Modeling Quality of Service for Workflows and Web Service Processes

Jorge Cardoso[1], Amit Sheth[2], John Miller[2], Jonathan Arnold[3], and Krys Kochut[2]

[1] Departamento de Matemática e Engenharias
Universidade da Madeira
9050-078 Funchal – Portugal
jcardoso@uma.pt

[2] LSDIS Lab, Department of Computer Science
[3] Fungal Genome Resource laboratory, Department of Genetics
University of Georgia
Athens, GA 30602 – USA

Abstract
Workflow management systems (WfMSs) have been used to support various types of business processes for more than a decade now. In workflows or Web processes for e-commerce and Web service applications, suppliers and customers define a binding agreement or contract between the two parties, specifying Quality of Service (QoS) items such as products or services to be delivered, deadlines, quality of products, and cost of services. The management of QoS metrics directly impacts the success of organizations participating in e-commerce. Therefore, when services or products are created or managed using workflows or Web processes, the underlying workflow engine must accept the specifications and be able to estimate, monitor, and control the QoS rendered to customers. In this paper, we present a predictive QoS model that makes it possible to compute the quality of service for workflows automatically based on atomic task QoS attributes. We also present the implementation of our QoS model for the METEOR workflow system. We describe the components that have been changed or added, and discuss how they interact to enable the management of QoS.

1 Introduction

With the advent and evolution of global scale economies, organizations need to be more competitive, efficient, flexible, and integrated in the value chain at different levels, including the information system level. In the past decade, Workflow Management

Systems (WfMSs) have been distinguished due to their significance and their impact on organizations. WfMSs allow organizations to streamline and automate business processes and reengineer their structure; in addition, they increase efficiency and reduce costs.

Several researchers have identified workflows as the computing model that enables a standard method of building Web service applications and processes to connect and exchange information over the Web (Chen, Dayal *et al.* 2000; Leymann 2001; Shegalov, Gillmann *et al.* 2001; Fensel and Bussler 2002). The new advances and developments in e-services and Web services set new requirements and challenges for workflow systems.

One important missing requirement is the management of Quality of Service (QoS). Organizations operating in modern markets, such as e-commerce activities and distributed Web services interactions, require QoS management. Appropriate control of quality leads to the creation of quality products and services; these, in turn, fulfill customer expectations and achieve customer satisfaction.

While QoS has been a major concern in the areas of networking (Cruz 1995; Georgiadis, Guerin *et al.* 1996), real-time applications (Clark, Shenker *et al.* 1992) and middleware (Zinky, Bakken *et al.* 1997; Frolund and Koistinen 1998; Hiltunen, Schlichting *et al.* 2000), few research groups have concentrated their efforts on enhancing workflow systems to support Quality of Service management. Most of the research carried out to extend the functionality of workflow systems QoS has only been done in the time dimension, which is only one of the dimensions under the QoS umbrella. Furthermore, the solutions and technologies presented are still preliminary and limited (Eder, Panagos et al. 1999). The industry has a major interest on the QoS of workflows and workflow systems. Currently, *ad-hoc* techniques can be applied to estimate the QoS of workflows.

For organizations, being able to characterize workflows based on QoS has four distinct advantages.

(1) *QoS-based design.* It allows organizations to translate their vision into their business processes more efficiently, since workflow can be designed according to QoS metrics. For e-commerce processes it is important to know the QoS an application will exhibit before making the service available to its customers.

(2) *QoS-based selection and execution.* It allows for the selection and execution of workflows based on their QoS, to better fulfill customer expectations. As workflow systems carry out more complex and mission-critical applications, QoS analysis serves to ensure that each application meets user requirements.

(3) *QoS monitoring*. It makes possible the monitoring of workflows based on QoS. Workflows must be rigorously and constantly monitored throughout their life cycles to assure compliance both with initial QoS requirements and targeted objectives. QoS monitoring allows adaptation strategies to be triggered when undesired metrics are identified or when threshold values are reached.

(4) *QoS-based adaptation.* It allows for the evaluation of alternative strategies when workflow adaptation becomes necessary. In order to complete a workflow according to initial QoS requirements, it is necessary to expect to adapt, replan, and reschedule a workflow in response to unexpected progress, delays, or technical conditions. When adaptation is necessary, a set of potential alternatives is generated, with the objective of changing a workflow as its QoS continues to

meet initial requirements. For each alternative, prior to actually carrying out the adaptation in a running workflow, it is necessary to estimate its impact on the workflow QoS.

This paper is composed of two parts. The first part presents a comprehensive model for the specification of workflow QoS as well as methods to compute and predict QoS. We start by investigating the relevant QoS dimensions that are necessary to correctly characterize workflows. We not only target the time dimension, but also investigate other dimensions required to develop a usable workflow QoS model. Once the QoS model is defined, algorithms are necessary to compute the QoS of workflows. Quality metrics are associated with tasks, and tasks compose workflows. The computation of workflow QoS is done based on the QoS of the tasks that compose a workflow.

The second part of this paper describes the enhancements that need to be made to workflow systems to support processes constrained by QoS requirements. The enhancements include the implementation of a QoS model, the implementation of algorithms to compute and predict workflow QoS, and the implementation of methods to record and manage QoS metrics. These enhancements have been carried out for the METEOR system (Kochut, Sheth et al. 1999) to allow the specification, recording, and computation of QoS. The support of QoS requires the modification and extension of several workflow system components, and the development of additional modules. While the implementation was made for the METEOR system and the development is based on a specific conceptual model, the main ideas presented in this study can be applied to the vast majority of workflow systems available.

This paper is structured as follows. Section 2 describes a workflow process that illustrates a real world scenario, which will be used to exemplify QoS through the rest of the paper. Based on our scenario, a set of new requirements is derived and the current limitations of WfMSs technology are stated. In section 3, we introduce our workflow QoS model and describe each of its dimensions. Section 4 describes how the quality of service of workflow tasks is calculated. Section 5 described how QoS estimates are set. In Section 6, we present an algorithm to compute and estimate workflow QoS. Section 7 is extensive and describes the modification of existing workflow system components and the creation of new modules that have been developed to support the workflow QoS management for the METEOR system. Each of the workflow components and new modules are analyzed individually. Section 8 presents an example of how to compute the QoS for the workflow introduced in our initial scenario. Section 9 discusses the related work in the QoS area. Finally, section 10 presents our conclusions.

2 Workflows, Tasks, Web services, and Web processes

Web services and e-services have been announced as the next wave of Internet-based business applications that will dramatically change the use of the Internet (Fabio Casati, Ming-Chien Shan et al. 2001). With the development and maturity of infrastructures and solutions that support e-services, we expect organizations to incorporate Web services as part of their business processes. While in some cases Web services may be utilized in an isolated form, it is natural to expect that Web services will be integrated as part of workflows (Fensel and Bussler 2002). The increasingly global economy requires

advanced information systems such as those supporting multi-enterprise and Web-scale processes. Important developments have already been made with the construction of systems to support workflows (enterprise level), distributed workflows (inter-enterprise and B2B level), and Web processes (global level) (Bussler 2003).

In the QoS model presented in this paper, tasks and Web services can be treated with no difference. Workflow systems require tasks to have a structure which includes information such as task name, formal parameters, relevant data, and invoked applications. Web services include the same kind of information. For example, in METEOR workflow system (Kochut, Sheth et al. 1999), business tasks have been wrapped with CORBA objects to enable a transparent remote invocation. With recent technological developments, a business task can now be wrapped with a Web service interface. One of the advantages of using Web services is to enable easier and greater interoperability and integration among systems and applications.

The analogy drawn between tasks and Web services is also valid for workflows and Web processes. Workflows represent the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules and are made of elements which comprise transitions, logic conditions, data flows, parallel and conditional building blocks, starting and ending points, splits, and joins. Web processes have precisely the same characteristics. These allows us to conclude that Web processes can be viewed as workflows that manage Web services instead of tasks (Cardoso and Sheth 2003).

Therefore, throughout this paper, the term ‘task’ or ‘workflow task’ corresponds to a traditional workflow task or a Web service. It will later become evident that in order for our model to be applied to workflows or Web processes, tasks or Web service only have to adhere to the QoS model.

3 Scenario

The Fungal Genome Resource laboratory (FGR 2002) at the University of Georgia has realized that to be competitive and efficient it must adopt a new and modern information system infrastructure. Therefore, a first step was taken in that direction with the adoption of a workflow management system (METEOR (Kochut, Sheth *et al.* 1999)) to support its laboratory processes (Hall, Miller et al. 2003). Since the laboratory supplies several genome services to its customers, the adoption of a WfMS has enabled the logic of laboratory processes to be captured in a workflow schema. As a result, all the services available to customers are stored and executed under the supervision of the workflow system.

3.1 Workflow Structure

Before discussing this scenario in detail, we review the basis elements of the METEOR workflow model.

A workflow is composed of tasks, networks and transitions. Tasks are represented using circles, networks (sub-workflows) using rounded rectangles, and transitions are represented using arrows. Transitions express dependencies between tasks and are

associated with an enabling probability (p_1, p_2,.., p_n). When a task has only one outgoing transition, the enabling probability is 1. In such a case, the probability can be omitted from the graph. A task with more than one outgoing transition can be classified as an *and-split* or *xor-split*. *And-split* tasks enable all their outgoing transitions after completing their execution. *Xor-split* tasks enable only one outgoing transition after completing their execution. *And-split* tasks are represented with a '*' and *xor-split* tasks are represented with a '+'. A task with more than one incoming transition can be classified as an *and-join* or *xor-join*. *And-join* tasks start their execution when all their incoming transitions are enabled. *Xor-join* tasks are executed as soon as one of the incoming transitions is enabled. As with *and-split* and *xor-split* tasks, *and-join* tasks and *xor-join* tasks are represented with the symbol '*' and '+', respectively. When no symbol is present to indicate the input or output logic of a task, then it is assumed to be an *xor*.

3.2 Workflow Description

Genomic projects involve highly specialized personnel and researchers, sophisticated equipment, and specialized computations involving large amounts of data. The characteristics of the human and technological resources involved, often geographically distributed, require a sophisticated coordination infrastructure to manage not only laboratory personnel and equipment, but also the flow of data generated.

One of the services supplied by the research laboratory is the DNA Sequencing workflow. A simplified version of the DNA Sequencing workflow is depicted in Figure 1.

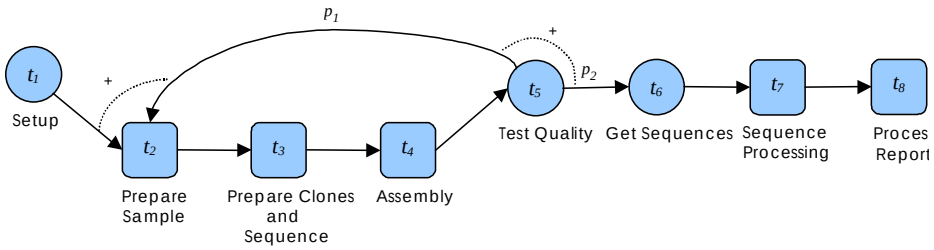

Figure 1– DNA Sequencing workflow

The workflow is composed of eight main tasks: *Setup*, *Prepare Sample*, *Prepare Clone and Sequence*, *Assembly*, *Get Sequences*, *Sequence Processing*, and *Process Report*. Each individual task carries out a particular function; if necessary, the workflow can be spread across multiple research centers.

The *Setup* task is responsible for initializing internal variables of the workflow process.

The second task, *Prepare Sample*, consists of isolating DNA from a biological sample. The samples can be prepared using a variety of protocols. These protocols need to be followed rigorously in order to obtain DNA that is not degraded in any form. A correctly prepared sample will originate a better DNA sequencing, since the quality of the DNA template is one of the most critical factors in DNA sequencing.

The task *Prepare Clones and Sequence* clones specific regions of the genome from DNA isolated in the previous step. This step can be fully automated by computer control (using, for example, a robotic system). This task also executes the sequencing, which uses DNA sequencing machines to read each biochemical "letter" (A, G, C or T) of a cloned DNA fragment. The output is composed of short decoded segments (a sequence such as AGGCATTCCAG…). The use of automated sequencers has revolutionized the field of bioinformatics by enabling scientists to catalogue sequence information hundreds of times faster than was possible with pre-existing scanning techniques. This new approach allows for automatic recognition, without major human intervention.

The *Assembly* task analyzes the DNA segments generated in the sequencing task. This step includes the assembly of larger contiguous blocks of sequences of DNA from small overlapping fragments. This is complicated by the fact that similar sequences occur many times in many places of the genome.

