Using Simulation and Performance Improvement Knowledge for Redesigning Business Processes

Trent Jaeger and Atul Prakash
Software Systems Research Laboratory
Department of Electrical Engineering and Computer Science
University of Michigan, Ann Arbor, MI 48109-2122.
Emails: {jaegert|aprakash}@eecs.umich.edu

Abstract

Recent business improvement methodologies, such as Business Process Reengineering, advocate the redesign of business processes as the primary method to obtain improvement in business performance. Current approaches to process redesign assist the reengineer in identifying problems in meeting business goals, but provide little support in determining how to resolve those problems to meet the desired goals. The number of feasible changes to the business that can potentially resolve the problems is large and it is hard to predict the effect that each change will have on overall business performance. We present a process redesign methodology that helps reengineers to efficiently generate changes that can help meet business performance goals. Based on the methodology, we describe a system that includes: (1) an executable business model for specifying business processes and business performance goals; (2) a simulator that executes these processes to determine actual performance and to help reengineers identify problems in meeting the specified goals; and (3) a performance improvement knowledge base that suggests changes to resolve these problems and meet the specified goals. We demonstrate the use of the process redesign methodology and the system on two non-trivial process redesign examples, and report our experience.

Keywords: Business process redesign, workflows, simulation, knowledge-based systems, search algorithms, knowledge acquisition, parallel program metrics, queuing theory.

1 Introduction

Most recent business improvement methodologies operate under the assumption that the performance [1] of a business is determined by the performance of its constituent business processes. Methodologies, such as Business Process Reengineering [13], Business Process Innovation [7], and Business Process Improvement [14], all advocate the redesign of business processes as the means to obtain overall business improvement. Process redesign generally involves the following sequence of tasks:

[1] The term *performance* in this paper refers to the combination of all measurable dimensions of the business, such as cost, response time, quality, etc.

1. Understand the business processes, resources, and goals
2. Identify business problems
3. Suggest modification options to the business processes/resources
4. Evaluate modification options to determine if they would help meet the desired goals.

First, an understanding of the business processes and goals is necessary to guide the remaining redesign tasks. Next, business problems are identified, so process redesign can be focused on the problems most in need of resolution. Once the major problems have been identified, the reengineers must suggest options for resolving these problems. Processes interact in complex ways, so any modification option needs to be carefully evaluated prior to its implementation.

Current process redesign approaches provide little or no support to the reengineers for generating modification options, so reengineers may waste a significant amount of time pursuing bad options and miss a number of potentially good modification options. In this paper, we describe a process redesign methodology that uses simulation and performance improvement knowledge to assist the reengineers in the generation of modification options to find a process design that satisfies their goals.

Much of the current process redesign is performed using informal reengineering teams (see [7, 11, 13, 35] for illustrative descriptions, [5, 12, 32, 34] for case studies, and [8, 30, 38] for redesign data management systems [2]). Informal process redesign permits the reengineers a great deal of flexibility in decision making, but the complexity of process interactions caused by competition for resources makes it difficult to identify business problems. Thus, the generation of modification options can be misguided. The techniques presented in this paper can be used to augment the reengineering teams approach, by providing a formal basis to prune the search for effective modification options.

Current simulation-based and knowledge-based systems help reengineers to better understand processes, identify certain business problems, and evaluate modification options. However, these systems do not directly assist the reengineer in generating solutions to these problems. In simulation-based redesign [2, 3, 4, 24, 33], the reengineers are required to build a formal, executable model of the business. This model is simulated to help identify performance problems and evaluate suggested modification options. Knowledge-based redesign [25, 37, 39] involves the construction of semantic models that enable the reengineers to assess the semantic impact of modifications. In both of these approaches, the reengineers must suggest the modification options for resolving any business problems. Even given specific problems, the suggestion of modification options is difficult because there are a variety of ways to solve a problem, so it may be hard to identify the better options.

[2] A fairly extensive summary of process redesign MIS publications through 1994 is provided in [1].

In particular, when a modification option eliminates one problem it often exposes another business problem, so finding options that balance these trade-offs appropriately is difficult.

We present a process redesign system (PRS) that supports the reengineers in all the process redesign tasks listed above. This system has a formal, business specification model that can be used to define an executable business system [19]. Simulation of this business system measures its current performance and identifies performance problems. A performance improvement knowledge base represents how the simulation results are used to generate the modification options that are most likely to improve business performance [20, 21]. Also, the PRS enables reengineers to extend the performance improvement knowledge base with domain-specific knowledge [20]. The PRS has a search mechanism that uses the performance improvement knowledge base to automatically generate a sequence of modification options whose predicted performance meets the business's performance goals. Implementation of the modifications in the business is outside the scope of the PRS.

This paper is organized as follows. In Section 2, we define a simple business example that demonstrates some important problems in process redesign. In section 3, we describe a traditional process redesign problem: improving the performance of a process in a sample sales distributorship. In section 4, we formally define the process redesign problem. In Section 5, we review the previous work related to solving the process redesign problem. In Section 6, we describe the PRS architecture. In Section 7, we detail the implementation of the PRS in solving the simple business example. In Section 8, we demonstrate the PRS on the sales distributorship example. In Section 9, we outline conclusions and discuss future work.

2 JiffyBurger Example

The example we will use to analyze the PRS is Dowdy and Lowery's JiffyBurger system [10]. JiffyBurger is a hamburger restaurant run by two employees. The goal of this example is to determine the best process for the two employees to provide hamburgers for their customers. The JiffyBurger employees service customers using two steps: 1) taking a customer order and 2) filling a customer order. Dowdy and Lowery suggest five possible process options for performing these steps:

1. **Twice-As-Fast (TAF)**: The two workers work together on each step. The steps are performed twice-as-fast with two workers than one. The customers wait in a single line for both steps to be completed.

2. **Sequential, Common-Line** (SCL): 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.

3. **Autonomous, Random-Line** (ARL): Each worker can perform both steps for each individual customer. There is one line per worker. The choice of line is made randomly.

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

Table 1: Queuing theory results for the 5 JiffyBurger options where: (1) X is throughput in customers per minute; (2) R is response time in minutes; and (3) W is waiting time in minutes

4. **Autonomous, Shortest-Line** (ASL): Each worker can perform both steps for each individual customer. There is one line per worker. Each customer chooses the shortest of the two lines.

5. **Autonomous, Common-Line** (ACL): Each worker can perform both steps for each individual customer. All customers wait in a single line for the next worker to become free.

Input data for the problems are:

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

Dowdy and Lowery use queuing theory to analyze the performance of the five options for: 1) average throughput; 2) response time; and 3) waiting time. Analytical solutions for the five options are shown in Table 1. The `ACL` option has the shortest waiting time, while the `TAF` option has the best throughput and response time. The choice of which option is best overall depends on the goals of the reengineers (and their management). We will assume that the performance of the `TAF` option represents the reengineers' goal.

An effective PRS should be able to start from any of the five process options and choose an option that meets the reengineers' goal. For example, if the current process is the `SCL`

Name (count)	*Dept*
Sales Mgr (1)	Outside Sales
Sales Person (5)	Outside Sales
Inside Mgr (1)	Inside Sales
Sales Assoc (3)	Inside Sales, Operations
Sales Engr (1)	Inside Sales, Outside Sales
Operations Mgr (1)	Operations
Stock Person (2)	Operations

Table 2: Sales distributorship employees and their departments

option, then the PRS should redesign that process option into either the `TAF` option or another, thus far unknown, option that has equivalent or better performance.

