Management and Utilization of Knowledge for the Automatic Improvement of Workflow Performance

Trent Jaeger
Atul Prakash

Software Systems Research Laboratory
Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48105, USA
E-mails: jaegert|aprakash@eecs.umich.edu

ABSTRACT
We present a framework that enables reengineers to build a base of performance improvement knowledge that can be used to automatically improve workflow performance. Automatic improvement of workflow performance involves modification of a business information system such that the predicted performance of its business workflows satisfies a performance goal. The number of possible modification options is very large, so a significant body of knowledge is needed to choose among them. We demonstrate, using a simple example, the requirements for the types of knowledge necessary in a automatic improvement framework. We define a knowledge model for representing these types of knowledge. We use the model to provide the framework with a body of domain-independent performance improvement knowledge. We then describe how the framework enables reengineers to provide additional performance improvement knowledge to the model and how the framework utilizes that knowledge to automatically improve workflow performance to meet the performance goal.

KEYWORDS
Workflows, business process reengineering, simulation, knowledge acquisition, knowledge representation, search algorithms.

INTRODUCTION
The popularity of business improvement methodologies, such as Business Process Reengineering [7], Business Process Innovation [5], and Business Process Improvement [8], demonstrates the importance of workflow design to the business community. These methodologies espouse the theory that business performance is a result of its workflows. Therefore, these methodologies improve business performance through the improvement of the business workflows.

Determination of a satisfactory set of workflows is a complex, time-consuming task. Three major tasks comprise workflow improvement:

1. Definition of the system
2. Identification of performance bottleneck(s)
3. Selection of the modifications to the system that mitigate the performance bottlenecks

In the past, research effort has concentrated on the first two tasks in domains such as PERT network analysis [2], software performance engineering [14], parallel program performance improvement [1, 13], and business process reengineering [3, 10]. All these tools simulate the system in question for collecting information about its performance. Some of these tools support specialized analysis of the simulation results. For example, PrM [14] uses sensitivity analysis to identify performance bottlenecks. Also, IPS-2 [13] and Quartz [1] use specialized performance metrics to identify possible performance improvement options.

Support for choosing the modification options to remove a performance bottleneck is currently lacking, however. Two systems that provide some support for the third task are the START/ES [4] expert system and our Automatic Improvement Framework [9]. START/ES recommends resource (i.e., hardware) changes to improve a computer system's performance. The major limitation of START/ES is that it does not suggest any modifications to the computer system processes (i.e., workflows).

COOCS 95 Milpitas CA USA © 1995 ACM 0-89791-706-5/9 5/08..$3.50

Often a performance bottleneck may be removed by redesigning a process, so workflow modifications should be supported. In the Automatic Improvement Framework, both resource and workflow modification options are suggested by the framework. In addition, the framework uses performance analysis metrics, such as those suggested by IPS-2 and Quartz, to estimate the effects of applying each option. The Automatic Improvement Framework uses these estimates to guide a search for a workflow design that satisfies the performance goals of the reengineer. We demonstrated the ability of the framework to reengineer a workflow in a business information system to meet its performance goals in [9].

In this work, we further investigate the knowledge requirements necessary to effectively find satisfactory workflow designs using the Automatic Improvement Framework. We choose a simple, well-documented example that demonstrates that our framework tends to settle on a local optimum using our current set of modification operators. From this example, we identify the several types of knowledge that are necessary to escape local optima. We extend the reengineering knowledge model to represent these types of knowledge and initialize the model with some domain-independent examples of this knowledge.

After extending the knowledge base, the framework is still missing some valuable reengineering knowledge: the reengineer's domain knowledge about possible modifications. Obtaining this domain knowledge is difficult, however. Reengineers often have significant domain knowledge, but they do not want to spend the time to enter it unless they know it will be relevant. Therefore, we modify the framework's improvement mechanism to integrate knowledge acquisition with the search for a solution.

The goal of the revised Automatic Improvement Framework is now to support the reengineer in the development of a knowledge base for workflow improvement. Like previous tools, the framework identifies workflow bottlenecks for the reengineer. However, the framework does not require the reengineer to choose a specific modification, but rather, the reengineer can suggest multiple potential modifications that the framework can evaluate. With this knowledge, the framework can then automatically perform the arduous task of identifying a business information system that satisfies the performance goal.

The structure of the paper is as follows. First, we review the formal definition for the problem of workflow improvement which we call the *automatic improvement problem*. We then define an example improvement problem that motivates the design of our knowledge base. Next, we outline the revised automatic improvement framework. Then, we show how it is used to improve the workflow performance of the example. Finally, we present our conclusions and point out some open issues.

AUTOMATIC IMPROVEMENT PROBLEM

Before we state the problem, we define the major 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 [12].
- **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 (3) 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.