The *Test Quality* task screens for the *Escherichia coli* (*E. coli*) contaminant in DNA contigs. The clones grown in bacterial hosts are likely to be contaminated. A quick and effective way to screen for the *E. coli* contaminant is to compare a given DNA sequence to the *E. coli* genome. For *E. coli*, this task is made easier by the availability of its full genome.

Get Sequences is a simple task that downloads the sequences created in the assembly step, using the FTP protocol.

The *Sequence Processing* task analyzes the DNA segments generated in the assembly step. The goal of this task is to find DNA sequences in order to identify macromolecules with related structures and functions. The new DNA sequence is compared to a repository of known sequences (*e.g.*, Swiss-Prot or GenBank), using one of a number of computational biology applications for comparison.

After obtaining the desired data from the *Sequence Processing* task, the results are stored, e-mailed, and a report is created. The *Process Report* task stores the data generated in the previous task in a database and creates a final report. It is responsible for electronically mailing the sequencing results to the persons involved in this process, such as researchers and lab technicians.

3.3 Workflow Application Requirements

In its normal operation, the Fungal Genome Resource laboratory executes the DNA Sequencing workflow in a regular manner. Workflow instances are started in order to render the sequencing services. In this scenario, and with current workflow technology, the execution of the workflow instances is carried out without any quality of service management on important parameters such as delivery deadlines, reliability, and cost of service. The laboratory wishes to be able to state a detailed list of requirements for the service to be rendered to its customers. Its requirements include the following:

- The final report has to be delivered in 31 weeks or less, as specified by the customer (*e.g.*, NIH).
- The profit margin has to be 10%. For example, if a customer pays $1,100 for a sequencing, then the execution of the DNA Sequencing workflow must have a cost for the laboratory that is less than $1,000.

- In some situations, the client may require an urgent execution of DNA sequencing. Therefore, the workflow has to exhibit high levels of reliability, since workflow failures would delay the sequencing process.

The requirements for the genetic workflow application presented underline three non-functional requirements: time, cost, and reliability. While the specification of such quality requirements is important, current WfMSs do not supply a model to delineate their specification or management.

Having already given a good description of the problem and motivating why a solution is needed for the specification and management of QoS, in the next section we present a QoS model which captures the specification of QoS metrics. This model is a basic stone of our work, and will be used, not only to specify the QoS, but also compute the QoS of workflows.

4 Workflow Quality of Service

Workflow QoS represents *the quantitative and qualitative characteristics of a workflow application necessary to achieve a set of initial requirements*. Quantitative characteristics can be evaluated in terms of concrete measures such as workflow execution time, cost, *etc*. Qualitative characteristics specify the expected services offered by the system, such as security and fault-tolerance mechanisms. QoS should be seen as an integral aspect of workflows; therefore, it should be integrated with workflow specifications. The first step is to define a workflow QoS model.

4.1 Characteristics of the QoS Model

One of the most popular workflow classifications distinguishes between *ad hoc* workflows, administrative workflows, and production workflows. This classification was first mentioned by (McCready 1992). The main differences between these types include structure, repetitiveness, predictability, complexity, and degree of automation.

The QoS model presented here is better suited for production workflows (McCready 1992) since they are more structured, predictable, and repetitive. Production workflows involve complex and highly-structured processes, whose execution requires a high number of transaction accessing different information systems. These characteristics allow the construction of adequate QoS models for workflow tasks. In the case of *ad hoc* workflows, the information, the behavior, and the timing of tasks are largely unstructured, which makes the procedure of constructing a good QoS model more difficult and complex.

4.2 Workflow QoS Model

Quality of service can be characterized according to various dimensions. We have investigated related work to decide which dimensions would be relevant to compose our QoS model. Our research targeted two distinct areas: operations management for organizations and quality of service for software systems. The study of those two areas is

important, since workflow systems are widely used to model organizational business processes, and workflow systems are themselves software systems.

On the organizational side, Stalk and Hout (1990) and Rommel *et al*. (1995) investigated the features with which successful companies assert themselves in competitive world markets. Their results indicated that success is related to the capability to compete with other organizations, and it is based upon three essential pillars: *time*, *cost*, and *quality*. Kobielus (1997) suggests that these dimensions should constitute the criteria that workflow systems should include and might benefit from. On the software system side, Frolund and Koistinen present a set of practical dimensions for distributed object systems' reliability and performance, which include TTR (time to repair), TTF (time to failure), and availability. Chung et al., (2000) present a framework, a set of tools, and methodology to make system design decisions based on analysis non-functional requirements.

Based on previous studies and our experience in the workflow domain, we have constructed a QoS model composed of the following dimensions: *time*, *cost*, and *reliability*. QoS specifications are set for task definitions. Based on this information, QoS metrics are computed for workflows (see section 6).

4.3 Task Time

Time is a common and universal measure of performance. The philosophy behind a time-based strategy usually demands that businesses deliver the most value as rapidly as possible. Shorter workflow execution time allows for a faster production of new products, thus providing a competitive advantage.

The first measure of time is *task response time* (T). Task response time corresponds to the time an instance takes to be processed by a task. The *task response time* can be broken down into two major components: *delay time* and *process time*. **Delay time** (DT) refers to the non-value-added time needed in order for an instance to be processed by a task. This includes, for example, the instance queuing delay and the setup time of the task. While, those two metrics are part of the task operation, they do not add any value to it. **Process time** (PT) is the time a workflow instance takes at a task while being processed; in other words, it corresponds to the time a task needs to process an instance. Therefore, *task response time* for a task *t* can be computed as follows:

$$T(t) = DT(t) + PT(t)$$

The delay time can be further broken down into *queuing delay* and *setup delay*. *Queuing delay* is the time instances spend waiting in a tasklist, before the instance is selected for processing. *Setup delay* is the time an instance spends waiting for the task to be set up. Setup activities may correspond to the warming process carried out by a machine before executing any operation, or to the execution of self-checking procedures. Another time metric that may be considered to integrate with the delay time is the *synchronization delay*, which corresponds to the time a workflow instance waits for other instances in an *and-join* task (synchronization). In our QoS model, this metric is not part of the task response time. This is because the algorithm we use to estimate workflow QoS

can derive this metric directly from the workflow structure and from the task response time. This will become more clear when we describe workflow QoS computation.

Breaking task response time into various pieces is important since it gives a more detailed model to be used by business analysts. Each piece correspond to an important attribute that needs to be analyzed and should not be overlooked. In many situations the different attributes are set by different people.

4.4 Task Cost

Task cost represents the cost associated with the execution of workflow tasks. During workflow design, both prior to workflow instantiation and during workflow execution, it is necessary to estimate the cost of the execution in order to guarantee that financial plans are followed. The cost of executing a single task includes the cost of using equipment, the cost of human involvement, and any supplies and commodities needed to complete the task. The following cost functions are used to compute the cost associated with the execution of a task.

Task cost (C) is the cost incurred when a task *t* is executed; it can be broken down into two major components: *enactment cost* and *realization cost.*

$$C(t) = EC(t) + RC(t)$$

The **enactment cost** (EC) is the cost associated with the management of the workflow system and with the monitoring of workflow instances. The **realization cost** (RC) is the cost associated with the runtime execution of the task. It can be broken down into: *direct labor cost, machine cost, direct material cost,* and *setup cost*. *Direct labor cost* is the cost associated with the person carrying out the execution of a workflow human task (Kochut, Sheth *et al.* 1999), or the cost associated with the execution of an automatic task with partial human involvement. *Machine cost* is the cost associated with the execution of an automatic task. This can correspond to the cost of running a particular piece of software or the cost of operating a machine. *Direct material cost* is the cost of the materials, resources, and inventory used during the execution of a workflow task. *Setup cost* is the cost to set up any resource used prior to the execution of a workflow task.

The EC and RC captures the distinction between the running costs of the workflow system deployment, operation, maintenance and monitoring vs. the costs associated with the execution of tasks.

4.5 Task Reliability

To model the reliability dimension of workflows, we have used concepts from system and software reliability theory (Hoyland and Rausand 1994; Ireson, Jr. *et al.* 1996; Musa 1999). The reliability analysis of systems often uses reliability block diagrams (RBD) as a representation of how the components of a system are connected. Elementary configurations of a RBD include the series and parallel configurations. Our approach is to create a mapping between RBD and workflow structures. This allows us to view a workflow as a system of independent components which can be then modeled and

analyzed using similar functions applied to RBD. The first step is to model the reliability of an individual task.

Task reliability (R) models what can be considered the most important class of workflow failures, task failures (Eder and Liebhart 1996) (also known as activity failures). Task failures can be organized into two main classes: system failures and process failures ((Eder and Liebhart) calls this second type of failures, semantic failures).

System failures. These consist of information technology and software failures which lead to a task terminating abnormally. Information technology and software include operating systems, communication protocols, hardware, etc. For example, a task manager is not able to contact its task because the CORBA server managing the task has failed due to a system breakdown is a system failure.

Process failures. These consist of business process exceptions which lead to an anomalous termination of a task. In a workflow, task structure (Krishnakumar and Sheth 1995) has an initial state, an execution state, and two distinct terminating states. For non-transactional tasks, one of the terminating states indicates that a task has failed, while the other state indicates that a task is done (Figure 2). For transactional and open 2PC tasks, the terminating states are aborted and committed. The model used to represent each task indicates that only one starting point exists when performing a task, but two different states can be reached upon its execution. For example, a database access task fails because of an invalid user password. The task enters the aborted state.

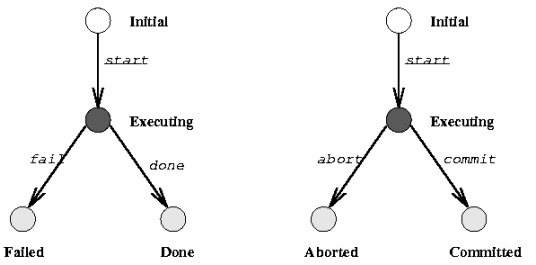

Figure 2 - Two task structures (Krishnakumar and Sheth 1995)

To describe task reliability we follow a discrete-time modeling approach. We have selected this solution since workflow task behavior is most of the time characterized in respect to the number of executions. Discrete-time models are adequate for systems that respond to occasional demands, such as database systems (*i.e*, discrete-time domain). This dimension follows from one of the popular discrete-time stable reliability models proposed in (Nelson 1973) and it is shown below.

$$R(t) = 1 - (\text{system failure rate} + \text{process failure rate})$$

System failure rate is the ratio between the numbers of time a task did not perform for its users and the number of times the task was called for execution, i.e. #(unsuccessful executions)/#(called for execution). *Process failure rate* provides information concerning the relationship between the number of times the state done/committed is reached and the

number of times the failed/aborted state is reached after the execution of a task (see the task model structure shown in Figure 2). It is calculated using the formula #(failed or aborted)/(#(failed or aborted) + #(done or commit)).

Alternatively, continuous-time reliability models can be used when the failures of the malfunctioning equipment or software can be expressed in terms of times between failures, or in terms of the number of failures that occurred in a given time interval. Such reliability models are more suitable when workflows include tasks that control equipment or machines that have failure specifications determined by the manufacturer. Ireson, Jr *et al.* (1996) presents several software reliability models which can be used to model this QoS dimension. The ideal situation would be to associate with each workflow task a reliability model representing its working behavior. While this is possible, we believe that the common workflow system users do not have enough knowledge and expertise to apply such models.

5 Creation of QoS Estimates

In order to facilitate the analysis of workflow QoS, it is necessary to initialize task QoS metrics and also initialize stochastic information which indicates the probability of transitions being fired at runtime. Once tasks and transitions have their estimates set, algorithms and mechanisms, such as simulation, can be applied to compute overall workflow QoS.

5.1 Creation of QoS Estimates for Tasks

Having previously defined the QoS dimensions for tasks, we now target the estimation of QoS metrics of tasks. The specification of QoS metrics for tasks is made at design time and re-computed at runtime, when tasks are executed. During the graphical construction of a workflow process, the business analyst and domain expert set QoS estimates for each task. The estimates characterize the quality of service that the tasks will exhibit at runtime.

Setting initial QoS metrics for some workflow tasks may be relatively simple. For example, setting the QoS for a task controlling a DNA sequencer can be done based on the time, cost, and reliability specifications given by the manufacturer of the DNA sequencer. In other cases, setting initial QoS metrics may prove to be difficult. This is the case for tasks that heavily depend on user input and system environment. For such tasks, it is convenient to study the workflow task based on real operations. The estimates are based on data collected while testing the task. The idea is to test the task based on specific inputs. This can be achieved by the elaboration of an operational profile (Musa 1993). In an operational profile, the input space is partitioned into domains, and each input is associated with a probability of being selected during operational use. The probability is employed in the input domain to guide input generation. The density function built from the probabilities is called the operational profile of the task. At runtime, tasks have a probability associated with each input. Musa (1999) described a detailed procedure for developing a practical operational profile for testing purposes.