This example is interesting because it is simple, but it presents difficulties for a PRS. Consider the transition from the `SCL` option to the `TAF` option. This transition requires two incremental modifications: (1) convert the `take order` step from 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 either of the intermediate process designs that has one 1-employee step and one 2-employee step performs badly ($R = 3.79$ min. and $W = 2.29$ min. according to the simulator); there is a long waiting time for the employee that performs both steps. There are several other incremental modifications that can modify the business system such that its predicted performance is better than this state, so the PRS needs the knowledge to recognize that a combination of improvements is necessary to find a design that satisfies the goal.

3 Sales Distributorship Example

As a more traditional process redesign example, consider the sample sales distributorship described below (redesign of this example is described in Section 8). This business consists of three interacting departments: 1) `outside sales` which solicits customer orders; 2) `inside sales` which processes customer orders; and 3) `operations` which fills the orders and maintains the inventory. This business has 14 employees that are assigned to the three departments as shown in Table 2.

The employees in these departments perform 11 types of processes that enable the business to sell goods and maintain its inventory. An example is the `make sales call` process which specifies how sales people meet with potential customers to solicit orders. The 11 process types and a brief description of each is provided in Table 3. Exponential distribution of the arrival and service times of all processes is assumed.

The sales distributorship management's goal is to improve the responsiveness of its pro-

Process Name	*Process Function*	A	R
Receive Products	Receive a product delivery	4 ± 1	110
Order Products	Send a product order	24 ± 3	210
Mgmt Review Operations	Review dept. performance	48 ± 4	60
Mgmt Review Inside	Review dept. performance	72 ± 4	317
Mgmt Review Outside	Review dept. performance	48 ± 24	60
Followup Sales Call	Review sales call	4 ± 1	90
Make Sales Call	On-site product pitch	$1/3 \pm 1/6$	328
Make Quotation	Quote the price of a possible order	$11/2 \pm 1$	694
Make Order	Take an order for products	6 ± 4	413
Salesperson Inquiry	Salesperson status inquiry	$1 \pm 1/2$	373
Customer Inquiry	Respond to customer inquiry	$1 \pm 3/4$	434

Table 3: Sales distributorship processes: A is the average arrival rate in hours ($\pm$ the maximum variance); R is the average response time in minutes predicted by the simulator using the resource model in Section 8.1.

cesses with little or no increase in salary costs. According to a simulation of the business [3], some of the processes that handle customer requests have high average response times (see Table 3, column R). Since these processes provide answers to customer questions, slow response times for these processes probably translate into a perception of poor customer service. Since the `make quotation` process has the worst response time, the reengineers choose to focus the redesign problem on that process first.

Generating modification options for redesigning the `make quotation` process is difficult because: (1) the number of reasonable modifications is large and (2) each modification to solve one problem may create new business problems. From the simulation results, it is known that the `inside mgr` and `sales engr` resources are highly utilized, so reduction of their utilizations is necessary. However, there are several ways to reduce the demand on these resources. For example in Table 4, we estimate that there are 31 reasonable simple modifications that can reduce the demand on these two resources. Note that this is not all the possible modifications, but only the ones that intuitively look like they could be effective. Also, if it takes a sequence of 10 operators to meet the goal[4] and each search node has the same number of reasonable choices, then the search space has a size of $\sum_{i=0}^{10} 31^i \simeq 10^{15}$ (ignoring duplicates). The choice among modifications is complicated because each modification can introduce new problems in satisfying the goal. For example, the option to add a resource increases the salary cost of the business. If the cost increase is high enough,

[3] See Section 8.1 for the definitions of the business resources and the `make quotation` process used by the simulation. Space limits prevent the specification of the other business processes.

[4] The solution generated by our system and given in Section 8 requires 10 operators to meet the specified goal.

Operator Type	*Applicable Objects*	*# of instances*
Add Resource	Add a `sales engr` or `inside mgr`	2
Modify Resource	Change skill set for relevant resource types	5 types * 2 skills = 10
Modify Step	Change skills required to complete step	10
Remove Step	Delete 1 of 3 optional steps	3
Add automation	Add 1 of 3 automated systems	3
Modify sub-process	Change which of 3 sub-processes is called	3

Table 4: Operator types and estimate of the number of "reasonable" instances that can reduce the utilization of the `sales engr` and/or the `inside mgr`

then adding a resource may prevent the salary cost goal from being satisfied.

4 Problem Definition

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

- **Definition 1**: 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. A workflow is a formal model for defining the relationships between individual tasks in a process [22, 27].

- **Definition 2**: A *business flow*, $bf \in BF$, is a double, $bf = (w_{init}, W_{bf})$, where w_{init} is the workflow which is initiates the business flow given an external event (e.g., customer request) and W_{bf} is the set of workflows that may be used in bf. A business flow is the formal representation of a business process.

- **Definition 3**: A *step*, $v \in V$, is a triple, $v = (c, SK, pg)$, 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) pg is the process goal satisfied by the execution of the step. Conditional statements are also steps. The result of evaluating a conditional statement determines the next set of steps to execute in the workflow.

- **Definition 4**: 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 named N_w is activated.

- **Definition 5**: A *resource*, $r \in R$, is a quadruple, $r = (N_r, Sk_r, c_r, TB)$, where: (1) N_r is the name of the resource; (2) Sk_r is r's set of skills; (3) c_r is the cost rate per hour for the resource; and (4) TB is a sequence of time blocks that indicate when r is available to perform a step.

- **Definition 6**: A *department*, $d \in D$, is a double, $d = (N_d, R_d)$, where N_d is the name of the department and R_d is the set of resources that are available to a department.

- **Definition 7**: A *business information system*, S is a quintuple, $S = (BF, D, W, T, R)$, where: (1) BF is a set of business flows; (2) D is a set of business departments; (3) W is a set of workflows; (4) T is a set of workflow triggers; and (5) R is a set of resources.

- **Definition 8**: 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 9**: An *operator*, *op*, modifies an instance of the business information system S_i to create a new business information system S_j. A more detailed definition of operators is provided in Section 6.1.

We informally describe the process redesign problem as the search for a process design whose predicted performance satisfies a performance goal. Formally, the process redesign 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}, whose predicted performance satisfies a performance goal, G_S. Implementation of the resultant design in the business is outside the scope of the problem.

Our definition of process redesign is closest to goals of Business Process Innovation [7]. Davenport grants that both radical and incremental process redesign have their place in business improvement. Our problem definition makes no commitment to whether radical or incremental improvement is required. It is the job of the reengineers to determine the aggressiveness of the goals. If aggressive goals are set, then radical redesign, as advocated in Business Process Reengineering, should be undertaken. If less ambitious goals are set, then the process redesign may require only incremental changes, as in Business Process Improvement [14].

The process redesign problem is theoretically complex because: (1) operators interact by clobbering the effects of prior modifications and (2) the number of operators for a large number of steps, triggers, and resources is large. One operator can undo a goal achieved by another operator in a manner analogous to clobbering in non-linear planning [6]. As we saw in the sales distributorship example, the achievement of a response time goal by adding a resource can undo the satisfaction of the cost goal. Because of this interaction, 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 a solution mechanism needs to constrain the search to make the problem tractable. The simulation results and knowledge about how to use these results to select operators are two tools we can use to constrain the search.

