business flow. In our example, we used the following user performance parameters:

- *Response time*: the average difference between the start time and the completion time of the business flow.
- *Maximum resource waiting time*: the maximum *average queue waiting time* [6] for a resource of

a particular skill type (e.g. engineer, manager, clerical, etc.) used by a business flow.

- *Cost*: the average sum of resource costs per business flow execution plus an idle resource cost to this business flow. The idle cost to a business flow is $\sum_{n=1}^{|R|} I_{r_n} * U_{r_{n\&bf}} / f_{bf}$ where: (1) $I_{r_n} = (1 - U_{r_n})\$_{r_n}$ is the idle cost of resource $r_n \in R$, given its utilization, U_{r_n} and cost per hour, $\$_{r_n}$; (2) $U_{r_{n\&bf}}$ is the utilization fraction of r_n attributable to all executions of business flow bf; and (3) f_{bf} is the number of times bf has been executed.
- *Labor automated*: the average total cost savings in labor by the automation of steps per business flow execution.
- *Value added*: the average total value added by the steps performed during each business flow execution.

Performance metrics are used to compute user performance parameters and to estimate operator effects. The following set of performance metrics are defined:

- *Step slack*: Is the average `response time` reduction that the removal of a step would achieve per business flow execution [7].
- *Step sums*: Averages for `response time`, `cost`, `value added`, and `labor automated` per business flow execution are maintained for each step.
- *Queue events*: For each skill, the number of steps that are serviced by a resource with that skill is collected.
- *Queue length-time product*: For each skill, sum the products of the duration between queue events and the queue length for that duration.

4.3 Set the performance goal

The initial and goal values for the `make quotation` business flow are listed below (initial to goal).

- *Response time*: 836 minutes to 300 minutes
- *Maximum resource waiting time*: 308 minutes to 30 minutes
- *Cost*: \$308 to \$250
- *Labor automated*: \$0 to \$50
- *Value added*: \$280 to \$275

4.4 Generate improvement operators

Improvement operators are defined for three types of objects: (1) steps; (2) triggers; and (3) resources. Below, we list the improvement operators grouped by object type.

- **Step operators**
  - *Add step*: Add a new step. Can only be performed by the user at present.
  - *Remove step*: Remove an optional step.
  - *Replace step*: Replace a step or steps that achieve a state transition with an alternative sequence of steps.
  - *Delegate step*: Shift the responsibility for a step to a skill whose resources have a lower utilization.
- **Trigger operators**
  - *Remove trigger*: Remove a trigger.
  - *Expand domain*: Increase the domain to which the trigger applies by modifying one of its antecedent statements.
  - *Contract domain*: Reduce the domain to which the trigger applies by modifying one of its antecedent statements.
- **Resource operators**
  - *Add resource*: Clone an existing resource.
  - *Remove resource*: Remove a resource.
  - *Train resource*: Train a new skill to an existing resource.
  - *Focus resource*: Remove a skill type from an existing resource.

An improvement operator is created if it can improve the value of at least one of the unmet performance goal elements. The number of improvement operators created is limited by a *filter* method for each operator type. Improvement operator choices (object and operator) which are known to be monotonically worse than another operator are rejected.

4.5 Evaluate improvement operators

Each improvement operator is evaluated to determine its effectiveness for meeting the performance goal. Evaluation occurs in two stages: (1) the effect of the operator on each goal element in the performance goal is computed and (2) the individual effects are normalized into a global evaluation. The effect of an operator, op_i, on a goal element, ge_j, is computed using an *operator effect function*, $fn_e(ge_j, op_i)$. There is an fn_e for each combination of operator type and user performance parameter. Space limits prevent us from presenting all the operator effects functions, but we detail two such functions below.

$fn_e((bf_j, value\_added, g, fn), replace\_step_i)$, the effect of replacing a step v_i on the **value added** parameter for the business flow bf_j, is estimated by:

$$(v'_{i_{VA}} - v_{i_{VA}}) * (f_{v_i\_in\_bf_j} / f_{bf_j})$$

where: (1) $v'_{i_{VA}}$ is the projected average value added per step execution upon replacing v_i; (2) $v_{i_{VA}}$ is the current average value added per step execution; (3) $f_{v_i\_in\_bf_j}$ is the number of executions of v_i in bf_j; and (4) f_{bf_j} is the number of executions of bf_j.

$fn_e((bf_j, response\_time, g, fn), add\_resource_i)$ is estimated by: (1) projecting the average execution times for each step after adding the resource i and (2) deriving a projected critical path, given the projected step execution times. Step execution times can change because the average waiting time on resource i's skills will be reduced. The projected average waiting time for a skill k, W'_{qk}, is estimated by using a multi-server, Markov queue model (from Hillier and Lieberman [6]):

$$W'_{qk} = (P'_{0_k}(\lambda_k / \mu'_k)^{s'_k} \rho'_k) / (s'_k!(1 - \rho'_k)^2 \lambda_k)$$

where: (1) P'_{0_k} is the projected probability that no customers are waiting for skill k (computation also in Hillier and Lieberman [6]); (2) λ_k is the arrival rate of steps requiring skill k (assumed to be unchanged by **add resource**); (3) μ'_k is the projected service rate of skill k; (4) s'_k is the projected number of resources with the skill k; and (5) $\rho'_k = \lambda_k / \mu'_k s'_k$.

Once all the step response times have been estimated, the critical path can be derived. Actions are taken to remove loops and conditional blocks, so a standard critical path algorithm can be used [4].

To allow a comparison of effectiveness between the various possible improvement operators, the individual $fn_e(ge_j, op_i)$ values for an improvement operator, op_i, are combined into a single evaluation using a *normalization function*, $n_{op_i} = fn_n(fn_e(ge_1, op_i), fn_e(ge_2, op_i), ..., fn_e(ge_n, op_i))$.