The task runtime behavior specification is composed of two classes of information (Table 1): basic and distributional. The basic class associates with each task's QoS dimension the minimum value, average value, and maximum value the dimension can take. For example, the cost dimension corresponds to the minimum, average, and maximum cost associated with the execution of a task. The second class, the distributional class, corresponds to the specification of a constant or of a distribution function (such as Exponential, Normal, Weibull, and Uniform) which statistically describes task behavior at runtime. In some situations it may not be practical to derive a distribution function, an alternative is to sample the distribution and specify it in the form of a histogram rather than an analytical formula. For example, Table 1 and Table 2 show the QoS dimensions for an automatic task (the *SP FASTA* task) and for a manual task (the *Prepare Sample* task; see section 3.2 for tasks descriptions).

	Basic class			Distributional class
	Min value	Avg value	Max value	Dist. Function
Time	0.291	0.674	0.895	Normal(0.674, 0.143)
Cost	0	0	0	0.0
Reliability	-	100%	-	1.0

Table 1 – Task QoS for an automatic task

	Basic class			Distributional class
	Min value	Avg value	Max value	Dist. Function
Time	192	196	199	Normal(196, 1)
Cost	576	576	576	576.0
Reliability	-	100%	-	1.0

Table 2 – Task QoS for a manual task

The values specified in the basic class are typically employed by mathematical methods in order to compute workflow QoS metrics, while the distributional class information is used by simulation systems to compute workflow QoS (Chandrasekaran, Silver et al. 2002; Miller, Cardoso et al. 2002). To devise values for the two classes, the designer typically applies the functions presented in the previous section to derive the task's QoS metrics. We recognize that the specification of time, cost, and reliability is a complex operation, which when not carried out properly can lead to the specification of incorrect values. Additionally, the initial specification may not remain valid over time. To overcome this difficulty, a task's QoS values can be periodically re-computed for the basic class, based on previous executions. The distributional class may also need to have its distribution re-computed. At runtime, the workflow system keeps track of actual values for the QoS dimensions monitored. QoS runtime metrics are saved and used to re-compute the QoS values for the basic class which were specified at design time. The workflow system re-computes the QoS values for each dimension; this allows the system to make more accurate estimations based on recent instance executions.

The re-computation of QoS task metrics is based on data coming from designer specifications and from the workflow system log. Depending on the workflow data available, four scenarios can occur: a) For a specific task *t* and a particular dimension *Dim*, the average is calculated based only on information introduced by the designer (Designer $Average_{Dim}(t)$); b) the average of a task *t* dimension is calculated based on all its executions independently of the workflow that executed it (Multi-Workflow $Average_{Dim}(t)$); c) the average of the dimension *Dim* is calculated based on all the times task *t* was executed in any instance from workflow *w* (Workflow $Average_{Dim}(t, w)$); and d) the average of the dimension of all the times task *t* was executed in instance *i* of workflow *w* (Instance $Average_{Dim}(t, w, i)$). Scenario d) can only occur when loops exist in a workflow.

While the formulae presented only show how to compute average metrics, similar formulae are used to compute minimum and maximum values.

The task QoS for a particular dimension can be determined at different levels; it is computed following the equations described in Table 3.

a)	$QoS_{Dim}(t) =$	Designer $Average_{Dim}(t)$
b)	$QoS_{Dim}'(t) =$	wi_1* Designer $Average_{Dim}(t)$ + wi_2* Multi-Workflow $Average_{Dim}(t)$
c)	$QoS_{Dim}(t, w) =$	wi_1* Designer $Average_{Dim}(t)$ + wi_2* Multi-Workflow $Average_{Dim}(t)$ + wi_3*Workflow $Average_{Dim}(t, w)$
d)	$QoS_{Dim}(t, w, i) =$	wi_1* Designer $Average_{Dim}(t)$ + wi_2* Multi-Workflow $Average_{Dim}(t)$ + wi_3* Workflow $Average_{Dim}(t, w)$ + wi_4* Instance Workflow $Average_{Dim}(t,w, i)$

Table 3 – QoS dimensions computed at runtime

The workflow system uses the formulae from Table 3 to predict the QoS of tasks. The weights wi_k are set manually. They reflect the degree of correlation between the workflow under analysis and other workflows for which a set of common tasks is shared.

The different equations are used based on the historical data available from past executions of tasks and workflows. For example, if the workflow system does not have any historical data in its log describing the QoS metrics of task t_n, then the equation a) will be used to predict a QoS model for task t_n. In the other hand, if the workflow system log's contains historical data describing the QoS metrics of task t_n, then equation b), c) and d) will be used to predict QoS metrics. The section of an equation depends on how much data is available.

Let us assume that we have an instance i of workflow w running and that we desire to predict the QoS of task $t \in w$. The following rules are used to choose which formula to apply when predicting QoS. If task *t* has never been executed before, then formula a) is chosen to predict task QoS, since there is no other data available. If task *t* has been executed previously, but in the context of workflow w_n, and w != w_n, then formula b) is chosen. In this case we can assume that the execution of *t* in workflow w_n will give a good indication of its behavior in workflow *w*. If task *t* has been previously executed in the context of workflow *w*, but not from instance *i*, then formula c) is chosen. Finally, if

task *t* has been previously executed in the context of workflow *w*, and instance *i*, meaning that a loop has been executed, then formula d) is used.

5.2 Probabilities Estimates for Transitions

In the same way we seed tasks' QoS, we also need to seed workflow transitions. Initially, the designer sets the transition probabilities at design time. At runtime, the transitions' probabilities are re-computed. The method used to re-compute the transitions' probabilities follows the same lines of the method used to re-compute tasks' QoS. When a workflow has never been executed, the values for the transitions are obviously taken from initial designer specifications. When instances of a workflow *w* have already been executed, then the data used to re-compute the probabilities come from initial designer specifications for workflow *w*, from other executed instances of workflow *w*, and if available, from the instance of workflow *w* for which we wish to predict the QoS. This corresponds to the use of functions similar to the ones previously defined for tasks' QoS (see Table 3).

The initialization of tasks QoS metrics and the initialization of stochastic information indicating the probability of transitions being fired at runtime give the necessary data to carry out the QoS computation of workflows. The QoS computation is investigated in the next section.

6 Workflow QoS Computation

Once QoS estimates for tasks and for transitions are determined, we can compute overall workflow QoS. We describe a mathematical modeling technique that can be used to compute QoS metrics for a given workflow process.

6.1 Mathematical Modeling

To compute QoS metrics for workflows based on task's QoS metrics we have developed the Stochastic Workflow Reduction (SWR) algorithm (Cardoso 2002). The SWR algorithm repeatedly applies a set of reduction rules to a workflow until only one atomic task (Kochut, Sheth *et al.* 1999) remains. Each time a reduction rule is applied, the workflow structure changes. After several iterations only one task will remain. When this state is reached, the remaining task contains the QoS metrics corresponding to the workflow under analysis.

Graph reduction rules have already been successfully used to verify the correctness of workflows. Sadiq and Orlowska (1999) present an algorithm that employs a set of graph reduction rules to identify structural conflicts in workflows. The algorithm starts by removing all structures from the workflow graph that are correct. This is achieved by iteratively applying a conflict-preserving reduction process. The reduction process eventually reduces a structurally correct workflow to an empty graph. If the workflow is not completely reduced, then structural conflicts exist.

In our approach, the set of reduction rules that can be applied to a given workflow corresponds to the set of inverse operations that can be used to construct a workflow. We

have decided to only allow the construction of workflows which are based on a set of predefined construction systems; this protects users from designing invalid workflows. Invalid workflows contain design errors, such as non-termination, deadlocks, and spliting of instances (Aalst 1999).

Additional reduction rules can be developed. We have decided to present the reduction concept with only six reduction rules, for three reasons. The first reason is because a vast majority of workflow systems support the implementation of the reduction rules presented. A study on fifteen major workflow systems (Aalst, Barros et al. 2000) show that most systems support, the reduction rules presented. The study does not discuss network patterns. The network pattern is intended to provide a structural and hierarchical division of a given workflow design into levels, in order to facilitate its understanding by the grouping of related tasks into functional units. The second reason is that the reduction rules are simple, making it easy to understand the idea behind the workflow reduction process. The last reason is that these rules are supported by the METEOR workflow management system and form a basic set of rules that should be supported by any modern workflow system.

The algorithm uses a set of six distinct reduction rules: (1) sequential, (2) parallel, (3) conditional, (4) fault-tolerant, (5) loop, and (6) network.

Reduction of a Sequential System. Figure 3 illustrates how two sequential workflow tasks t_i and t_j can be reduced to a single task t_{ij}. In this reduction, the incoming transitions of t_i and outgoing transition of tasks t_j are transferred to task t_{ij}.

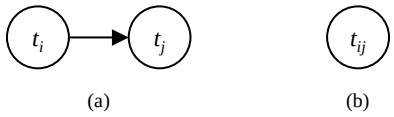

Figure 3 - Sequential system reduction

This reduction can only be applied if the following two conditions are satisfied: a) t_i is not a *xor/and* split and b) t_j is not a *xor*/*and* join. These conditions prevent this reduction from being applied to parallel, conditional, and loop systems. To compute the QoS of the reduction, the following formulae are applied:

$$T(t_{ij}) = T(t_i) + T(t_j)$$
$$C(t_{ij}) = C(t_i) + C(t_j)$$
$$R(t_{ij}) = R(t_i) * R(t_j)$$

Reduction of a Parallel System. Figure 4 illustrates how a system of parallel tasks t_1, t_2, …, t_n, an *and* split task t_a, and an *and* join task t_b can be reduced to a sequence of three tasks t_a, t_{1n}, and t_b. In this reduction, the incoming transitions of t_a and the outgoing transition of tasks t_b remain the same. The only outgoing transitions from task t_a and the only incoming transitions from task t_b are the ones shown in the figure below.

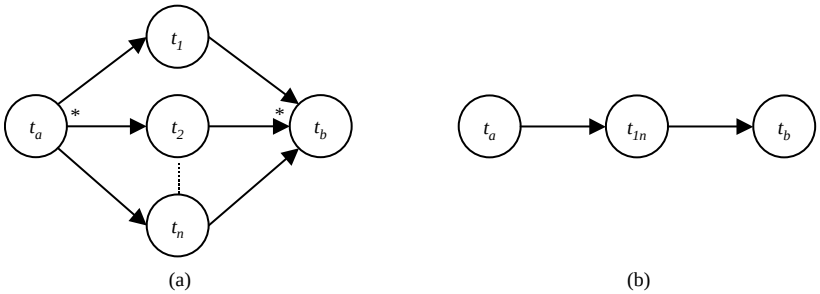

Figure 4 - Parallel system reduction

The QoS of tasks t_a and t_b remain unchanged. To compute the QoS of the reduction the following formulae are applied:

$$T(t_{1n}) = Max_{I \in \{1..n\}} \{T(t_i)\}$$

$$C(t_{1n}) = \sum_{1 \le i \le .n} C(t_i)$$

$$R(t_{1n}) = \prod_{1 \le i \le .n} R(t_i)$$

Reduction of a Conditional System. Figure 5 illustrates how a system of conditional tasks t_1, t_2, …, t_n, a *xor* split (task t_a), and a *xor* join (task t_b) can be reduced to a sequence of three tasks t_a, t_{1n}, and t_b. Task t_a and task t_b do not have any other outgoing transitions and incoming transitions, respectively, other than the ones shown in the figure. In this reduction the incoming transitions of t_a and outgoing transition of tasks t_b remain the same

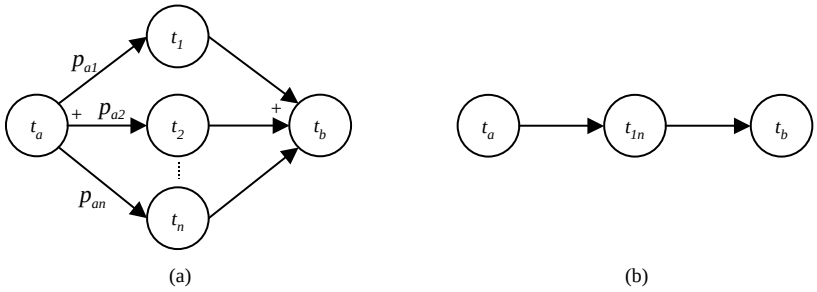

Figure 5 - Conditional system reduction

The QoS of tasks t_a and t_b remain unchanged. To compute the QoS of the reduction the following formulae are applied:

$$T(t_{1n}) = \sum_{1 \le i \le n} p_{ai} * T(t_i)$$

$$C(t_{1n}) = \sum_{1 \le i \le n} p_{ai} * C(t_i)$$

$$R(t_{1n}) = \sum_{1 \le i \le n} p_{ai} * R(t_i)$$

Reduction of a Loop System. Loop systems can be characterized by simple and dual loop systems. Figure 6 illustrates how a simple loop system can be reduced. A simple loop system in task t_i can be reduced to a task t_{li}. In this reduction, $p_i + \sum_{i=1}^{n} p_{oi} = 1$.