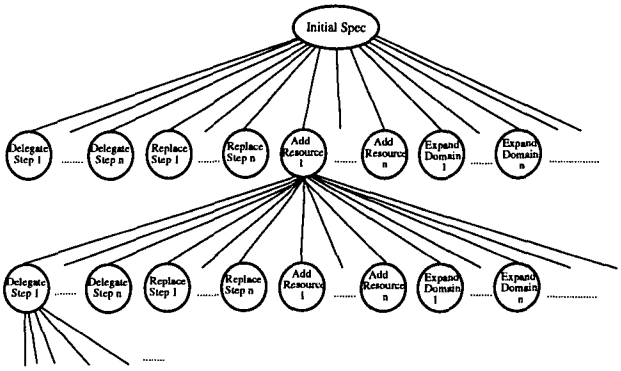

Figure 1: Improvement Search Space

- **Definition 8**: An *operator*, *op*, modifies an instance of the business information system S_i to create a new business information system S_j.

The *automatic improvement problem* is to determine a sequence of 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 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.

ORIGINAL FRAMEWORK

We summarize the original framework's mechanism for solving the automatic improvement problem. A detailed description of the original framework can be found in [9].

The original framework implements a best-first search over the space of business information systems to find a system that satisfies the performance goals of the business. Modifications to the business information system can be made using operators. In the original framework, operators for modifying steps, resources, and triggers are defined. Note that these are the most primitive objects that define the business information system. Operators are chosen based on the results of a simulation of the current business information system. Data collected during the simulation indicates the performance bottlenecks in the system. In addition, more complex metrics are computed from the simulation data to estimate the effect that an application of an operator has on the performance of the business information system. These estimates are used by the best-first search algorithm to choose at each step in the search the operator that leads to the best predicted performance.

AN EXAMPLE

We test the ability of the framework to improve workflow performance using Dowdy and Lowery's JiffyBurger example [6]. This example is chosen because it is simple, yet still demonstrates the need for a broad range of knowledge.

Problem Description

JiffyBurger is a restaurant run by two employees. The JiffyBurger employees service customers using two steps: 1) taking a customer order and 2) filling a customer order. The goal is to determine the best workflow for the two employees to provide hamburgers for their customers. The authors suggest five possible workflow and resource options for performing these steps:

1. **Common-Line**: Each worker can perform both steps for each individual customer. Customers wait in a single line for the next worker to become free.
2. **Twice-As-Fast (TAF)**: The two workers work together on each step. The steps are performed twice as fast. The customers wait in a single line for both steps to be completed.
3. **Sequential**: For every customer, one worker performs one step, and the other worker performs the other step. There are two customer lines: one for each step.
4. **Random-Line**: Each worker can perform both steps for each individual customer. Customers wait in a line for a worker to become available. The choice of line is made randomly.
5. **Shortest-Line**: Each worker can perform both steps for each individual customer. Customers wait in a line for a worker to become available. Each customer chooses the shortest of the two lines.

Input data for the example are:

- Exponential distribution of arrival and service times is assumed.
- Average time for one worker to perform either job is 1 minute.

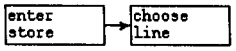

Figure 2: **Buy-a-burger** workflow

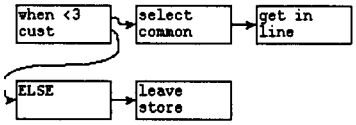

Figure 3: **Common-line** workflow

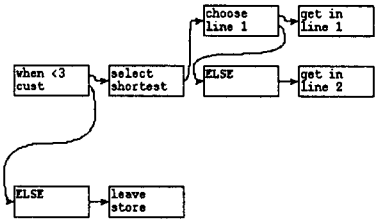

Figure 4: **Choose-line** workflow

- Average time for one worker to perform both jobs is 2 minutes.
- Average time for both workers to perform both jobs for one customer is 2 minutes.
- Customers enter the store with an average arrival rate of one customer every 2 minutes.
- If there are 3 customers in line when a new customer enters the store, then the new customer leaves the store.

Business System Specification

The five business flow options in the JiffyBurger example are implemented in our framework using three types of workflows: 1) store entry; 2) line entry; and 3) order processing. A customer enters the store and chooses a line according to the store and line entry workflows, respectively. The JiffyBurger employees execute the order processing workflow. The **buy-a-burger** workflow (Figure 2) represents the entry of a customer into the store. The **TAF**, **Common-Line**, and **Sequential** options all use a single line, so they share the **common-line** workflow (Figure 3) for line entry. The **Shortest-Line** and **Random-Line** options each uses a variation of the **choose-line** workflow (Figure 4) to choose the appropriate line. Order processing is represented by several variations of the **make-burger** workflow (Figure 5).

Differences between the JiffyBurger options are represented by changes in average service time and skill requirements in the workflow steps of the **make-burger** workflows. The two steps for which these parameters

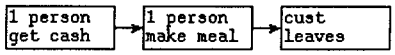

Figure 5: **Make-burger** workflow

Option/Step	*Avg Service Time*	*Skills*
Line 1/TO	1 min.	Cash 1
Line 1/FO	1 min.	Cook 1
Line 2/TO	1 min.	Cash 2
Line 2/FO	1 min.	Cook 2
Common/TO	1 min.	Cash
Common/FO	1 min.	Cook
TAF/TO	0.5 min.	Cash 1/Cash 2
TAF/FO	0.5 min.	Cook 1/Cook 2
Sequential/TO	1 min.	Cash 1
Sequential/FO	1 min.	Cook 2

Table 1: Step Definitions for **take order(TO)** and **fill order(FO)**

are varied, **take order** and **fill order**, are shown in Table 1. The service time for the **TAF** workflow steps is half of that of the other steps because the two employees work together.