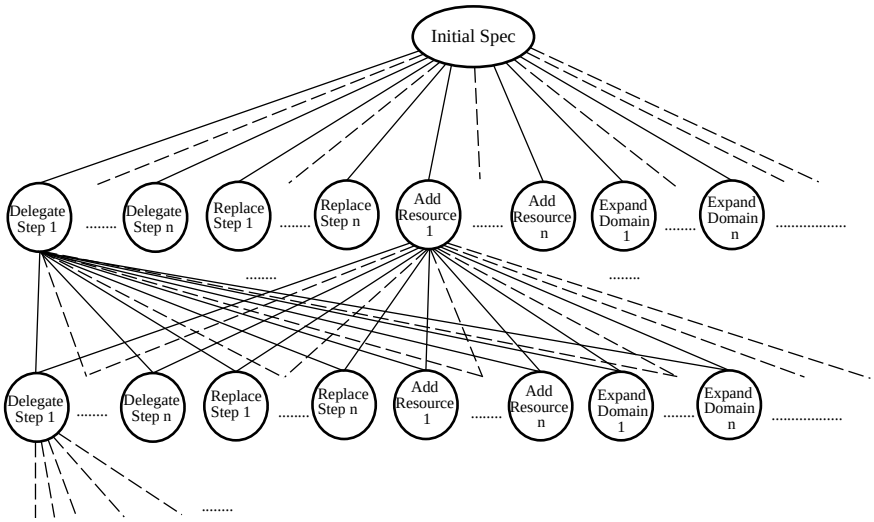

Figure 1: Process redesign problem search space

5 Related Work

In order to generate an operator sequence that is a solution to the process redesign problem, operators which solve current business problems need to be found. Simulation-based and knowledge-based analysis systems identify two important types of problems to solve (or prevent). We describe how these systems have motivated the problem identification capabilities we have built into the PRS.

Simulation-based analysis systems [2, 3, 4, 24, 33] identify performance problems in the business (i.e., primarily problems affecting its response time performance). Identification of performance problems is done using either sensitivity analysis (see [2] for an example) or performance metrics for specific problems (see [4]). Sensitivity analysis involves modifying inputs and measuring the resulting outputs of the system to see what changes make the greatest positive impact. Performance metrics are calculations that estimate the causes of specific problems in the model. Sensitivity analysis can more accurately identify a problem in a model, but is much more time-consuming because a large number of simulations must be run. Performance metrics can be computed after a single simulation, but do not always provide the best order for addressing problems [17]. For example, the critical path metric is not guaranteed to identify the step whose time reduction leads to the greatest reduction in response time. However, this metric does identify problems where there is a high potential for improvement. In our methodology, we use such metrics to help narrow down the search for modification options that are likely to have the most impact in meeting the goals.

Knowledge-based analysis systems [25, 37, 39] assist reengineers in identifying semantic problems in the business, but do not support performance analysis. For example, one system supports the definition of a goal hierarchy for a process [25]. This system then statically evaluates this goal hierarchy to verify that all the goals are being satisfied by the process. Semantic verification is important because reengineers need to verify that the

process design they select satisfies the semantic goals of the business. However, without performance analysis it is difficult to determine whether the process design meets the needs of the business. Also, the models used in these systems, though expressive, can be quite complex for reengineers to build. In our methodology, process goals are specified using performance constraints to motivate process redesign and dependency constraints to prevent semantically invalid changes.

Besides our system, we are aware of only one other system that attempts to support reengineers in the generation of modification options, the BPMS system [23]. Documentation on this system is limited, but our understanding is that the two systems differ in their use of simulation results. BPMS only uses the simulation results to guide the generation of modification options. In contrast, PRS uses simulation results to generate modification options, estimate their effects, and motivate the development of domain-specific knowledge. Estimation techniques are used to determine the likely effects of modification options which enables the better and worse modification options to be identified without the need for additional simulations. Our techniques also motivate the development of domain-specific knowledge during redesign. By integrating knowledge development with the redesign, the reengineers can be better motivated to develop knowledge to solve the current business problems.

In addition, the BPMS system differs from ours in the way it evaluates modification options. The BPMS system uses intelligent agents to guide the search for a process design. Each agent is responsible for improving a single performance parameter, but coordination is controlled by a global optimization over all parameters. Global optimization requires a normalization function that combines the individual parameter values into a single, global optimization value. We have used this technique in the past [21], but found it to be non-intuitive and difficult to control. Therefore, we propose the use of a search mechanism that performs a sequence of single parameter optimizations so that a complex normalization function is not required.

6 System Architecture

The PRS supports a model of process redesign in which the reengineers' main goal is to develop performance improvement knowledge, particularly domain-specific modification options. The PRS can then use this knowledge to automatically find a business information system whose predicted performance satisfies the reengineers' performance goal. The PRS assists the reengineers in the development of this performance improvement knowledge base by: (1) computing simulation results to identify the important performance problems for the reengineers to solve; (2) providing an extensible knowledge base of operator types to assist the reengineers in generating operator instances; and (3) defining interfaces for acquiring domain-specific knowledge from the reengineers.

Simulation measures the performance of business processes in two ways: (1) by user-level performance parameters and (2) by performance metrics. *User-level performance*

1. **Define an initial business system** that defines the initial vertex in the search space.
2. **Set the performance goal** for the business system.
3. **Measure the performance** of the current business system.
4. Unless the performance goal is met, **gather knowledge** about how the initial business information system can be improved.
5. **Generate operators** to continue the search. The generation step includes operator evaluation.
6. **Select the next operator** in the search space using a best-first, simulated-annealing search algorithm.
7. **Repeat** starting at step 3.

Figure 2: Process redesign mechanism

parameters describe the current performance of the process to the reengineers. For example, the average response time of a process is a user-level performance parameter because it is important to a reengineer. *Performance metrics* indicate business system objects that limit the values of the user-level performance parameters. For example, the critical path metric specifies the step that has contributed the most time to the critical path. Reduction of the response time of this step should reduce the average response time of the business flow because the critical path determines the response time.

The PRS provides a knowledge base of operator types and operator meta-knowledge. *Operator types* define the ways that a business system can be modified. Also, operator types store knowledge about using the simulation information to generate individual operators and to estimate their effects. In addition, business information system objects have attributes for the reengineers to specify domain-specific operator options or to specify the semantic requirements of processes. *Operator meta-knowledge* is used to aggregate related operators or trigger the generation of additional operators.

The PRS uses a solution mechanism that both guides the search and acquires performance improvement knowledge from the reengineers. The *process redesign mechanism* (shown in Figure 2) uses a simulated-annealing variation of best-first search [31] to find a business system that satisfies a performance goal. The performance goal can contain multiple user-level performance parameters, so a sequence of single parameter optimizations is used to satisfy the goal. The parameters are ordered based on the importance of a parameter's value to meeting the performance goal. Once a parameter is optimized its value may not become worse than its goal value. Thus, the flexibility for achieving the goal for an important parameter is greater than the flexibility for a less important parameter.

The process redesign mechanism encourages the reengineers to develop domain-specific knowledge for the PRS. The PRS identifies business problems in the performance measurement step (#3) and lists the proposed solutions in the operator generation step (#5).

The business problems are used to motivate the reengineers to provide domain-specific knowledge for resolving these problems in the gather knowledge step (#4). The operator generation information shows the current direction of the search to motivate the user to provide guidance and additional domain-specific knowledge in the select operators step (#6).

7 System Details

In this section, we detail the way that the reengineers use the PRS to solve the process redesign problem for the JiffyBurger example. A prototype PRS has been developed using the Common Lisp Object System (CLOS) [36]. An interface to the system has been constructed using Tcl/Tk [28].

7.1 Knowledge Model

We begin by defining the PRS's knowledge model. Note that the definitions of simple and compound operators below supersede Definition 9.