Once the reduction is applied, the probabilities of the outgoing transitions of task t_{li} are changed to $p_{lk} = \frac{p_{ok}}{1 - p_i}$, and $\sum_{k=1}^{n} p_{lk} = 1$. In the reduction of a loop system the loop is removed. Since the loop is removed we need to update the remaining outgoing transitions. Therefore, the probability of each outgoing transition needs to be divided by the probability of the loop not being followed (i.e., $1-p_i$).

Figure 6 – Simple loop system reduction

To compute the QoS of the reduction the following formulae are applied:

$$T(t_{li}) = \frac{T(t_i)}{1 - p_i}$$

$$C(t_{li}) = \frac{C(t_i)}{1 - p_i}$$

$$R(t_{li}) = \frac{(1 - p_i) * R(t_i)}{1 - p_i R(t_i)}$$

Figure 7 illustrates how a dual loop system can be reduced. A dual loop system composed of two tasks t_i and t_j can be reduced to a single task t_{ij}. In this reduction,

$p_i + \sum_{i=1}^{n} p_{oi} = 1$. Once the reduction is applied, the probabilities of the outgoing transitions of task t_{ij} are changed to $p_{lk} = \frac{p_{ok}}{1 - p_i}$ and $\sum_{k=1}^{n} p_{lk} = 1$.

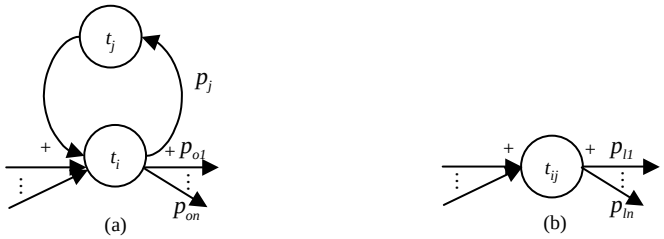

Figure 7 – Dual loop system reduction

To compute the QoS of the reduction the following formulae are applied:

$$T(t_{ij}) = \frac{T(t_i) + T(t_j) - (1 - p_j)T(t_j)}{(1 - p_j)}$$

$$C(t_{ij}) = \frac{C(t_i) + C(t_j) - (1 - p_j)C(t_j)}{(1 - p_j)}$$

$$R(t_{ij}) = \frac{(1 - p_j) * R(t_i)}{1 - p_j R(t_i) R(t_j)}$$

Reduction of a Fault-Tolerant System. Figure 8 illustrates how a fault-tolerant system with tasks t_1, t_2, …, t_n, an *and* split (task t_a), and a *xor* join (task t_b) can be reduced to a sequence of three tasks t_a, t_{1n}, and t_b. The execution of a fault-tolerant system starts with the execution of task t_a and ends with the completion of task t_b. Task t_b will be executed only if k tasks from the set $\{t_1, t_2, \ldots, t_n\}$ are executed successfully. In this reduction, the incoming transitions of t_a and the outgoing transition of tasks t_b remain the same. The idea of this reduction system is to allow several tasks $\{t_1, t_2, \ldots, t_n\}$ to be executed in parallel, carrying out the same function but in a different way, until k tasks have completed their execution. For example, in genomics several algorithms can be used to query genome databases given an initial probe. Let us assume that the tasks t_1, t_2, …, t_5 are execute in parallel and each task executes a distinct algorithm. Using a fault-tolerant system, we can specify that the parallel execution of the tasks continues until two of them complete their execution. In this scenario, we consider that the answers of the first two queries to complete are sufficient for the process to continue.

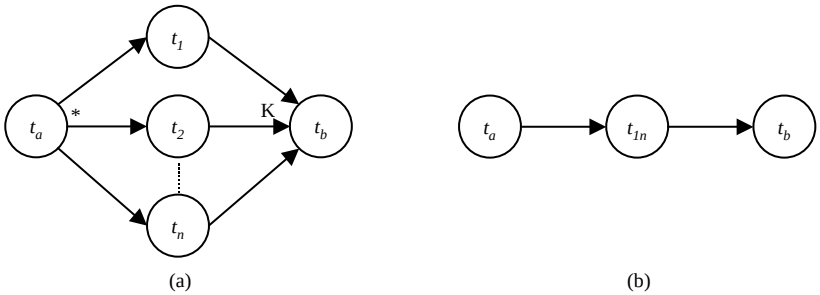

Figure 8 – Fault-Tolerant system reduction

The QoS of tasks t_a and t_b remain unchanged. To compute the QoS of the reduction the following formulae are applied:

The function $\underset{k}{Min}(s)$ selects the set of the k smallest numbers from the set s, and function $g(x)$ is defined as followed:

$$g(x) = \begin{cases} 0, x < 0 \\ 1, x \geq 0 \end{cases}$$

$$T(t_{1n}) = \underset{k}{Min}(\{T(t_1),...,T(t_n)\})$$

$$C(t_{1n}) = \sum_{1 \leq i \leq n} C(t_I)$$

$$R(t_{1n}) = \sum_{i_1=0}^{1} \ldots \sum_{i_n=0}^{1} g(\sum_{j=1}^{n} i_j - k) * ((1-i_1) + (2i_1 - 1)R(t_1)) * \ldots * ((1-i_n) + (2i_n - 1)R(t_n))$$

The formula $R(t_{1n})$ is utilized to compute reliability and corresponds to the sum of all the probabilistic states for which at least k tasks execute successfully.

A fault-tolerant system with n tasks can generate 2^n distinct probabilistic states (the power set). The function $R(t_{1n})$ adds all the probabilistic states that leads to the successful execution of the fault-tolerant system (*i.e.* at least k tasks execute successfully).

In the formula $R(t_{1n})$, the summation over i_1, …, i_n generates all the possible probabilistic states. Each probabilistic state is represented with a binary sequence (i_1 … i_n) for which 0 represents the failing of a task, and 1 represents its success.

For example, in a fault-tolerant system with three parallel tasks (n=3), the values of the indexes i_1=1, i_2=0, and i_3=1 represent the probabilistic state for which tasks t_1 and t_3 succeed and task t_2 fails.

The term $g(\sum_{j=1}^{n} i_j - k)$ is used to indicate if a probabilistic state should be considered in the reliability computation. A probabilistic state is considered only if the number of tasks succeeding is greater or equal to k, *i.e.* $\sum_{j=1}^{n} i_j \geq k$ (or equivalently $\sum_{j=1}^{n} i_j - k \geq 0$). In

our previous example, since i_1=1, i_2=0, i_3=1 and $\sum_{j=1}^{n} i_j = 2$, the probabilistic state (i_1=1, i_2=0, i_3=1) will be only considered if $k \leq 2$.

The reliability of a valid state (i.e., a state for which at least k tasks are executed successfully) is computed based on the product of the reliability of the tasks that compose the state. In our previous example, where i_1=1, i_2=0, i_3=1, and with k=2, the reliability of this state is $g(2\text{-}2)*((1\text{-}\ i_1)+(2i_1\text{-}1)R(t_1))*((1\text{-}\ i_2)+(2i_2\text{-}1)R(t_2))*((1\text{-}\ i_3)+(2i_3\text{-}1)R(t_3))$ which can be reduced to $1*R(t_1)*(1\text{-}R(t_2))*R(t_3)$. This corresponds to the product of the probability of task t_1 to succeed, the probability of task t_2 to fail, and the probability of task t_3 to succeed.

Reduction of a Network System. A network task represents a sub-workflow (Figure 9). It can be viewed as a black box encapsulating an unknown workflow realization with a certain QoS. A network task n_s, having only one task t_i, can be replaced by an atomic task t_j. This reduction can be applied only when the QoS of task t_i is known. In this replacement, the QoS of the atomic task t_j is set to the workflow QoS of the task t_i, *i.e*, $X(t_j) = X(t_i)$, $X \in \{T, C, R\}$.

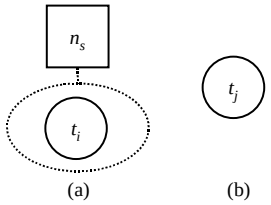

Figure 9 - Network system reduction

The input and output transitions of the network task n_s are transferred to the atomic task t_j.

7 QoS Model Implementation

In the previous sections, we presented a QoS model and the SWR algorithm to address non-functional issues of workflows, rather than workflow process operations. The model and algorithm that we have developed has been implemented for the METEOR workflow management system.

The METEOR project is represented by both a research system (METEOR 2002), and a suite of commercial systems that provide an open system based, high-end workflow management solution, as well as an enterprise application integration infrastructure. The system has been used in prototyping and deploying workflow applications in various domains, such as bio-informatics (Hall, Miller et al. 2003), healthcare (Anyanwu, Sheth et al. 2003), telecommunications (Luo 2000), defense (Kang, Froscher et al. 1999), and university administration (CAPA 1997).

The METEOR system has two enactment engines, ORBWork (Kochut, Sheth et al. 1999) and WEBWork (Miller, Palaniswami et al. 1998). In this section we describe the components that make up the METEOR system and the components that have been modified, extended, and created to enable QoS management in the context of the ORBWork engine.

The work discussed in this paper is part of the research system and is not part of any commercial product yet. It is necessary to make changes to four main components: the Enactment, the Manager, the Builder, and the Repository. These components and their relationship to the overall workflow system are illustrated in Figure 10.

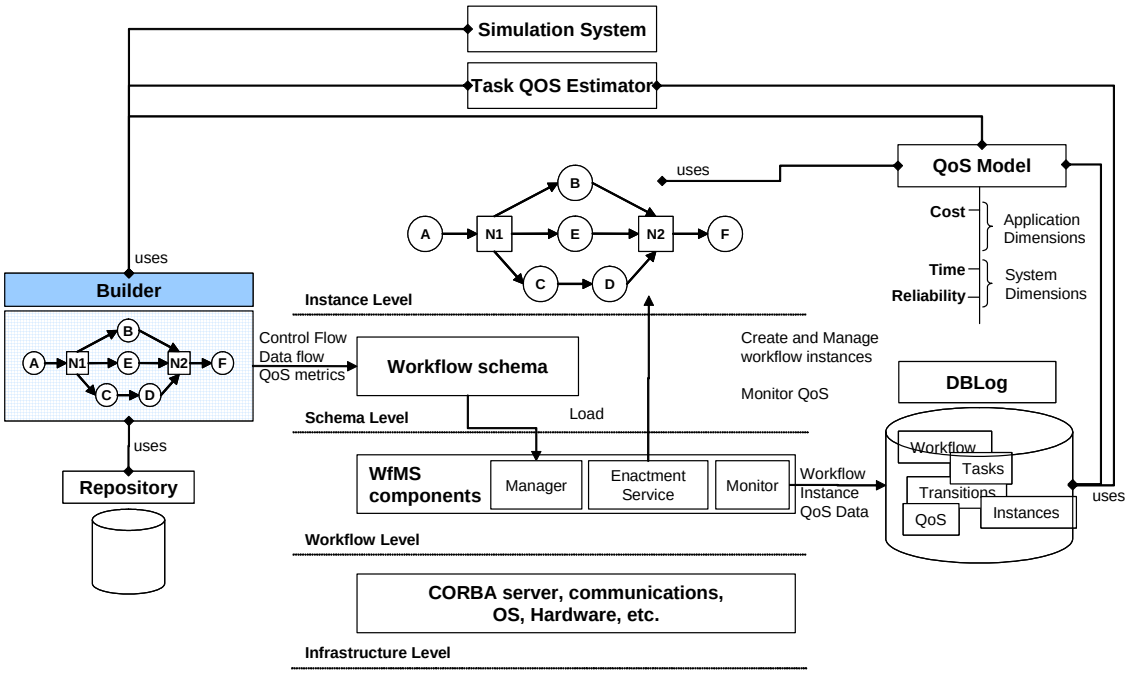

Figure 10 – QoS Management Architecture

7.1 Enactment System

The modifications that have been made to the ORBWork enactment system include alterations to the task schedulers, task managers, tasks, and monitors.