The two JiffyBurger employees are the resources in the example. Resources are defined by the skills that they can perform. Each employee has **cash** and **cook** skills, and skills which determine which employee services which line. **jiffy-1** services line 1 (i.e. has the **cash-1** and **cook-1** skills) and **jiffy-2** services line 2 (i.e. has the **cash-2** and **cook-2** skills).

ORIGINAL FRAMEWORK RESULTS

We show that the original framework settles on a solution that is a local optimum in all cases for this problem. We attribute this behavior to a lack of knowledge about large granularity changes, operator pruning, and, to a lesser extent, the search mechanism.

The performance results for the five options are shown in Table 2 (where X is throughput in customers per minute, R is response time in minutes, and W is waiting time in minutes). The *autonomous, common-line* system has the shortest waiting time, while the *twice-as-fast* system has the best throughput and response time. We choose the performance of the **TAF** option as the goal for our business. Therefore, we expect the framework to convert each of the five system options into the **TAF** option or some other option that performs better than the **TAF** option.

The original framework uses a best-first search algo-

Option	*X*	*R*	*W*
TAF	0.466	1.571	0.571
Sequential	0.423	2.909	0.909
Random-Line	0.423	2.909	0.909
Shortest-Line	0.450	2.333	0.333
Common-Line	0.454	2.202	0.202

Table 2: Queuing Theory Results for the 5 JiffyBurger Options

Initial Option	*Best Option*
TAF	TAF
Sequential	Common-Line
Random-Line	Random-Line,
Shortest-Line	Shortest-Line
Common-Line	Common-Line

Table 3: Initial and Best Options Found by the Original Framework

rithm to guide the search. The only operators that are available to the original framework are operators that modify the atomic entities in the business information system: the steps, resources, and triggers. These operators include replace step, delegate step, expand trigger domain, contract trigger domain, remove trigger, remove resource, train resource focus resource(definitions of these operators are provided in [9]). We are not permitted by the constraints of the example to add resources or remove steps.

Table 3 shows the best option which is generated within 25 iterations of the automatic improvement mechanism from each of the 5 initial options. We choose 25 iterations because the search space is small, so the framework should be able to find the solution within 25 iterations.

In each instance, the search mechanism selects a solution that is a local optimum. We identified three causes for this problem:

1. large granularity operators that may move the search to a significantly different area of the search space are not available;
2. too many operators that make minute changes are not pruned; and
3. the best-first search mechanism always chooses the highest evaluated choice.

Consider the transition from the **sequential** option to the **TAF** option. This transition requires a sequence of two operators: (1) convert the **take order** step from

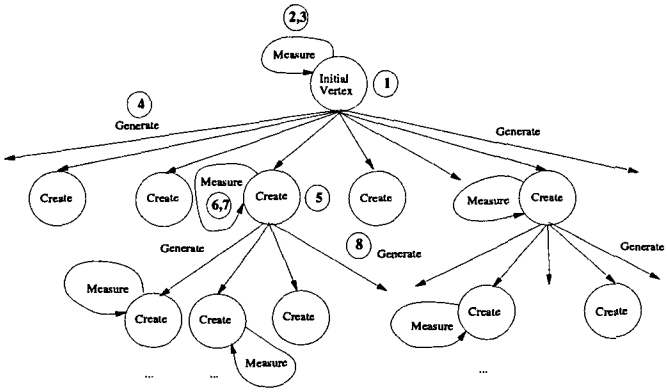

Figure 6: Automatic improvement mechanism

a 1-employee step to a 2-employee step and (2) convert the **fill order** step from a 1-employee to a 2-employee step. The problem is that the workflow specification that has one 1-employee step and one 2-employee step performs badly; there is a long waiting time for the employee that performs both steps. Since the framework also generates several other operators that do not significantly change the performance of the system, the best-first search mechanism always chooses one of these other operators instead.

REVISED AUTOMATIC IMPROVEMENT FRAMEWORK

There are two potential sources for the type of knowledge that will improve the framework's performance: (1) knowledge about improvements to more abstract concepts and (2) the reengineers' domain knowledge about system-specific improvements. Knowledge about how to improve more abstract business system concepts, such as workflows and departments, enables the framework to make the substantial modifications to the business information system that are necessary to escape a local optimum. For example, the original framework knows how to modify the resource requirements of a single step, but it does not know how generate a workflow where the resources are optimized in a specific way (such as a pipeline). However, this type of knowledge tends to identify the extremes of the modification possibilities. To perform more subtle improvements, the domain knowledge available to the reengineer must be added to the framework. For example, the reengineer may know that a specific step in the business can be implemented faster by a resource with a different skill, but it is unlikely that a generic framework would have this knowledge.