- **Definition 10**: An *operator type*, ty, is a triple, $ty = (P_{ty}, Fn_c, Fn_e)$, where: (1) P_{ty} is the set of performance parameters whose values can be improved by using an operator of ty; (2) Fn_c is a set of constructor methods for creating instances of the operator type ty to improve a parameters in P_{ty}; and (3) Fn_e is a set of operator effects functions for the parameters changed by operators of that type.

- **Definition 11**: A *simple operator*, op, is a quadruple, $op = (S_i, ty, o, m)$, where: (1) S_i is the i^{th} version of a business information system; (2) ty is the name of the operator type of op; (3) o (one of $v \in V$, $r \in R$, $t \in T$, $w \in W$, $bf \in BF$, 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 12**: A *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 (simple and/or compound operators) that are used to implement the compound operator.

- **Definition 13**: A *constructor method* of an operator type, ty, is a function, $fn_c(p_j, bf, S_i)$, that returns a set of operators of ty that improve p_j for bf in S_i. It is expected that p_j and bf indicate the current goal parameter ge_j being optimized.

- **Definition 14**: An *operator effects function* of an operator type, ty, is a function, $fn_e(p_j, ge_j, op)$, that returns a rational number that is an estimate of 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.

- **Definition 15**: A *modification attribute*, a_m, of a business system object o (one of $bf \in BF, w \in W, d \in D, r \in R, v \in V$, and $t \in T$) is a double, $a_m = (N_a, V_m)$ where: (1) N_a is the name of the attribute and (2) V_m is the set of modification values for the attribute.

- **Definition 16**: A *dependency*, pd, of a process object o (one of $bf \in BF$, $w \in W$, and $v \in V$) is a double, $pd = (o, PG)$ where PG is a process goal statement that must be satisfied before o can begin or complete (depending on the dependency type).

- **Definition 17**: An *operator generation method*, m_{og}, is a function, $m_{og}(OP)$ that generates compound operators OP_c from the current set of generated operators OP.

The *operator types* specify the types of operators that the PRS can generate. The *simple operators* are the operators applied to a single step, resource, or trigger. *Compound operators* represent an aggregation of operators, using simple operators and/or compound operators. Thus, an operator of arbitrary complexity can be generated by the system.

An operator type's *constructor methods* generate instances of that type. A constructor method is selected based on the current goal parameter being optimized. Constructor methods use the performance metrics to select which operator instances are likely to lead to improvement in that goal parameter. The expected effect of an operator on a performance parameter is estimated using the *operator effects functions* of the operator's type. These estimates are used in computing the overall evaluation for the operator. Constructor methods and operator effects functions are fairly complex and domain-independent, so we expect that operator types will be defined a priori by the system.

The *modification attributes* of a business system object store domain-specific options for modifying that object. The values of modification attributes help the constructor methods to find operator choices that are feasible. The *dependencies* represent the process semantics and are used to check if an operator will generate an illegal business system. If an operator generates an illegal business system, it is removed.

Operator generation methods enable further combination of operators via aggregation and implication. An aggregation generation method creates a compound operator that is the grouping of two or more operators. For example, if an operator increases the usage of one skill such that this skill is now the bottleneck, then an operator that reduces the demand on this skill can be aggregated with it. An implication generation method generates a set of operators to support the implementation of an operator. For example, if an operator creates a new skill, then resources need to be added or trained to perform that skill. The newly generated operators are aggregated with the original operator.

7.2 Define Business System

The five business flow options in the JiffyBurger example are implemented in the PRS using three types of workflows: 1) store entry; 2) line entry; and 3) order processing. A

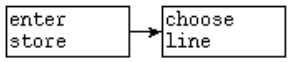

Figure 3: `Buy-a-burger` workflow

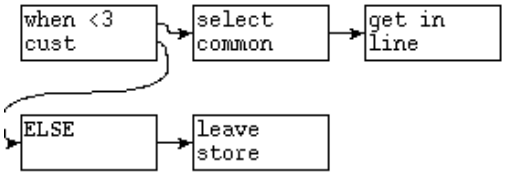

Figure 4: `Common-line` workflow

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 3) represents the entry of a customer into the store. The `TAF`, `SCL`, and `ACL` options all use a single line, so they share the `common-line` workflow (Figure 4) for line entry. The `ASL` and `ARL` options each uses a variation of the `choose-line` workflow (Figure 5) to choose the appropriate line. Order processing is represented by several variations of the `make-burger` workflow (Figure 6).

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 are varied, `take order` and `fill order`, are shown in Table 5. The service time for the `TAF` workflow steps is half of the time of the other options as specified in the problem statement (see Section 2).

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 as well as skills that determine the line to be serviced by that employee. `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).

Legal process designs are specified by order and completion dependencies. An *order dependency* specifies the set of conditions that must be met before a process object can be initiated. These conditions are specified in terms of step goals. In this example, the `fill order` step goal depends on the `take order` step goal being completed first. A *completion dependency* relation specifies the conditions that must be met before a process object can complete. The business flow depends on the `fill order` step goal being completed. These steps cannot be removed by the PRS unless other steps are added to fulfill their goals.

7.3 Initial Knowledge Base

When the reengineer initiates a process redesign, the PRS already contains a set of operator types and operator generation methods that describe the types of modifications

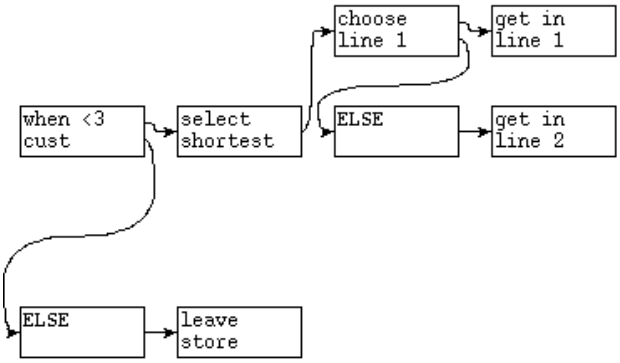

Figure 5: `Choose-line` workflow

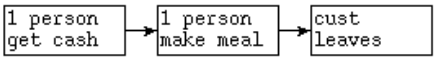

Figure 6: `Make-burger` workflow

that are possible. Modification attributes are also provided so the reengineers can provide knowledge to guide the generation of domain-specific operator instances.

Operator types and the associated objects they modify are shown in Table 6. Definitions of the actions of most of these operators are provided in [20, 21]. Below, we define a representative subset of these operators (grouped by the objects that they modify).

- **Steps**: `Remove step`: Remove an optional step; `Modify step`: Replace a step that achieves a specific process goal with an alternative step.
- **Resources**: `Add resource`: Clone an existing resource; `Train resource`: Train a new skill to an existing resource.
- **Workflows/Business Flows**: `Collaborate`: Increase the number of resources participating in each step of the workflow. The `modify step` operator can be used to revise the specifications of skills and effort for each step.
- **Departments**: `Reduce (utilization)`: Add resources for each type until the utilization of each skill is below a threshold value (0.8 is our current threshold); `Maximize (utilization)`: Remove resources of each type until no more resources can be removed without the required utilization reaching 1.

Operator types include the definitions of their constructor methods and operator effects functions (defined in Section 7.8).