In ORBWork enactment system, task schedulers, and tasks are responsible for managing runtime QoS metrics. From the implementation point of view, we divide the management of the QoS dimensions into two classes: the system and the application class. The dimensions of the system class are managed by system components (*e.g.* a task scheduler), while the dimensions of the applications class are managed by components dynamically created to support a particular workflow application (*e.g.* a task implementation). In our system, the system class includes the time and reliability dimensions, while the application class includes the cost dimension.

Since task schedulers decide the starting time of task execution and are notified of task completion, they are responsible for managing the dimensions of the system class. Task

realizations are the candidate components to manage the cost dimension since they include the necessary functions to dynamically change initial estimates.

7.2 Managing Time

In section 2 we have described task response time (T) as the time an instance takes to be processed by a task. Task response time is composed of two major components: *delay time* (DT) and *process time* (PT). Delay time is further broken down into *queuing delay* (QD) and *setup delay* (SD). This makes the response time of a task *t* represented as followed:

$$T(t) = DT(t) + PT(t) = QD(t) + SD(t) + PT(t)$$

To efficiently manage the time dimension, workflow systems must register values for each of the functions involved in the calculation of task response time (T). Currently, we register values for all the functions, except for the setup delay. The time dimension has its values set according to the task structure illustrated in Figure 11. Each state has been mapped to one of the functions that compose the time dimension. ORBWork system follows this task structure to represent workflow task execution behavior (Krishnakumar and Sheth 1995). To more effectively support QoS management, the original structure has been extended, with the inclusion of the *Pre-Init*, as shown in Figure 11.

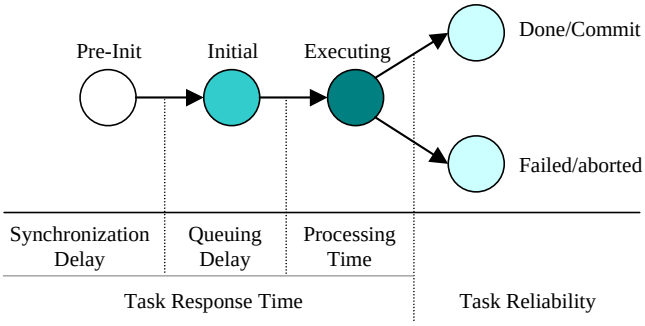

Figure 11 – Revised task structure (extended from (Krishnakumar and Sheth 1995))

The synchronization delay time is calculated based on the difference between the time registered when a task leaves the *pre-init* state and the time registered when it enters the state. A task *t* remains in the *pre-init* state as long as its task scheduler is waiting for another transition to be enabled in order to place the task into an *initial* state. This only happens with synchronization tasks, *i.e. and-join* tasks (Kochut 1999), since they need to wait until all their incoming transitions are enabled before continuing to the next state. For all other types of input and output logic (*xor-split*, *xor-join*, *and-split*) the synchronization delay time is set to zero.

As for the synchronization delay time, the queuing time is the difference between the time a task leaves and enters the *initial* state. A task in the *initial* state indicates that the task is in a queue waiting to be scheduled (by its task scheduler). ORBWork task schedulers treat their queues with a FIFO policy. One interesting queuing policy variation

is associated with the scheduling of human-tasks. For a human-task instance, being in the *initial* state means that the task has been placed in a worklist for human processing. A user can select any human-task in a worklist, as long as the user role matches the task role. In this case, the queuing policy is SIRO (Serve In Random Order). Depending on the workflow system, other useful queuing policies can be used, such as priority queues. When a task instance enters a queue a time-stamp is attached to it. When the task is removed from the queue for scheduling, another time-stamp is attached to it so that the total queuing time can be calculated later. When a task is ready to be executed it transits to the *executing* state. As with the previous calculations, the time a task remains in this state corresponds to the processing time.

Another important time metric is the *synchronization delay* (SyncD). This measure corresponds to the time *and-join* tasks spend waiting for all the incoming transitions to be enabled. The SyncD(t) of a task t is the difference of t_b, the time registered when all the incoming transitions of task t are enabled, and t_a, the time registered when the first incoming transition was enabled, *i.e.* $t_b - t_a$. This measure gives valuable information that can be used to re-engineer business processes to increase their time efficiency.

7.3 Managing Reliability

When a task is ready to execute, a task scheduler activates an associated task manager. The task manager invokes and oversees the execution of the task itself. Once activated, the task manager stays active until the task itself completes. When the task has completed or terminated prematurely with an exception, the task manager notifies its task scheduler.

During a task invocation or realization, a number of undesirable events may occur. Two distinct types of failure may arise (see section 4.5): system failure and process failure. A system failure occurs when the task scheduler is not able to create a task manager or when a task manager is not able to invoke its task. A process failure occurs when an exception is raised during the realization of the task. An exception is viewed as an occurrence of some abnormal event that the underlying workflow management system can detect and react to. If an exception occurs during the realization of a task, it can be placed in the done or fail state (for non-transactional tasks) and commit or abort (for transactional tasks). The former state indicates that the task execution was unsuccessful, while the latter state indicates that a task is executed successfully (Krishnakumar and Sheth 1995).

In our implementation, it is the responsibility of task schedulers to identify the failures of a task invocation or execution in order to subsequently set the reliability dimension. Later this information is used to compute the failure rate, which is the ratio between the number of times the failed/aborted state is reached and the number of times a task was invoked for execution plus the ratio between the number of times the task scheduler is not able to create a task manager or when a task manager is not able to invoke its task and the number of times a task was scheduled for execution by the workflow system.

7.4 Managing the Cost

Task managers are implemented as an object and are classified as transactional or non-transactional, depending on the task managed. Human tasks do not have an associated task manager.

The task manager is responsible for creating and initializing a QoS cost data structure from QoS specifications for the task overseen. When the supervised task starts its execution, the data structure is transferred to it. If the task is a non-transactional one (typically performed by a computer program), a set of methods is available to programmatically manage the initial QoS estimates. No methods are supplied to change the time and reliability dimensions since the task schedulers are responsible for controlling these dimensions. For transactional tasks (*i.e.* a database operation), only the time and reliability dimensions are dynamically set at runtime. The cost dimension, once initialized from the QoS specifications, cannot be changed. This is because database systems do not make available information evaluating the cost of the operations executed. Once the task completes its execution, the QoS data structure is transferred back to the task manager, and later from the task manager to the task scheduler. The only responsibility of the task scheduler will be to incorporate the metrics registered for the time and reliability dimensions (see section 4.2) into the QoS data structure and send it to the monitor to be processed (see next section).

In the case of human tasks (performed directly by end-users), the QoS specifications for the cost dimension is included in interface page(s) (as HTML templates) presented to the end-user. When executing a human task, the user can directly set the cost dimension to values reflecting how the task was carried out. As mentioned previously, human-tasks do not have a task manager associated with them, and therefore a specific task scheduler is responsible for the task supervision. When the task completes its realization, the task scheduler parses the interface page(s) and retrieves the new QoS metrics that the user may have modified.

7.5 Monitor

When workflows are installed and instances are executed, the enactment system generates information messages (events) describing the activities being carried out. The monitor is an independent component represented by an object that records all of the events for all of the workflows being processed by the enactment system.

The DBlog is a suitable interface that the monitor uses to store workflow runtime data in a database. The runtime data generated from workflow installations and instances execution is propagated to the DBlog that will be in charge of storing the information into a specified database.

The data model includes metadata describing workflows and workflow versions, tasks, instances, transitions, and runtime QoS metrics. In addition to storing runtime QoS, we also store designer-defined QoS estimates. The data model captures the information necessary to subsequently run suitable tools to analyze workflow QoS. One of the primary goals of using a database system loosely coupled with the workflow system is to enable different tools to be used to analyze QoS, such as project management and statistical tools.

DBlog is populated when workflows are installed and instances executed. The DBlog schema was designed to store three distinct categories of information, reflecting workflow systems operations with QoS management. The first category corresponds to data events generated when workflows are installed. During installation, information describing workflow structure (which includes tasks and transitions) is stored. The second category of information to be stored corresponds to the QoS estimates for tasks and transitions that are specified at the workflow design phase. The third category corresponds to the information which describes how instances are behaving at runtime. This includes data indicating the tasks' processing time, cost, and the enabling of transitions. The monitoring of transitions is important to build functions which probabilistically describe their enabled rate. The computation of workflow QoS metrics is based on this stochastic structure.

Since the database stores real-time runtime information of tasks QoS metrics, we are also investigating the implementation of mechanisms to automatically notify or alert operators and supervisors when QoS metrics reach threshold values, so that corrective actions can be taken immediately.

7.6 Workflow Builder

The workflow builder tool is used to graphically design and specify a workflow. In most cases, after a workflow design no extra work is necessary and it can be converted automatically to an application by a code generator. The builder is used to specify workflow topology, tasks, transitions (control flow and data flow), data objects, task invocation, roles, and security domains (Kang, Park et al. 2001). During the design phase, the designer is shielded from the underlying details of the runtime environment and infrastructure, separating the workflow definition from the enactment system on which it will be installed and executed. To support workflow QoS management the designer must be able to set estimates for transition probabilities and QoS estimates for tasks. This information is later combined with historical data, which plays a larger role as more instances are executed, to create a runtime QoS model for tasks and a probability model for transitions.

The workflow model and the task model have been extended to support the specification of QoS metrics. To support these extensions, the builder has been enhanced to allow designers to associate probabilities with transitions and to make possible the specification of initial QoS metrics for tasks (see section 5.1). Previously, the workflow model only included data flow mappings associated with transitions. The association of probabilities with transitions transforms a workflow into a stochastic workflow. The stochastic information indicates the probability of a transition being fired at runtime. The QoS model specified for each task and transitions probabilities are embedded into the workflow definition and stored in XML format.

7.6.1 Setting Initial Task QoS Estimates

At design time, each task receives information which includes its type, input and output parameters, input and output logic, realization, exceptions generated, *etc*. All this information makes up the task model. The task model has been extended to accommodate

the QoS model. Task QoS is initialized at design time and re-computed at runtime when tasks are executed. During the graphical construction of a workflow process, each task receives information estimating its quality of service behavior at runtime. This includes information about its cost, time (duration), and reliability.

The task QoS estimates are composed of two classes of information (see section 5.1): basic and distributional. The basic class associates with each task QoS dimension the estimates of the minimum, average, and maximum values that the dimension can take. The second class, the distributional class, corresponds to the specification of a distribution function which statistically describes tasks behavior at runtime. Figure 12 illustrates the graphical interface that is used to specify the basic and distributional information to setup initial QoS metrics.

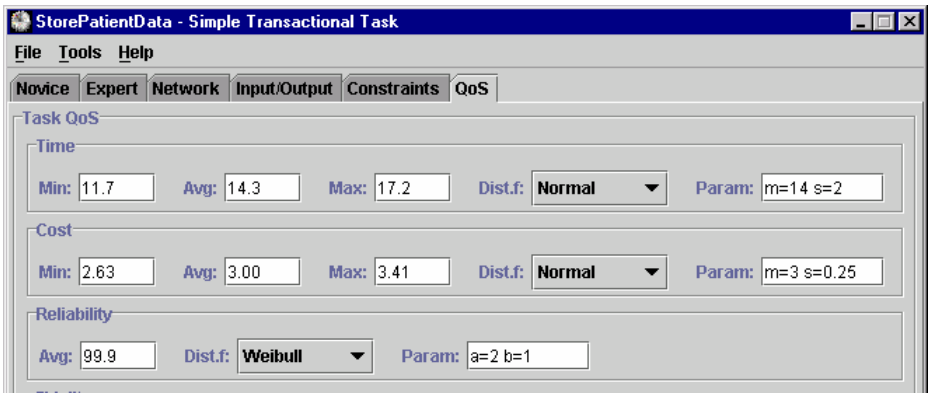

Figure 12 – Task QoS basic and distributional class

The values specified in the basic class are used by mathematical methods, while the distributional class information is used by simulation systems.

Once the design of a workflow is completed, it is compiled. The compilation generates a set of specification files and realization files for each task. The specification files (Spec files) include information describing the control and data flow of each task. The realization files include the operations or instructions for a task to be executed at runtime. For human tasks, HTML files are generated, since they are carried out using a web browser. For non-transactional tasks, java code files are generated and compiled. At runtime, the executables are executed automatically by the enactment system. Finally, for non-transactional tasks a file containing the necessary data to connect to databases is generated. To enable the enactment system to acquire and manipulate QoS information, the builder has been extended to generate QoS specification files for each task. For human tasks we have decided to embed the QoS metrics directly into the HTML forms that are generated.