Thus, the goal of the revised framework is to enable the reengineer to develop the knowledge necessary to find a business information system that satisfies the performance goal. The framework can then implement the search for the satisficing business information system

automatically. The framework achieves this goal by: (1) managing a knowledge representation of improvement knowledge and (2) utilizing this knowledge to find solutions and to acquire new knowledge from the reengineer. Our experience indicates that reengineers will extend the knowledge base in an incremental fashion, so the revised framework is designed to acquire knowledge from the reengineer as the solution progresses.

Automatic Improvement Knowledge Model

We first define the revised knowledge model. The knowledge model consists of: (1) operator type definitions; (2) the rules that generate operator instances; and (3) the metrics that evaluate the effectiveness of operator instances. Below, we define the concepts in the knowledge model. Note that these definitions supersede the previous definition of an operator (Definition 8).

- **Definition 9**: An *simple 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 (one of add, remove, or modify s); (3) s (a member of either $v \in V$, $r \in R$, $t \in T$, $w \in W$, $bf \in BF$, or S_i) 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.
- **Definition 10**: An *compound operator*, op_c, is a double, $op_c = S_i, OP$, where: (1) S_i is the i^{th} version of a business information system and (2) OP is a sequence of operators (either compound or simple operators) that are used to implement the compound operator.
- **Definition 11**: An *operator effects function*, is a function, $fn_e(p_j, ge_j, op)$, that estimates the effect that operator *op* has on the value of the performance parameter p_j in goal element ge_j. For a compound operator, fn_e combines the effects of its constituent operators. See [9] for more information and the definition of specific operator effects functions.
- **Definition 12**: An *operator type*, *ty*, is a double, $ty = (C, Fn_e)$, where C is a set of constructor functions for creating operator instances (called *op*'s, see below) of operator type *ty*. Fn_e is a set of operator effects functions for the parameters changed by operators of that type (see Definition 13, below).
- **Definition 13**: A *operator generation rule*, r_{og}, is a double, $r_{og} = (cond, act)$, where *cond* is an antecedent condition that must be satisfied before a new operator is instantiated and *act* is an action statement that instantiates an operator of a specific type using one of its C functions.

The *simple operators* are the operators applied to a single step or resource. *Compound operators* represent an aggregation of operators, either simple operators or other compound operators. Thus, an operator of arbitrary complexity can be created.

An *operator type* definition defines a specific type of change to the business information system. An operator type is associated with a set of constructor methods that define how an operator of that type is generated and a set of operator effects functions for estimating the effect the operator has on the system's performance. The effect an operator has on the system determines its evaluation. Operator effects functions are fairly complex and domain-independent, so we do not expect reengineers to add operator effects functions.

An *operator generation rule* is used to specify when to generate an operator of a specific type. When the condition of the operator generation rule is true, then an operator instance of the associated type is created using the constructor specified in the corresponding action statement.

A goal of the knowledge model is to assist the reengineers in entering operator generation rules into the knowledge base. Historically, users have had trouble entering formally correct rules into a knowledge base, so we only require that the reengineers add modification options to the system. The framework has performance metrics to identify performance bottlenecks and knowledge about the types of operators that can eliminate a bottleneck.

A *modification attribute* is an attribute of an object that specifies options for the values of one of the object's attributes. Three types of modification attributes can be specified for an object attribute `X`: (1) `add-X` contain values that can be added to the object's attribute `X`; (2) `modify-X` contain a set of values that can be used to replace the value of `X`; and (3) `remove-X` contains the values that can be removed from `X`. For example, a resource has certain skills it uses to perform a task. The modification attribute `add-skills` specifies the other skills that can added to the resource and the effect of this change on the values of other resource attributes, such as cost. Whenever an object is identified as a bottleneck and the modification of attribute `add-X`, `modify-X`, or `remove-X` may positively affect that bottleneck, then operators are created using those values. For example, when a skill is in high demand, resources that can add that skill using `add-X` or `modify-X` are generated. Operator types for each of the modification attributes are defined to ensure that the reengineer enters the correct information and to provide operator effects functions for the attribute.

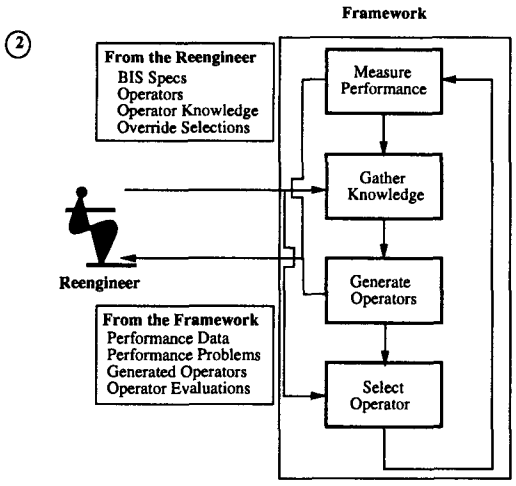

Figure 7: Interaction between reengineer and framework

Automatic Improvement Mechanism

The automatic improvement framework uses an automatic improvement mechanism guide the development of solutions to the automatic improvement problem. The *automatic improvement mechanism* implements a search as follows (corresponding to stages indicated in Figure 6):

1. **Define an initial business system** that is represented by the initial vertex in the search space.
2. **Measure the performance** of the initial business information system.
3. **Gather knowledge** about how the initial business information system can be improved. Set the performance goal.
4. Unless the performance goal is met, **operators are generated** to continue the search. The generation step includes operator evaluation.
5. **Select the next operator** in the search space to simulate.
6. **Measure the performance** of the initial business information system.
7. **Gather more knowledge** about how to improve the business information system's performance using the new current vertex as the focal point. This step is essentially the same as step 3.
8. Repeat starting at step 4.