As an example, we define constructor method for the `collaborate` operator to reduce the `average response time` of a business flow (shown in Figure 7). This constructor

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 5: Step definitions for `take order(TO)` and `fill order(FO)`

Steps	*Resources*	*Triggers*	*Flows*	*Departments*
Add	Add/Remove	Add	Pipeline	Reduce
Remove	Train	Remove	Guide	Maximize
Delegate	Focus	Constrain	Collaborate	Train-All
Escalate	Add Automation	Expand	Simplify	Focus-All
Modify	Remove Automation		Parallelize	

Table 6: Operators by business object type (*flows* are business flows and workflows)

method generates a compound operator. This compound operator is an aggregation of a set of simple operators which maximize the resource allotment to the critical path steps that have the highest durations (i.e., sum of waiting time and working time). The attribute `modify-skills` contains the skill options for each step and their estimated durations. For each step, the step skill set with the minimum duration is found. 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.

In addition, the initial knowledge base defines the modification attributes for the business objects. A list of modification attributes is provided in Table 7. Definitions of most of these modification attributes are presented in [20]. Below, we define a few representative attributes:

- **Steps**: `Modify-Skills`: Replace the skills required to execute a step with a new set of skills. A change in a step's skills may also affect the duration to execute the step.
- **Workflows**: `Modify-Steps`: Replace one or more steps with these steps.
- **Resources**: `Add-Skills`: Skills that can be added (e.g., through training) to a resource.

Collaborate constructor for response time (business_flow)

sub_ops	Attribute of `collaborate` that stores its constituent, simple operators
step_skills	The set of skills that resources must provide to complete a *step*
modify_skills	Attribute of *step* containing its *step_skills* options
min_time_skills	The *step*'s skill set option with the minimum average work time

Set *sub_ops* to NULL
Set *steps* to the *business_flow* steps with the highest time contributions to the critical path
For each *step* in *steps*:
    Find the *min_time_skills* from the *modify_skills* attribute of *step*
    If the *min_time_skills* is not the same as the current *step_skills* of *step*:
        Create a new operator *op* to change the *step_skills* of *step* to *min_time_skills*
        Add *op* to *sub_ops*
If no *sub_ops* have been generated, delete `collaborate`

Figure 7: Constructor function for `collaborate` to improve `average response time`

Steps	*Resources*	*Triggers*	*Flows*	*Departments*
Modify-ServiceTime	Add-Skills	Modify-Domain	Add-Steps	Add-Resource
Modify-Skills	Remove-Skills	Step-sizes	Modify-Steps	Modify-Resources
Modify-Design	Modify-Skills	Limits	Remove-Steps	Remove-Resource
Dispensable-P	Modify-Cost		Modify-WF	Add-Workflow
			Remove-WF	Remove-Workflow

Table 7: Modification attributes by business object type (*flows* are business flows and workflows)

- **Departments**: `Add-Resources`: Definitions of resources that can be added.

Operator generation methods specify ways that operators can be aggregated or imply the need for other operators. For example, the operator generation method shown in Figure 8 would aggregate an operator that increases the effort of a highly utilized resource with one that decreases the effort of that resource.

7.4 Set Performance Goal

Once the business system is defined, the reengineer can set the performance goal for the business flow. The performance goal is defined in terms of user-level performance parameters. Since user-level performance parameters are used by the reengineers, parameters selected to be user-level parameters should be well-understood by the reengineers.

The current user-level performance parameters are defined (all parameters are specific to a single business flow type):

Resource demand reduction (OP bf)
For each *op* in OP
  For each *res* which has its demand increased by *op*
    If *res* has a utilization within $n\%$ of the highest utilized resource in bf
      Select *other_op* in OP that reduces the utilization of *res*
        by increasing the utilization of only lower-utilized resources
      Create a new operator that is the aggregation of *op* and *other_op*

Figure 8: Operator generation method that aggregates an operator that increases the demand of an highly utilized resource with an operator that reduces the demand on this resource by increasing the demand on only lower-utilized resources

- **Average response time**: The sum response times of each business flow run divided by the number of runs
- **Average waiting time**: The sum of the waiting time over all steps on the critical path of the business flow divided by the number of runs
- **Average cost**: The sum of the costs of each business flow run divided by the number of runs
- **Value added**: The sum of the value added by the steps executed divided by the number of business flow runs
- **Resource costs**: Sum of the hourly costs of the resources used in any instance of the business flow
- **Automation percentage**: The percentage of a business flow's average cost that is derived from automated activities

To set a goal, the reengineer specifies goal values for user-level parameters. For example, a goal for the `TAF` option is: (1) `average response time` = 1.6 minutes and (2) `average waiting time` = 0.6 minutes. The other user-level attributes are not important for this problem, so we need not set goals for them. The goal is satisfied if the predicted value of each user-level parameter for the business flow is better than (i.e., $>$ or $<$ depending on the parameter) the goal value for that parameter.

Since a sequence of single parameter optimizations are used to find the goal, it may be necessary to set limits on the values that another parameter may take. For example, if `average response time` is being optimized, then the value of resource costs may need to be limited to prevent a high cost solution from being found. Otherwise, it may not be possible to achieve the `resource cost` goal once the response time goal has been met.

7.5 Measure Performance

The PRS measures the performance of the current business information system and presents the major performance issues to the reengineer. The overall performance of the business information system is shown in terms of the user-level performance parameters defined above. Performance metrics identify the major performance problems in the business. The performance metrics measured by the PRS include the following:

- The step in each business flow that accounts for the most time on the critical path [26]
- The step in each business flow that accounts for the most slack on the critical path [18]
- The skill in the business flow with the highest average queue waiting time [29]
- The resources with the highest and lowest utilizations
- The step that accounts for the highest cost in the business flow

This information is presented to motivate the reengineer to develop performance improvement knowledge that could possibly solve these problems. For example, if a reengineer knows that a step spends the most time on the critical path, then the reengineer knows that it is worthwhile to develop new designs for this step and/or to reduce the utilization of the resources that execute that step.

7.6 Gather Knowledge

Once the reengineer reviews the measurement results, the reengineer may be able to provide additional performance improvement knowledge to the PRS. This knowledge is provided in the form of modification attribute values (see the Define Business System Section, 7.3 above). Because the specification of these values can be complex in some instances, we provide interface support for some of their specification.

Observing that the process redesign mechanism does not try the sequence of operators that would generate the `TAF` option from any of the other four options, we would like to define a compound operator that implements this change directly. 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 workflow specification window (see Figure 9). The reengineer first specifes that a new operator to modify a business flow is to be added (not shown). Next, the reengineer selects the workflow to modify. The reengineer then edits that workflow's steps in the workflow specification window to specify the changes in the operator. The commands under the

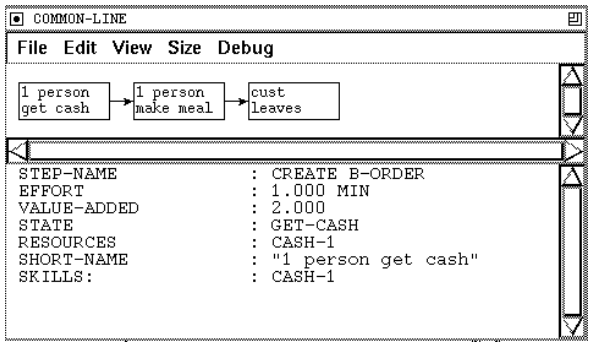

Figure 9: Adding an operator using the workflow specification window: choose the desired step operators under `Edit` menu entry to specify changes to steps.