7.6.2 Re-Computing QoS Estimates

The initial QoS specifications may not be valid over time. To overcome this difficulty we re-compute task QoS values for the basic class, based on previous executions, as described in section 5.1. The same applies for transitions. The distributional class also needs to have its distribution re-computed. This involves the analysis of runtime QoS

metrics to make sure that the QoS distribution functions associated with a task remain valid or need to be modified.

The re-computation of QoS estimates for tasks and for transition probabilities is done based on runtime data generated from past workflow executions that have been stored in the database log. We have developed a QoS Estimator module that lies between the builder and the database log. The QoS Estimator creates a QoS model for tasks based on the information stored in the DBlog. It also calculates transition probability functions based on the transitions enabled at runtime. Figure 13 illustrate the architecture of the QoS Estimator module. When a workflow is being designed, if the tasks selected to compose the workflow have been previously executed, then their QoS metrics are re-computed automatically using the QoS Estimator module.

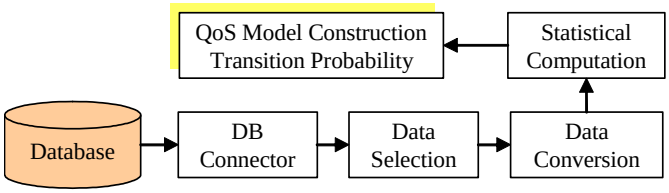

Figure 13 – QoS Estimator Module

DB connector. The DB Connector is responsible for the establishment of a connection to the database. Currently, we support relational databases that implement the JDBC protocol.

Data Selection. The data selection component allows for the selection of task QoS metrics, as defined by the designer and tasks previously executed. Four distinct selection modes exist, and for each one a specific selection function has been constructed. Each function corresponds to one of the functions presented to re-compute QoS estimates for tasks in section 5.1. The component can select tasks QoS metrics from information introduced by the user at design time, from tasks executed in the context of any workflow, from tasks executed in the context of a specific workflow *w*, and from tasks executed from a particular instance *i* of workflow *w*.

Data Convertion. Once a subset of the tasks present in the database log is selected, the data describing their QoS may need to be converted to a suitable format in order to be processed by the Statistical Computation component. The data conversion component is responsible for this conversion. For example, if the processing time of a task is stored using its *start execution date* and *end execution date*, the data conversion component applies the function $f(t) = end\_execution\_date(t) - start\_execution\_date(t)$ to compute the processing time (PT). As another example, let us assume that the reliability of a task is stored in the database using the keywords *done*, *fail*, *commit*, and *abort* (as in ORBWork). In this case, the data conversion component converts the keywords *done* and *commit* to the value 1, indicating the success of the task, and converts the keywords *fail* and *abort* to the value 0, indicating the failure of the task. This abstraction allows the statistical component to be independent from any particular choice of storing runtime information.

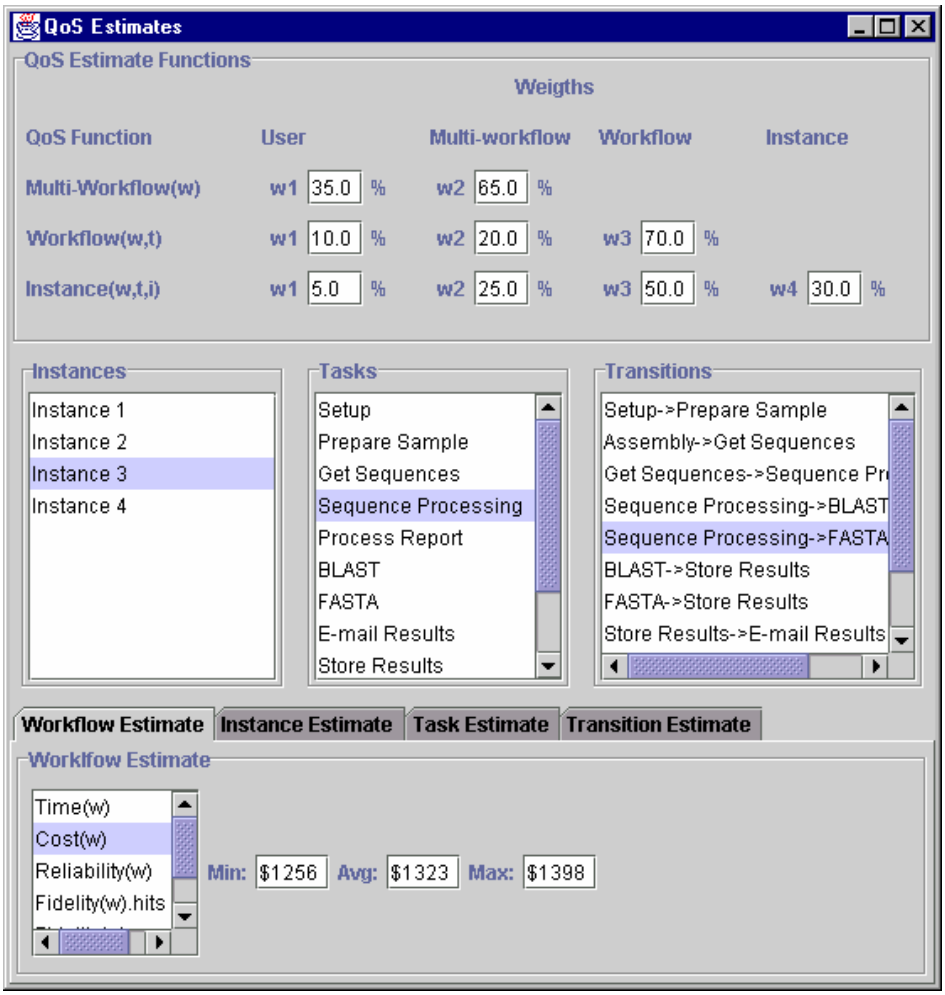

Figure 14 – GUI to calculate QoS estimates

Statistical Computation. Once an appropriate set of tasks has been retrieved from the database and their QoS data has been converted to a suitable format, it is transferred to the statistical computation component to estimate QoS metrics. Currently, the module only computes the minimum, average, and maximum for QoS dimensions, but additional statistical functions can be easily included, such as standard deviations, average deviation, and variance.

Four distinct functions have been developed to compute estimates for the tasks selected in the previous step. Each function is to be used when computing QoS dimensions and corresponds to four scenarios that can occur.

Model Construction. The *QoS Model Construction* component uses the information from the Statistical Computation component and applies a set of functions to re-compute the QoS model (the functions have been presented in Table 3) for each task. Figure 14 shows the graphical user interface available to set the QoS functions and their associated weights, and to visualize the QoS estimates automatically computed for workflows,

instances, tasks, and transitions. The QoS computation is carried out using the SWR algorithm (described in the next section).

8 Workflow QoS Computation Example

The Fungal Genome Resource (FGR) laboratory is in the process of reengineering their workflows. The laboratory technicians, domain experts, and managers have agreed that an alteration to the *Prepare and Sequence* and *Sequence Processing* workflows would potentially be beneficial when sequencing DNA.

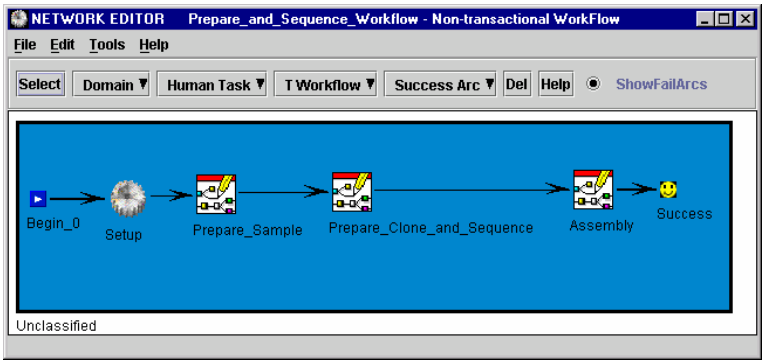

Figure 15 – Prepare and Sequence Workflow

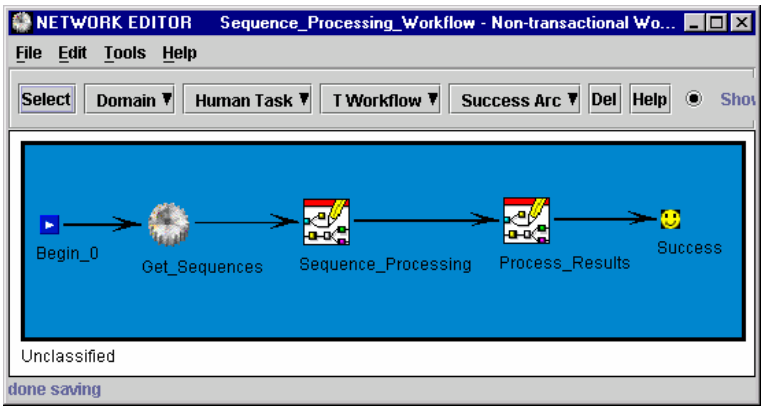

Figure 16 – Sequence Processing Workflow

To improve the efficiency of the processes being managed by the workflow system, the bioinformatics researchers decided to merge the two processes. The researchers noticed that the quality of the DNA sequencing obtained was in some cases useless due to *E. coli* contamination. Additionally, it was felt that it would be advantageous to use other algorithms in the sequence processing phase. Therefore, to improve the quality of the process, the *Test Quality* task and the *SP FASTA* task were added.

Clones grown in bacterial hosts are likely to become contaminated. A quick and effective way to screen for the *Escherichia coli* (*E. coli*) contaminants is to compare the clones against the *E. coli* genome. For *E. coli*, this task is made easier with the availability of its full genome.

The task *SP FASTA* has of the same objective of the task *SP BLAST* (a task of the sequence processing sub-workflow). Both tasks compare new DNA sequences to a repository of known sequences (*e.g.*, Swiss-Prot or GenBank.) The objective is to find sequences with homologous relationships to assign potential biological functions and classifying sequences into functional families. All sequence comparison methods, however, suffer from certain limitations. Consequently, it is advantageous to try more than one comparison algorithm during the sequence processing phase. For this reason, it was decided to employ the BLAST (Altschul, Gish *et al.* 1990) and FASTA (Pearson and Lipman 1988) programs to compare sequences.

The following actions were taken to reengineer the existing workflows:

a) Merge the *Prepare and Sequence* workflow from Figure 15 and the *Sequence Processing* workflow from Figure 16,

b) Add the task *Test Quality* to test the existence of *E. coli* in sequences, and

c) Execute the search for sequences in genome databases using an additional search algorithm (FASTA).

At this point, the alterations to introduce into the processes have been identified. From the functional perspective, the lab personnel, domain experts, and workflow designer all agreed that the new workflow will accomplish the intended objective. The new re-engineered workflow is named *DNA Sequencing.* It is illustrated in Figure 17.

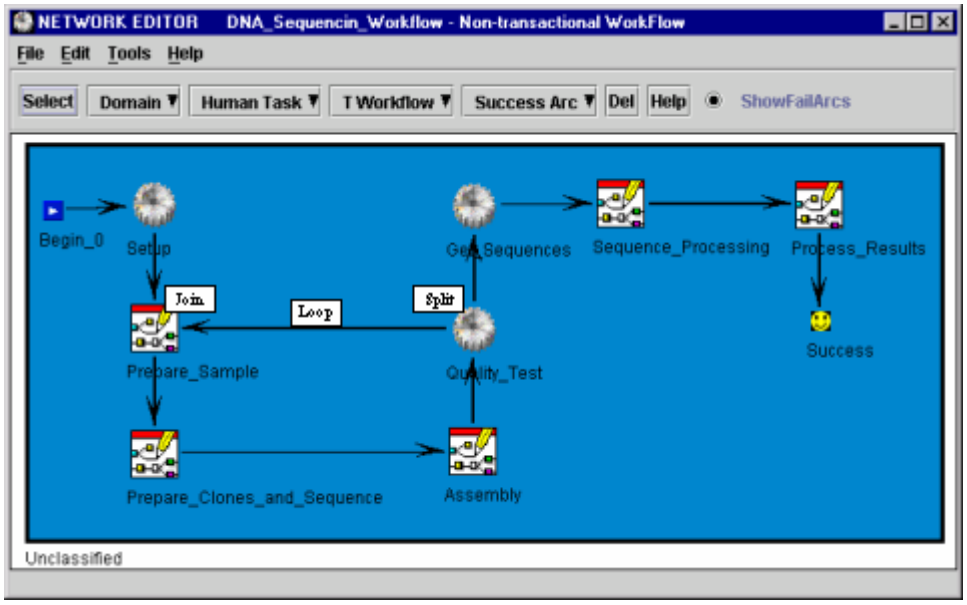

Figure 17 – DNA Sequencing Workflow

8.1 Setting QoS Metrics

While the workflow design meets the functional objectives, non-functional requirements also need to be met. Prior to the execution of the new workflow, an analysis is necessary to guarantee that the changes to be introduced will actually produce a workflow that meets desired QoS requirements, *i.e.*, that the workflow time, cost, and reliability remain

within acceptable thresholds. To accomplish this, it is necessary to analyze the QoS metrics and use the *SWR* algorithm (Cardoso 2002; Cardoso 2002) to compute workflow quality of service metrics.