The framework provides several operator types (many associated with modification attributes) and operator generation rules in the initial knowledge base provided to the reengineer. After the reengineer enters the initial business information system definition, the mechanism implements an interaction between the reengineer and the system shown in Figure 7. The framework interface provides information to the reengineer at the following points in the automatic improvement mechanism:

- **Measure performance**: Simulate the business information system and collect performance data.
- **Gather (more) knowledge**: The framework identifies performance problem areas in the business information system.
- **Select operators**: The framework identifies the current best operator and other operators that apply, but are not generated due to a lack of values for modification attributes.

Given this information, a reengineer can provide the additional knowledge to the framework:

- **Gather (more) knowledge**: Specify additional values for modification attributes to specify additional operators that can be generated.
- **Generate operators**: The reengineer can request that a specific operator be generated by the framework. This operator is added to the current set of operators for the current business information system. Also, the reengineer can remove generated operators.
- **Select operators**: Select another operator as the next operator that the framework should try.

FRAMEWORK DETAILS

In this section, we detail the way that the reengineer uses the revised framework to solve the automatic improvement problem for the JiffyBurger example.

Initial Knowledge Base

The initial knowledge base contains the definition of several operator types and operator generation rules. Primitive operators for steps, resources, and workflow triggers are defined in [9]. In this section, we define the new operator types that are available to modify the more abstract framework concepts. In this example, we are only permitted to modify the workflows and business flows, so only operators that apply to these concepts are defined. The operator generation rule and construction function of one of these operators are then detailed.

- **Pipeline**: Convert the workflow to a pipeline. This is done by using several `modify step` operators to assign each step skill to a specific resource (e.g.

Collaborate Rule
If *response_time(bf)* < *response_time(ge)*
Then *collaborate(bf)*
/* Increase collaboration in business flow */

Figure 8: Operator Generation Rule for `collaborate`

`cash` to `cash-1` for the resource assigned to `cash-1`). This operator applies when the average response time is greater than the arrival rate.

- **Guide**: Again use `modify step` to assign a skill to each step in the workflow to serve as a 'guide' for the business flow (e.g. include the `cash` skill in every step). Choose the skill that is required the longest amount of time in the business flow. Use this operator when the waiting time is greater than the goal value.
- **Collaborate**: Increase the number of resources participating in each step of the workflow to reduce step duration. The reengineer must specify the effect of adding a resource. No change is made if this effect is not specified. The `modify step` operator can be used to redefine the specifications of skills and effort for the step. This operator should be run when the response time is greater than the goal value.
- **Simplify**: Add a simplified version of the business flow to handle the 'easy' cases. Simplification is done by removing optional steps, such as authorization steps. The initial domain for easy cases should be specified by the reengineer. Several `add step` operators compose this operator. This operator should be run when the response time is greater than the goal value.

We now define an example of a operator generation rule and constructor function for one of our new operators, `collaborate` (shown in Figure 8). This operator applies when the response time of a business flow is below its goal value.

The constructor function for `collaborate` is shown in Figure 9. This constructor generates a compound operator that consists of a set of simple operators that minimize the time it takes to execute each step in the business flow. The attribute `modify-skills` contains the skill (i.e., resource) options for a step. For each step, the skill set that is estimated to perform the step in the minimum time is found by `find_collab`. If this is not the current skill set, then an operator is created to modify the step's skill set. The `collaborate` operator is composed of these `modify step` operators. If no steps can increase the collaboration in the business flow, the operator is deleted.

collaborate(bf)

fast_skills a skill set /* *step*'s min duration skill set */
step_op an operator /* op to modify skill set of *step* */
this an operator /* the collaborate operator */
sub_ops a set of operators /* the set of *step_op* in *this* */

for *step* in *bf*
  /* Find skill set in *step*'s modify-skills that performs
  * the *step* in the minimum duration */
  fast_skills =find_collab(*step*)
  /* If *fast_skills* is not the *step*'s current skill set
  * Modify *step*'s skills attribute to *fast_skills* */
  if (*fast_skills* != *step*->current_skills())
    step_op = *step*->modify("*skills*", *fast_skills*)
    /* *sub_ops* stores *step_op*'s that compose *this* */
    add *step_op* to *sub_ops* attribute of *this*
/* If no *sub_ops*, prune this op */
if (!*sub_ops*)
  mark *this* for deletion

Figure 9: Constructor Function for `collaborate`

Business System Definition

The specification of the business information system objects and performance data for our example is shown in the An Example Section. In addition, the reengineer may enter operator generation knowledge using the modification attributes. Modification attributes specify how an object's definition can be changed. Each system object has a set of modification attributes, including the general modification attribute `modify-specs`. Below, we list some examples of modification attributes.