`edit` menu entry specify the step operators that can be used to edit the individual steps (e.g., `remove step`, `modify step`, etc.). The reengineer can then edit the business flow's other workflows until the resultant specification is generated. The operator option is the sequence of operators used by the reengineer to create the resultant specification. To specify the `create TAF` option, the following step operators are defined:

- 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 duration 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 duration to 0.5 minutes.

7.7 Generate Operators

The PRS generates operators using the following mechanism:

1. The process redesign mechanism improves one goal parameter at a time, so operator types that can improve the performance of that parameter are identified. In this example, `average response time` is the current user-level parameter to be improved. Many operator types, including the workflow operator `collaborate` and the step operator `modify step`, can improve the `average response time` of a business flow.
2. For each of these operator types, its constructor method designed to improve the current goal parameter is executed to generate operators of that type. For example, the constructor function in Figure 7 would be executed to create instances of the `collaborate` operator. Also, the `modify` business flow operator type's constructor method would use the values set in the `modify-workflows` modification attribute in Section 7.6.

3. For each operator that changes a goal of a process object (either by re-ordering, removal, or goal modification of steps), the dependencies on the process object are checked. If a dependency is violated, then the operator is removed. For instance, if an operator reordered the `take order` and `fill order` steps then it would be deleted, since semantically `fill order` is specified to follow `take order`.

4. The operator generation methods (such as the one shown in Figure 8) are now executed to aggregate complimentary operators or to identify the need for new operators, which are then aggregated with the operators that need them.

Operator generation focuses on the creation of operators to address current performance problems. An operator type is selected if it can improve the value of the current performance parameter being optimized. Constructor methods use the information from performance metrics for those sub-standard performance parameters to determine which operators to generate. Other operators may be generated by operator generation methods.

7.8 Evaluate Operators

Operator evaluation estimates the effectiveness of an operator in 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 the value of a user-level performance parameter p_j in the goal element, ge_j, is computed using an *operator effect function*, $fn_e(p_j, ge_j, op_i)$. The global evaluation value is computed by a function that takes into account the estimated values for the current parameter being optimized, previously optimized parameters, and any other parameter that has a limit.

The *operator effects functions* estimate the change in the value of a user-level performance parameter caused by the application of an operator. Operator effects functions are defined for each combination of operator type and user-level performance parameter. Operator effects functions for costs are straightforward because these values are simply sums over all steps. Operator effects functions for `average response time` and `average waiting time` are quite complex, however, because process interactions affect these values. We detail one `average response time` operator effects function below.

Average response time change caused by adding a resource

$fn_e(R, goal(bf_j, R, R_{gj}, fn), add\_resource_i)$ 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 new average waiting time, W'_q, is estimated to be the maximum of the average waiting times for any skill k required by the step. The average waiting time for a skill k, W'_{qk}, is estimated by using a multi-server, Markov queue model that assumes exponentially distributed interarrival and service times (from Hillier and Lieberman [16]):

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

where: (1) P'_{0_k} is the projected probability that no customers are waiting for skill k; (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) $\rho'_k = \lambda_k/\mu'_k s'_k$.

The value for the projected probability that no customers are waiting for skill k, P'_{0_k} is computed by:

$$P'_{0_k} = 1/[\sum_{n=0}^{s'_k-1}(\lambda_k/\mu'_k)^n/n! + (\lambda_k/\mu' k)^{s'_k}/s'_k! \sum_{n=s'_k}^{\infty}(\lambda_k/s'_k\mu'_k)^{n-s'_k}]$$

The value of μ'_k is estimated using the updated utilization and number of resources for skill k.

$$\mu'_k = \lambda_k/s'_k U'_k$$

where U'_k is the updated resource utilization for skill k given the addition of a new resource.

Once all the step response times have been estimated, the modified critical path can be derived. Average step execution times are computed per business flow execution rather than step execution, so loops and conditional blocks can be removed. Using these values, a standard critical path algorithm can be used [9].

The overall evaluation of an operator is based on the evaluation of the operator on the current goal parameter, the effect that the operator has on previously optimized goal parameters, and other goal parameter limits. The value E_n of the current goal parameter n is computing using an operator effects function as described above. The overall evaluation value is computed using the following algorithm.

$$E = \begin{cases} E_n & \text{if } E_i \text{ better than } E_{i,goal} \text{ for } i \text{ from 0 to } n-1 \text{ and} \\ & \text{if } E_i \text{ better than } E_{i,limit} \text{ for } i \text{ from } n+1 \text{ to } m \\ -\infty & \text{otherwise} \end{cases}$$

where: (1) E_n is the evaluation value for goal parameter n; (2) $E_{i,goal}$ is the goal value for goal parameter i; (3) $E_{i,limit}$ is the limit value for the goal parameter i; and (4) m is the number of goal parameters. This evaluation algorithm prevents: (1) the values of previously optimized parameters from becoming worse than their goal and (2) the values of other goal parameters from becoming worse than their limits.

7.9 Select Operators

The PRS uses a best-first search algorithm augmented with a simulated annealing capability [31]. The best-first search algorithm is used because it is the most flexible search algorithm that can utilize the operator evaluations computed in the previous section. In the past [21], we also examined using the A* [15] algorithm, but it is difficult to design a satisfactory mechanism to estimate the distance to the goal because the difference between estimated operator effects is not large enough to differentiate the operators by goal distance. We add a simulated annealing capability to the best-first search algorithm to help avoid local optima. Simulated annealing permits the mechanism to choose a worse state with a probability that decreases as the search progresses.

The simulated annealing, best-first search mechanism consists of the following components:

- The *search problem* 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; (2) 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. k is used to bias the system toward choosing complex operators. 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 by $T = 1/3n$ where n is the number of states examined so far. This simple linear function has performed satisfactorily thus far.

The search algorithm also controls which parameter is being optimized. The optimization of a parameter is complete when the parameter's value satisfies its goal and some fixed number of further improvements have been tried. At present, five further improvements are permitted.

7.10 Results

The `TAF` option is found within two search nodes by the `modify business flow` operator that uses the reengineer's `create TAF` option. Any one of three operators could have been used to find the solution: `modify business flow`, `collaborate`, or the aggregated operator generated by the `resource demand reduction` operator generation method. Another operator with a lower evaluation was chosen in the first search node because the simulated annealing property allows the search algorithm to choose worse options early in the search.

The performance of the PRS shows that: (1) the PRS can acquire and represent the operators necessary to improve this example's performance and (2) the search mechanism can use these operators to find a business flow design whose predicted performance satisfies the performance goal. The first point is justified by the fact that three different operators can generate the `TAF` option. The aggregation methods and the workflow-level `collaborate` operator provide domain-independent ways to create the necessary operator. Also, domain-specific options can be specified by the reengineers which enable the PRS to make changes that it otherwise could not.

The second point is demonstrated by the fact that the system identifies that these operator options are the best options to examine. The operator's constructor methods and operator generation methods generated these operators over other possibilities. In this example, the operator effects functions identified these operator options as having the best `average response time`.

8 Sales Distributorship Process Redesign

Next, we demonstrate the PRS's ability to reengineer the sales distributorship process, `make quotation`. Recall that the `make quotation` business flow represents one of the eleven processes in a sample sales distributorship (see Section 3 for a review of this example).

8.1 Define Business System

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 10).
- **Complex 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 11). 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 12).