The first step is to gather QoS estimates for the tasks involved in the *Prepare and Sequence* and *Sequence Processing* workflows. These workflows have been executed several times in the past, and the workflow system has recorded their QoS metrics. The designer QoS estimates have been set using the following methods. For human tasks, the laboratory technicians and researchers have provided estimates for the QoS dimensions. For automated tasks, we have used training sets. For example, for the *SP BLAST* task we have constructed a training set of sequences of different lengths. The sequences have been processed with BLAST, and their QoS has been recorded. For the time dimension, we have used linear regression to predict future metrics (the BLAST algorithm has a linear running time (Altschul, Gish *et al.* 1990).) Equation 1 was used to estimate the BLAST running time to process a sequence:

$$y = a + bx, \quad a = \overline{Y} - b\overline{X} \quad \text{and} \quad b = \frac{n\sum_{i=1}^{n} x_i y_i - \left(\sum_{j=1}^{n} x_j\right)\left(\sum_{k=1}^{n} y_k\right)}{n\sum_{l=1}^{n} x_l^2 - \left(\sum_{m=1}^{n} x_m\right)^2} \quad (1)$$

where x is the independent data (input size) and y is the dependent data (running time). The estimated function is defined as:

$$y = a + bx, \text{ with } a = 78.37, b = 0.0071 \quad (2)$$

For the new tasks introduced (*Test Quality* and *SP FASTA*), no QoS runtime information is available. The only QoS information that can be used to compute the workflow QoS is the one the designer specified at design time. The initial QoS estimates are shown in Table 4.

	Designer Specifications		
Tasks	**T(t)**	**C(t)**	**R(t)**
Quality Test	0.01	$0.0	100%
SP FASTA	9.59	$0.0	100%

Table 4 – Test Quality and FASTA initial QoS estimates

Since the *SP FASTA* task is an automated task, we have used a training set of sequences to derive and set designer QoS estimates. For the time dimension, we have used the linear regression from Equation 1 and defined the function represented in Equation 3 to estimate its duration (FASTA has a linear running time (Pearson and Lipman 1988).)

$$y = a + bx, \text{ with } a = 1061.9, b = 4.11 \quad (3)$$

8.2 Computing QoS Metrics

The domain experts believe that there is a strong agreement between the tasks QoS exhibited during the execution of the *Prepare and Sequence* and the *Sequence Processing* workflows, and the expected QoS of the tasks to be scheduled by the *DNA Sequencing* workflow. This belief is based on the fact that the tasks executed in the two initial workflows will be executed without any change by the newly constructed workflow. The following functions have been utilized to re-compute QoS metrics based on designer and runtime information:

b)	$QoS_{Dim}'(t)$	0.2*Designer $Average_{Dim}(t)$ + 0.8*Multi-Workflow $Average_{Dim}(t)$
c)	$QoS_{Dim}(t, w)$	0.2*Designer $Average_{Dim}(t)$ + 0.2*Multi-Workflow $Average_{Dim}(t)$ + 0.6*Workflow $Average_{Dim}(t, w)$

Table 5 – Re-computation of the QoS dimensions for the DNA Sequencing workflow

To represent the QoS agreement among tasks from different workflows, the domain experts have decided to set the weights according to the following beliefs. For formula b), the domain experts believe that the recorded QoS of tasks previously executed will give good estimates for the execution of tasks scheduled by the new workflow. Thus, the experts set the weights wi_1 and wi_2 of formula b) to 0.2 and 0.8, respectively. The domain experts also believe that as soon as tasks are scheduled by the new workflow, the QoS estimates should rely on the latest QoS data recorded from the *DNA Sequencing* workflow. Also, they consider that when QoS data is available from the *DNA Sequencing* workflow, the importance given to the designer estimates should have the same influence as the QoS estimates recorded for the execution of tasks scheduled by other workflows than the *DNA Sequencing*. Therefore, for formula c), the experts set the weights wi_1, wi_2, and wi_3 to 0.2, 0.2, and 0.6, respectively. In our experiments, we only predict workflow QoS metrics before the execution of workflow, not during workflow execution; thus, we did not to set the weights for formula d) from Table 3.

Since the new workflow has a loop that did not exist in any of the previously executed workflows, it is necessary to estimate the probability of the transition (*Test Quality*, *Prepare Sample*) to be enabled at runtime. Based on prior knowledge of sequencing experiments, the researchers calculate that approximately 10% of the DNA sequence will contain *E. coli* bacteria and that thus there is a 10% probability of the loop back transition being enabled.

8.3 Results

We have run a set of ten experiments. Each experiment involved the execution of the SWR algorithm to predict QoS metrics of the *DNA Sequencing* workflow and the actual execution of the workflow. The results are shown for the four QoS dimensions in Figure 18. The diamonds indicate the QoS estimates (moving average) given by the SWR algorithm and the squares indicate empirical runtime metrics. The three dimensions

analyzed and presented in Figure 13 show that the predicted results are a good estimation for the measured ones.

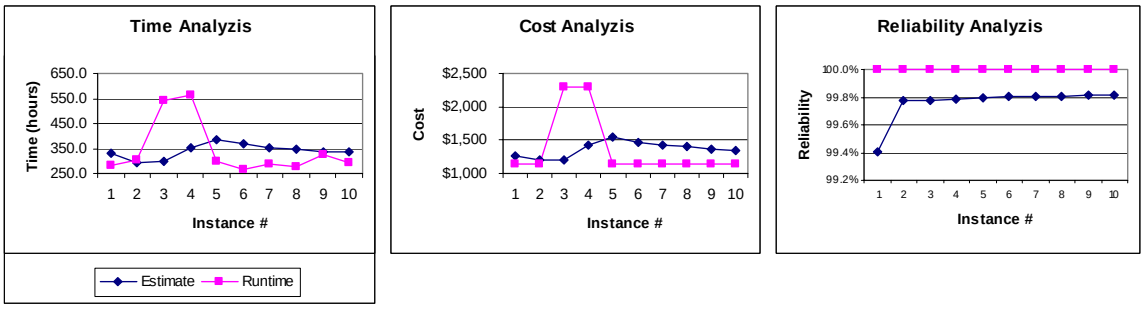

Figure 18 – Experiment results

For the time analysis, the most relevant information that can be interpreted from the chart is the observation that the instances 3 and 4 have registered actual running times that are considerably different from the values estimated. This is due to the topology of the workflow. During the process, it is expected that some DNA sequences will contain *E. coli* contamination. When this happens, re-work is needed, and the first part of the workflow, involving the tasks *Prepare Sample*, *Prepare Clone and Sequence*, and *Assembly*, has to be re-executed. The first part of the workflow takes approximately 99% of the overall workflow execution time. Thus, when *E. coli* contamination is present in a sequence, the time needed to execute the workflow almost doubles. Since it is impossible to know if a DNA sequence will contain *E. coli* or not, the SWR algorithm gives an estimate for instance 3 which is significantly different from the registered values. When instance 4 is executed, the QoS metrics from the previous instance are considered for the QoS estimation. As a result, it can be seen in the chart that the SWR estimation approximates the mean of the recent time metrics recorded, *i.e.* there is an increase of the time estimate in response to the recent increase of workflow time. If more instances detect the presence of *E. coli* contamination, the results of the SWR algorithm for the time dimension will gradually approximate the 550 hours level. When instances number 5 through 10 are executed, they do not detect the presence of contamination in the sequences processed. As a result, the SWR estimates are more accurate, and the estimates start to slowly approximate lower time values.

The costs associated with each task have been provided from technical datasheets describing the DNA Sequencing process. For the cost analysis, the explanation of the results observed follows the same rational as the one provided for the time analysis.

The reliability analysis is relatively easy to interpret. For the first instance executed, the SWR algorithm has used information specified by the designer and derived from task executions from the *Prepare and Sequence* and *Sequence Processing* workflows. The information suggests that the reliability of the new workflow design will be 99.4%. But during our experiments, the ten instances executed never failed. Thus, a 100% reliability value has been registered for each workflow instance. During the instance executions, the reliability estimates given by the SWR algorithm slowly approximate 100%. Nevertheless, it is expected that as the workflow system executes more instances, the reliability of the DNA Sequencing workflow will decrease.

For all the QoS dimensions, the speed of approximation of the SWR algorithm is directly dependent on the weights that have been set for the re-computation of the QoS dimensions (see Table 5 for the weights used in the DNA Sequencing workflow). A higher weight associated with the multi-workflow function implies a faster approximation when the SWR algorithm is applied. The same principal applies to the instance workflow function.

9 Related Work

While a significant amount of QoS research has been done in the areas of networking (Cruz 1995; Georgiadis, Guerin *et al.* 1996), real-time applications (Clark, Shenker *et al.* 1992) and middleware (Zinky, Bakken *et al.* 1997; Frolund and Koistinen 1998; Hiltunen, Schlichting *et al.* 2000), the work found in the literature on quality of service for WfMS is limited. The Crossflow project (Klingemann, Wäsch *et al.* 1999; Damen, Derks *et al.* 2000; Grefen, Aberer *et al.* 2000) has made a major contribution. In their approach, the information about past workflow executions is collected in a log. From this information, a continuous-time Markov chain (CTMC) is derived and used to subsequently calculate the time and the cost associated with workflow executions. An estimation component provides QoS predictions of running workflow instances. These estimates are based on performance models given as CTMC models and produced by the offline monitoring component, which analyzes past executions of workflows, and on the online monitoring component. Compared to the Crossflow project, our approach includes reliability as part of the QoS model. Our model allows for the computation of workflow QoS using two distinct methods: a mathematical and a simulation approach. When the mathematical approach is used, there is no need to define distribution functions. This makes its use simple for business analysts and domain experts. The simulation approach allows the association of distributions with workflow activities (Chandrasekaran, Silver et al. 2002). Not only exponential functions can be employed, as with the Crossflow, but any distribution made available by the simulation system can be used.

While the research on QoS for WfMS is limited, the research on time management, which is under the umbrella of workflow QoS, has been more active and productive. Gillmann *et al.*, (Gillmann, Weissenfels *et al.* 2000; Gillmann, Weikum *et al.* 2002) present a tool for the configuration of distributed workflow systems in order to meet specified goals for throughput, response time, availability, and performability. Their approach is based on continuous-time Markov chains and Markov reward models to predict the performance, availability, and performability of a WfMS under a given load. The performance model estimates the throughput of workflow instances and the waiting time for service requests. The availability model estimates the downtime of the a WfMS given the failure and restart rates for the various components. The performability model predicts the performance taking into account temporarily non-available servers. The use of Markov models makes this approach very similar with the Crossflow system.

Eder *et al.* (1999) and Pozewaunig *et al.* (1997) present an extension of CMP and PERT frameworks by annotating workflow graphs with time, in order to check the validity of time constraints at process build-time and instantiation-time, and to take pre-emptive actions at run-time. Their approach is only applicable to directed acyclic graphs

(DAG). While DAGs can be extended with a special construct that formally maintains the overall structure of a graph to support loops, this has not been contemplated. This is a significant limitation since many of workflows have cyclic graphs. Cycles are, in general, used to represent re-work actions or repetitive activities within a workflow. Our approach deals with acyclic workflows as well as with cyclic workflows.

Researchers at Ulm (Reichert and Dadam 1998; Dadam, Reichert *et al.* 2000) also recognize that time is an important aspect of workflow execution. The ADEPT project includes the modeling of real-time deadline constraints and the consequences of missing deadlines in the case of structural changes of a workflow instance during its execution. With each workflow task, minimal, and maximal durations may be specified. The system only supports the specification and monitoring of deadlines. The monitoring system notifies users when deadlines are going to be missed. There is no provision for the estimation of QoS metrics. Bauer and Dadam (Bauer and Dadam 2000) also show how a distributed WfMS can be developed to minimized the communication load of the components at run time. Their approach uses a cost model and a distribution algorithm to calculate an appropriate variable server assignment expressions at build time.

Chandra, Gong et al. (2003) use dynamic resource allocation techniques to provide guarantees to web applications running on shared data centers. The authors use a system architecture that combines online measurements with prediction and resource allocation techniques. Their work can be paired with our QoS model and the techniques can be use to compute, predict, and analyze the time dimensions of Web services and workflow.