- **Steps**
  - `Modify-Effort`: Change the time it takes to execute a step.
  - `Modify-Skills`: Replace the skills required to execute a step with a new set of skills. A change in a step's skills also affects the effort to execute a step.
- **Workflows**
  - `Add-Steps`: Add a step, include its dependencies.
  - `Modify-Steps`: Replace one or more steps with these steps.
  - `Remove-Steps`: Steps that can be removed from the workflow.
- **Business Flows**
  - `Modify-Workflows`: Modify the definitions of one or more workflows in the business flow.
- **Resources**

- `Add-Skills`: A set of skills that this resource can learn. Any modification to skills may affect the cost of the resource.
- `Modify-Skills`: Replace the resource's current set of skills.
- `Remove-Skills`: Skills that can be removed from the resource.
- `Modify-Cost`: Replace the cost of the resource.

- **Departments**
  - `Add-Resources`: Definition of a resource that can be added.
  - `Modify-Resources`: Modify the set of resources available to the department.
  - `Remove-Resources`: Resources that can be removed.
  - `Add-Workflows`: Add a workflow to the department (note that workflows are associated with a single department).
  - `Remove-Workflows`: Workflows that can be removed from the department.

- **Business Information System**
  - `Modify-Departments`: Modify the specifications of departments in the business.

Measure Performance

The framework measures the performance of the current business information system and presents the major performance issues to the reengineer. The goal is to focus the reengineer on the important performance problems in the business information system, so the reengineer may be motivated to supply additional knowledge for the framework to use. The framework presents summary information about the performance of each business flow, so the reengineer can see the performance of each step and skill in a business flow. The following are a list of some of the performance problems that are identified by the framework:

- The step in each business flow that accounts for the most time on the critical path [11].
- The step in each business flow that accounts for the most slack on the critical path [11].
- The skill with the highest average queue waiting time [15].
- The skill in the business flow with the highest average queue waiting time.
- The step that accounts for the highest cost in the business flow.

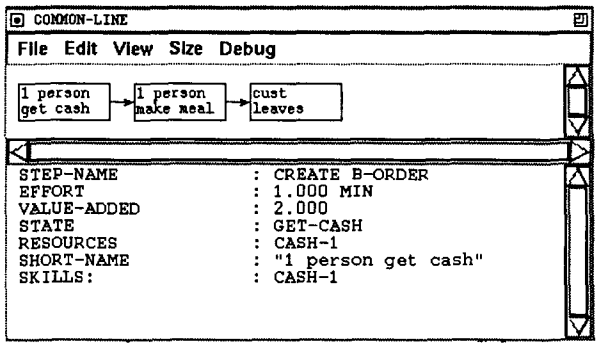

Figure 10: Operator Specification Interface

Gather Knowledge

Once the reengineer reviews the measurement results, the reengineer may be able to provide additional domain knowledge to the framework. This domain knowledge is provided in the form of modification attribute values (see Business Information System Section, above). However, for some complex modification attributes, we would like to provide some interface support, so that it becomes relatively straightforward to enter the attribute values.

Observing that the framework does not try the sequence of operators that would generate the `TAF` option from any of the other four options, we would like to enter a rule to generate a compound operator that implements this change. The `modify-workflows` modification attribute of business flows stores modification options for changing multiple workflows in a business flow. However, we do not want to burden the reengineer with too much knowledge about the modification attributes. What we would like to do is to present the reengineer with an interface where the reengineer can demonstrate a modification to a business flow. This modification is then stored in the `modify-workflows` modification attribute.

The new modification option, called `create TAF` by the reengineer, is added using the business flow specification interface (see Figure 10). The reengineer first chooses the `add operator` command in the `file` menu selection. The reengineer then demonstrates the modifications that would be made to implement the change by editing each of the business flow's workflows until the resultant definition is created. The modification attribute stores a specification for changes to a set of workflows (in the same form as the `modify-steps` modification attribute of workflows). The specification can be used to create a compound operator consisting of a set of `modify` operators on the business flow's steps. The following modifications are implemented by the `create TAF` operator:

- Modify the `choose line` step in the `buy-a-burger` workflow to choose the `common-line` workflow.
- Modify the `get cash` step in the `make-burger` workflow to use two skills (`cash-1` and `cash-2`) and reduce the effort to 0.5 minutes.
- Modify the `complete meal` step in the `make-burger` workflow to use two skills (`cook-1` and `cook-2`) and reduce the effort to 0.5 minutes.

The operator effects functions for `modify-workflows` can be applied to estimate the effect of this operator on the current business flow. The reengineer can then see the predicted performance of this operator, so the reengineer can determine whether to remove or modify this operator or add other operators.