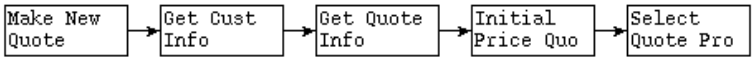

Figure 10: `Create quotation` workflow

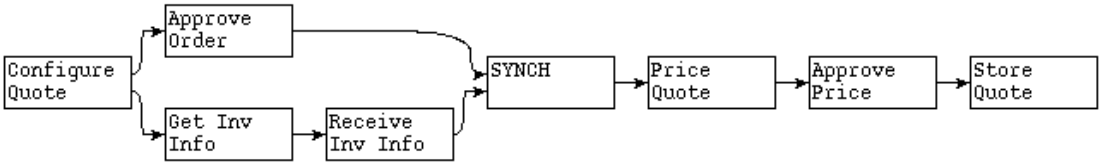

Figure 11: `Complex config quote`, `low effort config quote`, and `high value config quote` workflows

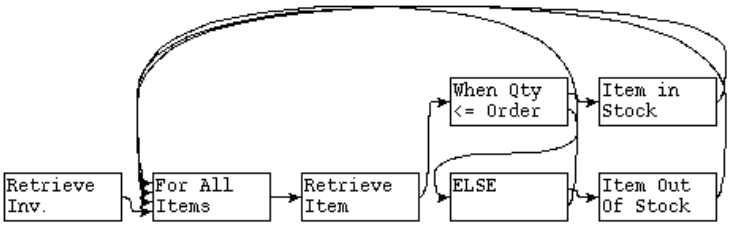

Figure 12: `Check inventory` workflow

Step	*S (min.)*	*V ($)*	*A*	*A %*	*Step Skills*
Make New Quote	1	0	N/A	–	Pricing
Get Cust Info	2	0	N/A	–	Pricing
Get Quote Info	10	0	N/A	–	Pricing
Initial Price Quo	1	0	N/A	–	Pricing
Configure Quote	10/10/60	200	Config Support	25%	Pricing, Prob. Solv.
Approve Order	5/10/20	20	Mgmt Support	20%	Manager
Price Quote	5/30/30	30	DB	67%	Pricing
Approve Price	5/20/20	20	Mgmt Support	20%	Manager
Store Quote	2/2/2	10	DB	50%	Pricing
Retrieve Inv.	4	0	DB	75%	Stock
Retrieve Item	4	0	DB	75%	Stock
Item In Stock?	3	10	N/A	–	Stock
Item Out Of Stock?	3	10	N/A	–	Stock

Table 8: Step performance attributes for the `create quotation`, `low effort config quote`/`high value config quote`/`complex config quote`, and `check inventory` workflows: S is service time in minutes; V is value added in \$; A is the name of the automated system that can be used to assist the resources in completing the step; A% is the percent reduction in service time if the automated system is used; and step skills are the skills required to complete the step

The performance attribute values for each step with a non-zero service time in the five `make quotation` workflows are listed in Table 8. Table 9 shows the resources available to execute steps in the business for all 11 business flows.

The simulator chooses among the `low effort config quote`, `high value config quote`, and `complex config quote` workflows using triggers. An example of the trigger that activates the `high value config quote` workflow is shown in Figure 13. The `step sizes` of the trigger are used by the process redesign 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. The `limits` restrict the extent of the modifications to a trigger.

8.2 Results

Table 10 shows the PRS results for reengineering the `make quotation` business flow. The *initial results* show the performance of the business flow prior to reengineering. The *goal* describes the process's predicted performance desired by the reengineers [5]. The *final results* list the values of the user-level performance parameters for the business flow design that satisfied the goal.

[5]The performance goal may also specify the desired performance of other processes.

Name (count)	*Cost/Hr*	*Skills*
Inside Mgr (1)	$100	Mgr, Problem-solving, Pricing
Sales Mgr (1)	$120	Mgr, Problem-solving, Sales
Operations Mgr (1)	$90	Mgr, Stock, Deliver
Sales Engr (1)	$80	Problem-solving, Pricing, Phone
Sales Assoc (3)	$40	Pricing, Phone
Sales Person (5)	$70	Sales, Phone
Stock Person (2)	$30	Stock, Deliver
DB (1)	$250	Specific Steps
Config Support (1)	$100	Specific Steps
Mgmt Support (1)	$200	Specific Steps