Normalization functions have been defined for both the A* [5] and the best-first search algorithms, but, since best-first search is used to improve the example, only its normalization function is detailed here. Upon creation of the search space, a *sigmoid* function [13] is created for each goal element using two points: (1) if the initial value does not satisfy the goal use (initial value, 0.3), else use (0, 10^{-5}) and (2) (goal value, 0.7). A sigmoid function is used because it is continuous and monotonic, so any differences between $fn_e(ge_j, op_i)$ values can be translated into different n_{op_i} values.

4.6 Create new vertices

Once all the n_{op_i}'s have been computed, the automatic improvement mechanism can select the best operators to apply. To limit the fan-out of the search space, the number of operators to apply is limited to a fixed value of five, at present. The next vertex to measure is determined by the search algorithm. In best-first search, the vertex that has the highest value for n_{op_i} is the next vertex selected. The automatic improvement mechanism is then repeated starting at *Measure Performance*.

5 Example results

The automatic improvement mechanism examined 21 business information systems before it identified a specification that satisfies the performance goal. The operator sequence that satisfies the goal is shown in Table 3 (**labor automated** is always 0, so it is not shown). Business flow performance is improved by resolving a resource bottleneck on the **inside mgr** and reducing business flow cost. The resource bottleneck is resolved by using: (1) **add resource** to add a new **inside mgr**; (2) **focus resource** to prevent the **inside mgr** from performing problem solving steps; and (3) **delegate step** to change the skill requirements of the the **configure quote** step in the **standard config quote** workflow from 1 problem solving resource and 1 pricing resource to 2 pricing resources.

The **cost** is reduced by using: (1) **remove resource** to remove an underutilized **stock person** and (2) **delegate step** to require a less expensive resource; and (3) **add resource**, **focus resource**, and **delegate step** to decrease the amount of resource idle time by reducing the time spent waiting for the **inside mgr**.

The following issues require further investigation:

- **Avoidance of local minima**: We have found some examples where the automatic improvement mechanism may believe that it has found the best

Op	*Response Time*	*Max Res. Wait. Time*	*Cost*	*Value Added*
—	836 min.	308 min.	$308	$280
Add Resource	373 min.	104 min.	$319	$280
Focus Resource	322 min.	66 min.	$293	$280
Remove Resource	335 min.	74 min.	$247	$280
Delegate Step	220 min.	22 min.	$246	$280

Table 3: Satisficing operator sequence

possible solution, but the solution is not globally optimal.

- **Investigation of scalability**: It needs to be determined whether the approach will scale to business systems of larger complexity.
- **Help with operator specification**: End users must specify some domain-dependent operators, like `replace step`. Currently, users specify the set of modifications that are possible before the search begins. A better strategy would be for the system to help identify where modifications are likely to be beneficial.
- **Improvement in estimation accuracy**: Continued research is needed to develop more accurate operator effects and normalization functions.

6 Summary and conclusions

We have defined a formal, executable specification model for business information systems that enables end users to validate and to automatically improve the predicted performance of a system. The formal, executable specification model represents the functional and performance behavior of the system. These specifications are then incrementally modified using a set of improvement operators until the stated performance goal is met.

For typical systems, the number of improvement options is usually very large, so exhaustive search to find the right operator sequence is impractical. We suggest the use of AI techniques to limit the scope of the search. We provide heuristic functions that use the results of a set of performance metrics to guide the selection of operators. For a fairly complex, example business system, we found that AI techniques using these heuristic functions can be successfully used to suggest improvements in the specifications.

References

[1] T. E. Anderson and E. D. Lazowska. Quartz: A Tool for Tuning Parallel Program Performance. In *SIGMETRICS 1990*, pages 115–125, 1990.

[2] F. Bodart and Y. Pigneur. A Model and a Language for Functional Specifications and Evaluation of Information System Dynamics. In *Trends in Information Systems*, pages 195–217. Elsevier Science Publishers, 1986.

[3] E. W. Brehm et al. START/ES – An Expert System Tool for Performance and Reliability Analysis. In *Computer Performance Evaluation '92: Modeling Techniques and Tools*, pages 107–119, 1993.

[4] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[5] P.E. Hart et al. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on SSC*, 4:100–107, 1968. Also see a follow-up in *SIGART Newsletter*, pages 28-29, 1972.

[6] F. S. Hillier and G. J. Lieberman. *Introduction to Stochastic Models in Operations Research*. McGraw-Hill, 1990.

[7] J. K. Hollingsworth and B. P. Miller. Slack: A performance metric for parallel programs. Technical report, Computer Sciences Technical Report, March 1992.

[8] R. K. Keller et al. The Macrotec Toolset for CASE-based Business Modelling. In *IEEE Sixth International Workshop on Computer-Aided Software Engineering*, pages 114–118, 1993.

[9] R. Medina-Mora et al. The Action Workflow Approach to Workflow Management Technology. In *CSCW 92 Proceedings*, pages 281–288, November 1992.

[10] B. P. Miller et al. IPS-2: The Second Generation of a Parallel Program Measurement System. *IEEE Transcations on Parallel and Distributed Systems*, 1(2):206–217, 1990.

[11] A. Opdahl and A. Solvberg. A Framework for Performance Engineering during Information System Development. In *Advanced Information System Engineering*, pages 65–87. Springer-Verlag, 1992.

[12] J. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.

[13] D. E. Rumelhart et al. Learning internal representations by error propogation. In *Readings in Machine Learning*, pages 115–137. Morgan Kaufmann Publishers, 1990.

[14] G. Steele Jr. *Common LISP: The Language*. Digital Press, 1990.