Generate Operators

Operator generation is implemented in four sequential steps by four different types of operator generation rules. The operator generation rule types are listed below in the sequence that they are used:

- **Indexing**: An indexing rule generates an operator that applies in the current situation. For example, the operator generation rule for the `collaborate` operator is an indexing rule. A `collaborate` operator can be generated when the response time needs to be reduced.
- **Implication**: An implication rule generates a set of operators to support the implementation of an operator. For example, if the framework creates a `pipeline` operator, additional resources may be needed to perform the steps in the pipeline. New operators that add these resources are created and aggregated with the `pipeline` operator.
- **Aggregation**: An aggregation rule creates a compound operator that includes a set of operators that are identified to be compatible by the rule. For example, if an operator increases the usage of one skill such that this skill now has the maximum queue waiting time, then an operator that reduces the demand on this skill can be aggregated with it. Note that the original operators remain in the set of operators for the current business information system.
- **Elimination**: An elimination rule prunes operators that are not expected to provide any help. For example, if we can prove that an operator cannot perform better than another, we can eliminate it. Also, we want to eliminate operators that have been applied several times in different circumstances, but do not change the system's performance significantly.

The implementation of implication and aggregation rules requires that the actions of any operators that are to be aggregated do not conflict. For example, we cannot aggregate two operators that change the skill requirements for the same step to different values because one operator will undo the change of the other. We define two operators as conflicting if they modify overlapping specification attributes in different ways. That is, two operators can modify the specifications of the same object as long as they do not modify the value of the same attribute. Also, two operators can modify the same attribute in the same object as long as they set it to the same value.

The reengineer may provide input for generating operators directly by requesting that an operator be generated or removing an operator that the system generated. A reengineer may also request that an operator always be or never be generated in a run of the automatic improvement mechanism. This is essentially the specification of a condition for an action in an operator generation rule. We do not provide support to add conditions, at present, but we think we will need to permit the reengineers to add simple conditions in the future.

Select Operators

The search mechanism in the revised framework is modified to help it avoid local optima by: (1) preferring operators that make greater change and (2) allowing the mechanism to move to a worse state. Operators that make a large change can move the search into a significantly different area of the search space. By allowing the search algorithm to move to a worse state, the mechanism may be able to move toward a new, more global optima.

To achieve our goals, we add a simulated annealing capability to the best-first search mechanism used in the original framework [16]. Simulated annealing permits the mechanism to choose a worse state with a probability that decreases as the search progresses. We make the simulated annelaing capability sensitive to the complexity of the operator, so an operator that reduces the performance of the system when a major change is made is preferred over an operator that reduces the performance of the system with a minor change. The simulated annealing, best-first search mechanism consists of the following components:

- The *search algorithm* that maintains the set of untried operators and their evaluation values and the best state found so far.
- A *probability function* that computes the likelihood that it is appropriate to select a particular operator.

- An *annealing schedule* that evolves the probability that a worse state will be chosen.

The algorithm works as follows. At each iteration in the search, the untried operators from each search vertex are collected. These operators include the operator with the best evaluation value so far. The best evaluation value is set to the variable B. For each operator, we compute a probability that the operator is applicable at the present time using the formula

$$p = e^{-(B-E)/kT}$$

where: (1) E is that operator's evaluation; k is the number of changes necessary to implement the operator; and, (3) T is the current annealing temperature based on the annealing schedule. k is the operator complexity measured by the number of simple operators needed to implement the operator. The next operator is randomly chosen from the set of operators whose value for p is greater than a random number generated between [0,1]. The value for T is computed as follows, $T = 1/3n$ where n is the number of states examined so far. Exponential and geometric functions reduce the probability of choosing a worse state too quickly in this example.

REVISED FRAMEWORK RESULTS

The **TAF** option is found within a few search nodes by any one of three operators: **create TAF**, **collaborate**, or operator generation rule added by the reengineer in the Generate Operators Section. It is not guaranteed that one of these operators will be chosen first due to the simulated annealing property of the search algorithm. Sometimes an operator that is not expected to perform as well is chosen first.

Of course, we have no guarantee of performance improvement based on the solution of one simple example, but we have several reasons to expect more robust performance from the framework. We are pleased that the framework has multiple ways to find the **TAF** option. In the original framework, there is only one way to find the **TAF** option, and the original framework continually rejects this option. Also, we expect better performance due to the addition of more domain-specific knowledge. Finally, other types of generation knowledge, such as implication, aggregation, and elimination knowledge will give the framework more power to make effective decisions.

There still appears to be room for further response time improvement in the **TAF** options. We see that in the **TAF** option both the collecting payment and producing a burger steps are on the critical path. Our goal is to remove one of those steps from the critical path. Thus, we devise the following new option.

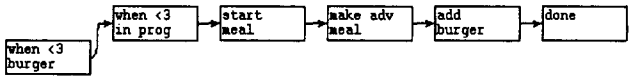

Figure 11: **Prepare-meal** workflow

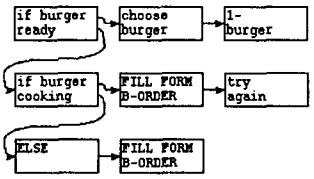

Figure 12: **Fill-order** workflow

6. **Burgers-Ready**: The two workers service customers together, but when there is time they make burgers. We never allow the surplus of burgers to exceed 6. The initial number of burgers is set to 6.