Table 9: Resource definitions for the sales distributorship example (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))
   :limits (quotation->total-cost 200) (quotation->no-of-items 10))
```

Figure 13: `High value config quote` trigger

The resource utilization metric shows that the `sales engr` and `inside mgr` are heavily loaded (with initial utilizations of 0.914 and 0.958, respectively). This leads to high critical path waiting times for the steps executed by these resources (particularly, `approve price` and `approve order` which account for more waiting time than the other steps combined).

After reviewing the performance metrics on the initial run, the reengineer provides the following modification options to the PRS: (1) the `approve order` steps can be removed by setting its `dispensable-p` value to true; (2) the `problem-solving` skill is added as an option in the `modify-skills` attribute of the `configure order` and approval steps to increase the number of resources that can perform these steps; (3) the `sales assocs` are allowed to be trained in the `problem-solving` skill by adding that skill to the `add-skills` attribute of these resources; and (4) a `pricing` resource, such as a `sales assoc`, can collaborate with the `inside mgr` to perform the `approve price` steps by adding the skill options to the `modify-skills` attribute of these steps. All four modification options directly reduce the demand on the `inside mgr` and two also reduce the demand on the `sales engr`.

The performance of the `make quotation` business flow after the PRS has reengineered it is shown in Table 10 (see the *final results* line). The search path taken by the PRS is

Scenario	*R*	*Res. $*	*W*	*V*	*BF $*	*A*
Initial	693.6 min.	$450	628.1 min.	$290	$243.8	0%
Goal (Limits)	150	625 (800)	75	250	200	20
Final Results	132.4	620	70.2	270	191.4	6.0

Table 10: Performance data for the initial, goal, and final `make quotation` process design: R is the `average response time` in minutes; $Res\$$ is the `resource cost` in $; W is the `average waiting time` in minutes; V is the `average value added` $; $BF\$$ is the `average cost` of the business flow in $; and A is the `automation percentage` reduction in business flow costs due to automation

shown in Table 11. The PRS first improves `average response time` by: (1) adding the `mgmt support` system to aid the `inside mgr` in finding the necessary orders and customer statuses in the approval steps (node 2); (2) removing the optional `approve order` steps performed by the `inside mgr` (node 3); (3) adding the `config support` system to aid the `sales engr` in the configuration task (node 4); (4) performing the `approve order` and `store quote` steps in parallel (node 5); and (5) increasing the domain in which the `low value config` workflow applies rather than the other two, more complex workflows (node 7). Nodes 7 through 10 try to further reduce the response time of `make quotation` by: (1) increasing the likelihood that the `low value config` workflow is run (nodes 7 and 10) and (2) delegating the `manager` approval task in the `approve price` step to `problem solving` resources (node 8). The `collaborate` operator is tried multiple times, but with little success. As shown in Table 11, its expected response time is significantly underestimated. At present, the `average response time` estimate for `collaborate` is based on the slack of the effected steps only, so it doesn't take into account the increased waiting time of the `pricing` resources in other steps. The equations given in Section 7.8 for estimating the `average response time` given the addition of a resource would be more appropriate, so we'll fix this in the future.

After node 11, `resource cost` becomes the optimization parameter in the search. It is improved by: (1) removing the `mgmt support` automation and adding an `inside mgr` resource (node 12) and (2) removing a `stock person` resource (node 13). An operator generation method combines a `remove automation` operator with an `add resource` operator for the resource whose demand is increased to create the operator in node 12. At node 13, the goal is satisfied.

9 Conclusions

We have defined a process redesign system (PRS) that uses simulation and performance improvement knowledge to automatically modify business processes until their predicted performance satisfies a goal. Business processes are represented using a formal, executable specification model which enables reengineers to simulate these processes to measure the

# (prev)	*Operator*	*Objects*	*R (Est.)*	*Res. $*
1 (0)	Collaborate	`approve price` steps	721 (306) min.	$450
2 (0)	Add Automation	`mgmt support` system	339 (419)	650
3 (2)	Agg. Remove Steps	`approve order` steps	284 (294)	650
4 (3)	Add Automation	`config support` system	182 (153)	750
5 (4)	Parallelize (Steps)	`app. price/store quote`	133 (110)	750
6 (5)	Collaborate	`approve price` steps	176 (104)	750
7 (5)	Contract Domain	`high value config` trigger	116 (120)	750
8 (7)	Delegate Step	`approve price` step	113 (111)	750
9 (8)	Expand Domain	`high value config` trigger	124 (111)	750
10 (8)	Expand Domain	`low value config` trigger	116 (113)	750
11 (10)	Collaborate	`approve price` steps	115 (110)	750
12 (11)	Rem. Auto. & Add Res.	`mgmt config/inside mgr`	108 (93)	650
13 (12)	Remove Resource	`stock person`	132 (149)	620

Table 11: Search trace of the PRS on `make quotation` process: *Operator* is the operator type; *Objects* list the objects modified; *R* gives the resultant `average response time` (Est. is the estimated `average response time`); and *Res*$ gives the resultant `resource cost` (estimates for *Res*$ are the equal to the simulated value)

current performance of the business processes and identify the likely causes of performance problems. The PRS provides a set of performance improvement operators for modifying the specifications until the stated performance goals are 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. Heuristics based on a set of performance metrics guide the generation and selection of operators. Also, the PRS motivates the reengineers to provide domain-specific knowledge about possible modifications by identifying specific performance problems. From the reengineers' perspective, process redesign is the task of developing performance improvement knowledge. The PRS uses this knowledge to automatically modify business process specifications to meet their performance goal.

The PRS was demonstrated on two examples. In the JiffyBurger example, it was shown that the PRS can represent a variety of knowledge that is useful in reengineering a process such that the process redesign mechanism can avoid locally optimal designs. In the sales distributorship example, we found that the PRS can be used to modify a complex process such that the predicted performance of the resultant design satisfies the reengineers' performance goals.

The PRS's performance depends on the quality of its knowledge base and the accuracy of its metrics. A high quality knowledge base generates only operators that can lead to improvement in the business system. If too few of these operators are generated or too few

bad operators are pruned then the knowledge base needs further development. Knowledge base analysis tools that indicate the quality of the knowledge base may be useful to signal where knowledge development is needed. Also, once an operator is generated, metrics are used to estimate its effect. Further development of performance metrics is necessary to increase the probability that good operators are selected.

References

[1] T. Barothy, M. Peterhans, and K. Bauknecht. Business process reengineering: Emergence of a new research field. *SIGOIS Bulletin*, 16(1):3–10, August 1995.

[2] R. Bhaskar, H. S. Lee, A. Levas, R. Petrakian, F. Tsai, and B. Tulskie. Analyzing and re-engineering business processes using simulation. In *Proceedings of the 1994 Winter Simulation Conference*, pages 1206–1213, 1994.

[3] 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.

[4] D. Bridgeland and S. Becker. Simulation Satyagraha, a successful strategy for business process reengineering. In *Proceedings of the 1994 Winter Simulation Conference*, pages 1214–1220, 1994.

[5] J. R. Caron, S. L. Jarvenpaa, and D. B. Stoddard. Business reengineering at CIGNA corporation: Experiences and lessons learned in the first five years. *MIS Quarterly*, 18:233–250, September 1994.

[6] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32(3):333–377, 1987.

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

[8] A. R. Dennis, R. M. Daniels Jr., G. Hayes, and J. F. Nunamaker. Automated support for business process reengineering: a case study at IBM. In *Proceedings of the 26th Hawaii Int'l Conference on System Sciences*, volume 3, pages 169–178, 1993.

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

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

[11] S. Guha, W. J. Kettinger, and J. T. C. Teng. Business process reengineering: Building a comprehensive methodology. *Information Systems Management*, 10(3):13–22, 1993.

[12] G. Hall, J. Rosenthal, and J. Wade. How to make reengineering really work. *Harvard Business Review*, 71(6):191–131, 1993.

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

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

[15] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on SSC*, 4:100–107, 1968.

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

[17] J. K. Hollingsworth and B. P. Miller. Parallel program performance metrics: A comparison and validation. In *92 SUPERCOMPUTER*, pages 4–13, 1992.

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

[19] T. Jaeger and A. Prakash. BIZSPEC: A Business-Oriented Model for Specification and Analysis of Office Information Systems. In *Proceedings of the 5th Conference on Software Engineering and Knowledge Engineering*, pages 191–198, 1993.

[20] T. Jaeger and A. Prakash. Management and utilization of knowledge for the automatic improvement of workflow performance. In *Proceedings of the 1995 Conference on Organizational Computing Systems*, pages 32–43, 1995.

[21] 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.

[22] S. M. Kaplan, W. J. Tolone, D. P. Bogia, and C. Bignoli. Flexible, active support for collaborative work with conversationbuilder. In *Proceedings of the 1992 Conference on Computer-Supported Cooperative Work*, pages 378–385, 1992.

[23] D. Karagiannis. BPMS: Business process management systems. *SIGOIS Bulletin*, 16(1):10–13, August 1995.

[24] R. K. Keller, R. Lajoie, M. Ozkan, F. Saba, X. Shen, T. Tao, and G. V. Bochmann. The Macrotec toolset for CASE-based business modelling. In *IEEE Sixth International Workshop on Computer-Aided Software Engineering*, pages 114–118, 1993.

[25] J. Lee. Goal-based process analysis: a method for systematic process redesign. In *Proceedings of the 1993 Conference on Organizational Computing Systems*, pages 196–201, 1993.

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

[27] 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.

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

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

[30] G. Rein. Collaboration technology for organizational design. In *Proceedings of the 26th Hawaii Int'l Conference on System Sciences*, volume 3, pages 137–148, 1993.

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

[32] J. F. Rockart and J. D. Hofman. Systems delivery: Evolving new strategies. *Sloan Management Review*, 33(4):21–31, 1992.

[33] R. M. Shapiro. Integrating BPR with image-based work flow. In *Proceedings of the 1994 Winter Simulation Conference*, pages 1221–1228, 1994.

[34] J. E. Short and N. Venkatraman. Beyond business process re-design: Redefining Baxter's business network. *Sloan Management Review*, 34(1):7–21, 1992.

[35] H. A. Smith and J. D. McKeen. Reengineering the corporation: where does I. S. fit in? In *Proceedings of the 26th Hawaii Int'l Conference on System Sciences*, volume 3, pages 120–126, 1993.

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

[37] A. Tsalgatidou and S. Junginger. Modeling in the re-engineering process. *SIGOIS Bulletin*, 16(1):17–24, August 1995.

[38] D. Vogel, R. Orwig, D. Dean, J. Lee, and C. Arthur. Reengineering with Enterprise Analyzer. In *Proceedings of the 26th Hawaii Int'l Conference on System Sciences*, volume 3, pages 127–136, 1993.

[39] E. S. K. Yu and J. Mylopoulos. An actor dependency model of organizational work – with application to business process reengineering. In *Proceedings of the 1993 Conference on Organizational Computing Systems*, pages 258–268, 1993.