The **burgers-ready** option involves two separate, parallel business flows. In one business flow, a customer pays for and receives a burger. When there are no customers, the second business flow is run to make burgers for future customers. If a burger has already been made, the service rate of the workflow is reduced, otherwise the service rate is the same as the **TAF** option. The **burgers-ready** operator consists of the following modifications:

- Create the **prepare-meal** workflow (shown in Figure 11).
- Create the **fill-order** workflow (shown in Figure 12).
- In the **buy-a-burger** workflow, replace the **choose line** step with a step that sends a message to trigger the **common-line** workflow.
- Replace the **complete meal** step in each of the **make-burger**, workflow with a step that sends a message to the **fill-order** workflow.
- Add an arrival specification for the **prepare-meal** business flow with an arrival rate of 1 every 1.5 minutes and the **buy-a-burger** business flow with an arrival rate of one every 2 minutes.

The two new workflows define how burgers are made and how to pick up an already-made or soon-to-be-made burger. The addition of the new steps specify which workflows are to be used.

The **burgers-ready** option does indeed provide a shorter response time to the customer, but the waiting time

exceeds that of the **TAF** option. The customer sometimes has to wait for the employees to finish a burger in progress because the **prepare-burger** workflow is not preempted when a new customer arrives. Because of the high waiting time, this option does not meet the performance goal for this business information system.

- **Response Time**: 1.398 min
- **Throughput**: 0.494 customers/min
- **Waiting Time**: 0.854 min

CONCLUSIONS AND FUTURE WORK

In this paper, we define a knowledge base for the automatic improvement of workflow performance. The knowledge base provides operator generation and selection knowledge that guides the automatic improvement framework to find a business information system specification that satisfies a performance goal. The reengineers generally have access to valuable domain-specific knowledge that enables further improvement to the business information system. Therefore, a knowledge acquisition tool that poses performance problems and collects candidate solution has also been constructed.

The addition of the improvement knowledge base enables us to easily solve our example problem. In fact, we are even able to specify a new modification option that further reduces the response time. The waiting time is higher than the goal value, however. Of course, we cannot guarantee performance based on the solution of one simple example, but many of the issues of a large problem are represented here. We believe that we have made progress in the areas of: (1) collection of domain knowledge; (2) reducing susceptibility to local optima; and, (3) defining the types of knowledge required in the framework. We expect that additional interface tools will be necessary to enter domain-specific knowledge, such as operator generation rules, however.

The main improvement to the current framework is to improve the access techniques available to the reengineer. A reengineer may want to request a specific piece of information from the framework. We need a declarative query language for the performance data and improvement knowledge base, so the reengineer can express these requests. Scalability remains an unresolved issue, although we are confident that we can control the problem size using techniques such as PrM's models [14] and further limits to the amount of calculation per object.

ACKNOWLEDGEMENTS

We thank the anonymous referees for their many helpful comments.

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] A. Badiru. A Simulation Approach to PERT Network Analysis. *Simulation*, pages 245–255, October 1991.

[3] R. Bhaskar *et al.* Analyzing and re-engineering business processes using simulation. In *Proceedings of the 1994 Winter Simulation Conference*, pages 1206–1213, 1994.

[4] E. W. Brehm, R. T. Goettge, and F. W. McCaleb. START/ES – An Expert System Tool for Performance and Reliability Analysis. In *Computer Performance Evaluation '92: Modeling Techniques and Tools*, pages 107–119, 1993.

[5] T. Davenport. *Process Innovation: Reengineering Work through Information Technology.* Harvard Busines School Press, 1993.

[6] L. Dowdy and C. Lowery. *P.S. to Operating Systems.* Prentice-Hall, 1993.

[7] M. Hammer and J. Champy. *Reengineering the Corporation: A Manifesto for Business Revolution.* Harper Business, 1993.

[8] H. J. Harrington. *Business Process Improvement: The Breakthrough Strategy for Total Quality, Productivity, and Comptetiveness.* McGraw-Hill, 1991.

[9] T. Jaeger, A. Prakash, and M. Ishikawa. A Framework for the Automatic Improvement of Workflows to Meet Performance Goals. In *Proceedings of the 6th Conference on Tools with Artificial Intelligence*, pages 640–646, 1994.

[10] R. K. Keller *et al.* The Macrotec toolset for CASE-based business modelling. In *IEEE Sixth Int'l Workshop on CASE*, pages 114–118, 1993.

[11] K. P. Lockyer and J. H. Gordon. *An Introduction to Critical Path Analysis.* Pitman, 1991.

[12] R. Medina-Mora, T. Winograd, R. Flores, and F. Flores. The Action Workflow Approach to Workflow Management Technology. In *CSCW 92 Proceedings*, pages 281–288, November 1992.

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

[14] 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.

[15] A. Ravindran, D. T. Phillips, and Solberg J. J. *Operations Research.* John Wiley and Sons, 1987.

[16] E. Rich and K. Knight. *Artificial Intelligence.* McGraw-Hill, 1991.