Marjanovic and Orlowska (1999) describe a workflow model enriched with modeling constructs and algorithms for checking the consistency of workflow temporal constraints. The rules that regulate the time component of a workflow are modeled by a set of temporal constraints. Three time constraints are specified. A task duration constraint, a deadline constraint, and an interdependent temporal constraint. This last constraint limits when a task should start/finish relative to the start/finish of another task. At build time and at run time the consistency of temporal constraints is verified. Their work mainly focuses on how to manage workflow changes, while accounting for temporal constraints, and do not target the prediction of workflows execution duration.

Other researchers have also identified the need for a QoS process model. A good example is the DAML-S specification (Ankolekar, Burstein *et al.* 2001; DAML-S 2001), which semantically describes business processes (as in the composition of Web services). The use of semantic information facilitates process interoperability between trading partners involved in e-commerce activities. This specification includes constructs which specify quality of service parameters, such as quality guarantees, quality rating, and degree of quality. While DAML-S has identified the importance of Web services and business processes specifications, the QoS model adopted should be significantly improved in order to supply a more functional solution for its users. One current limitation of DAML-S' QoS model is that it does not provide a detailed set of classes and properties to represent quality of service metrics. Their QoS model needs to be extended to allow for a precise characterization of each dimension, and we hope our work can be one of the inputs in that direction. The addition of semantic concepts, such as minimum, average, maximum, and the distribution function associated with a dimension, will allow the implementation of algorithms, for the automatic computation of QoS metrics for processes based on atomic tasks and sub-processes' QoS metrics.

10 Conclusions

The evaluation of how business is conducted, such as with e-commerce, brings a new set of challenges and requirements that need to be explored and answered. Many e-commerce applications are composed of Web services forming workflows. The composition these workflows cannot be undertaken while ignoring the importance of QoS measurements. The correct management of such QoS specifications directly impacts the success of organizations participating in e-commerce and also directly impacts the success and evolution of e-commerce itself.

In this paper, as a starting point, we show the importance of QoS management for workflows and WfMSs. We then present a comprehensive QoS model. This model allows for the description of workflow components from a QoS perspective; it includes three dimensions: time, cost, and reliability. The use of QoS increases the added value of workflow systems to organizations, since non-functional aspects of workflows can be described. The model is predictive; based on the QoS of workflow components (tasks or Web services), the QoS of workflows (networks) can be automatically computed. This feature is important, especially for large processes that in some cases may contain hundreds of tasks. We present a mathematical model that formally describes the formulae to compute QoS metrics among workflow tasks. Based on these formulae, we develop an algorithm (SWR algorithm) to automatically compute the overall QoS of a workflow. The algorithm applies a set of reduction rules to a workflow, until only one task remains which represents the QoS for the entire workflow.

Having a theoretical QoS model, we explain how the model was implemented in the METEOR workflow management system. The objective was to identify the challenges and difficulties that the implementation of QoS faces. The support of QoS management requires the modification and extension of most of workflow system components. This includes the enactment system, the workflow builder (or designer), the monitor, the code generator, the repository, the workflow model, and the task model. Additionally, new components need to be implemented, such as a QoS estimator module to create QoS estimates for tasks and probabilities for transitions. The monitor needs an additional interface so that runtime tasks QoS metrics are propagated and logged into a database for data processing purposes.

To test the validity of the QoS model, the SWR algorithm, and the QoS implementation we have deployed a set of production workflows in the area of genetics at the Fungal Genome Resource laboratory. We executed workflow instances based on real data and the generated QoS data have been collected and analyzed. We have use these settings to collect data points and validate them with the help of scientists but the process itself was not made operational. The analysis of the data indicates that the QoS model and algorithm presented give a suitable framework to predict and analyze the QoS of production workflows.

11 References

Aalst, W. M. P. v. d. (1999). Generic Workflow Models: How to Handle Dynamic Change and Capture Management Information. Proceedings of the Fourth IFCIS International

Conference on Cooperative Information Systems (CoopIS'99), Edinburgh, Scotland, IEEE Computer Society Press. pp. 115-126.

Aalst, W. M. P. v. d., A. P. Barros, et al. (2000). Advanced Workflow Patterns. Seventh IFCIS International Conference on Cooperative Information Systems. pp. 18-29.

Altschul, S. F., W. Gish, et al. (1990). "Basic local alignment search tool." Journal of Molecular Biology **215**: 403-410.

Ankolekar, A., M. Burstein, et al. (2001). DAML-S: Semantic Markup for Web Services. Proceedings of the International Semantic Web Working Symposium (SWWS), Stanford University, California. pp. 39-54.

Anyanwu, K., A. Sheth, et al. (2003). "Healthcare Enterprise Process Development and Integration." Journal of Research and Practice in Information Technology, Special Issue in Health Knowledge Management **35**(2): 83-98.

Bauer, T. and P. Dadam (2000). Efficient Distributed Workflow Management Based on Variable Server Assignments. Advanced Information Systems Engineering, 12th International Conference CAiSE 2000, Stockholm, Sweden. pp. 94-109.

Bussler, C. (2003). B2B Integration: Concepts and Architecture, Springer-Verlag.

CAPA (1997). Course Approval Process Automation (CAPA). Athens, GA., LSDIS Lab, Department of Computer Science, University of Georgia.

Cardoso, J. (2002). Quality of Service and Semantic Composition of Workflows. Department of Computer Science. Athens, GA, University of Georgia**:** 215.

Cardoso, J. (2002). Stochastic Workflow Reduction Algorithm, LSDIS Lab, Department of Computer Science, University of Georgia. http://lsdis.cs.uga.edu/proj/meteor/QoS/SWR_Algorithm.htm

Cardoso, J. and A. Sheth (2003). "Semantic e-Workflow Composition." Journal of Intelligent Information Systems (JIIS). **21**(3): 191-225.

Chandra, A., W. Gong, et al. (2003). Dynamic Resource Allocation for Shared Data Centers Using Online Measurements. Proceedings of the Eleventh International Workshop on Quality of Service (IWQoS 2003), Berkeley, Monterey, CA, Springer. pp. 381-400.

Chandrasekaran, S., G. Silver, et al. (2002). Service Technologies and their Synergy with Simulation. Proceedings of the 2002 Winter Simulation Conference (WSC'02), San Diego, California. pp. 606-615.

Chen, Q., U. Dayal, et al. (2000). Dynamic-Agents, Workflow and XML for E-Commerce Automation. EC-Web. pp. 314-323.

Chung, L., B. Nixon, et al. (2000). Non-Functional Requirements in Software Engineering, Kluwer Academic Publishers.

Clark, D., S. Shenker, et al. (1992). Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism. Proceedings of ACM SIGCOMM. pp. 14-26.

Cruz, R. L. (1995). "Quality of service guarantees in virtual circuit switched networks." IEEE J. Select. Areas Commun. **13**(6): 1048-1056.

Dadam, P., M. Reichert, et al. (2000). Clinical Workflows: the Killer Application for Process Oriented Information Systems. 4th International Conference on Business Information Systems (BIS 2000), Poznan, Poland. pp. 36-59.

Damen, Z., W. Derks, et al. (2000). Business-to-business E-Commerce in a Logistics Domain. The CAiSE*00 Workshop on Infrastructures for Dynamic Business-to-Business Service Outsourcing, Stockholm, Sweden.

DAML-S (2001). Technical Overview - a white paper describing the key elements of DAML-S.

Eder, J. and W. Liebhart (1996). Workflow Recovery. IFCIS Conference on Cooperative Information Systems, Brussels, Belgium. pp. 124-134.

Eder, J., E. Panagos, et al. (1999). Time Management in Workflow Systems. BIS'99 3rd International Conference on Business Information Systems, Poznan, Poland, Springer Verlag. pp. 265-280.

Fabio Casati, Ming-Chien Shan, et al. (2001). "E-Services - Guest editorial." The VLDB Journal **10**(1): 1.

Fensel, D. and C. Bussler (2002). The Web Service Modeling Framework, Vrije Universiteit Amsterdam (VU) and Oracle Corporation. http://www.cs.vu.nl/~dieter/ftp/paper/wsmf.pdf

FGR (2002). Fungal Genome Resource laboratory, http://gene.genetics.uga.edu/

Frolund, S. and J. Koistinen (1998). "Quality-of-Service Specification in Distributed Object Systems." Distributed Systems Engineering Journal **5**(4): 179-202.

Georgiadis, L., R. Guerin, et al. (1996). "Efficient Network QoS Provisioning Based on Per Node Traffic Shaping." IEEE ACM Transactions on Networking **4**(4): 482-501.

Gillmann, M., G. Weikum, et al. (2002). Workflow Management with Service Quality Guarantees. ACM SIGMOD'2002 International Conference on Management of Data, Madison, Wisconsin.

Gillmann, M., J. Weissenfels, et al. (2000). Performance and Availability Assessment for the Configuration of Distributed Workflow Management Systems. International Conference on Extending Database Technology (EDBT), Konstanz, Germany.

Grefen, P., K. Aberer, et al. (2000). "CrossFlow: Cross-Organizational Workflow Management in Dynamic Virtual Enterprises." International Journal of Computer Systems Science & Engineering **15**(5): 227-290.

Hall, R. D., J. A. Miller, et al. (2003). Using Workflow to Build an Information Management System for a Geographically Distributed Genome Sequence Initiative. Genomics of Plants and Fungi. Eds. R. A. Prade and H. J. Bohnert. New York, NY, Marcel Dekker, Inc.**:** 359-371.

Hiltunen, M. A., R. D. Schlichting, et al. (2000). Survivability through Customization and Adaptability: The Cactus Approach. DARPA Information Survivability Conference and Exposition (DISCEX 2000). pp. 294-307.

Hoyland, A. and M. Rausand (1994). System Reliability Theory: Models and Statistical Methods, Wiley, John & Sons, Incorporated.

Ireson, W. G., C. F. C. Jr., et al. (1996). Handbook of reliability engineering and management. New York, McGraw Hill.

Kang, M. H., J. N. Froscher, et al. (1999). A Multilevel Secure Workflow Management System. Proceedings of the 11th Conference on Advanced Information Systems Engineering, Heidelberg, Germany, Springer-Verlag. pp. 271-285.

Kang, M. H., J. S. Park, et al. (2001). Access Control Mechanisms for Inter-organizational Workflows. Proceedings of 6th ACM Symposium on Access Control Models and Technologies, Chantilly, VA.

Klingemann, J., J. Wäsch, et al. (1999). Deriving Service Models in Cross-Organizational Workflows. Proceedings of RIDE - Information Technology for Virtual Enterprises (RIDE-VE '99), Sydney, Australia. pp. 100-107.

Kobielus, J. G. (1997). Workflow Strategies, IDG Books Worldwide.

Kochut, K., A. Sheth, et al. (1999). "Optimizing Workflow." Component Strategies **1**(9): 45-57.

Kochut, K. J. (1999). METEOR Model version 3. Athens, GA, Large Scale Distributed Information Systems Lab, Department of Computer Science, University of Georgia.

Kochut, K. J., A. P. Sheth, et al. (1999). ORBWork: A CORBA-Based Fully Distributed, Scalable and Dynamic Workflow Enactment Service for METEOR. Athens, GA, Large Scale Distributed Information Systems Lab, Department of Computer Science, University of Georgia.

Krishnakumar, N. and A. Sheth (1995). "Managing Heterogeneous Multi-system Tasks to Support Enterprise-wide Operations." Distributed and Parallel Databases Journal **3**(2): 155-186.

Leymann, F. (2001). Web Services Flow Language (WSFL 1.0), IBM Corporation. http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

Luo, Z. (2000). Knowledge Sharing, Coordinated Exception Handling, and Intelligent Problem Solving to Support Cross-Organizational Business Processes. Department of Computer Science. Athens, GA, University of Georgia**:** 171.

Marjanovic, O. and M. Orlowska (1999). "On modeling and verification of temporal constraints in production workflows." Knowledge and Information Systems **1**(2): 157-192.

McCready, S. (1992). There is more than one kind of workflow software. Computerworld. **November 2:** 86-90.

METEOR (2002). METEOR (Managing End-To-End OpeRations) Project Home Page, LSDIS Lab. http://lsdis.cs.uga.edu/proj/meteor/meteor.html

Miller, J. A., J. S. Cardoso, et al. (2002). Using Simulation to Facilitate Effective Workflow Adaptation. Proceedings of the 35th Annual Simulation Symposium (ANSS'02), San Diego, California. pp. 177-181.

Miller, J. A., D. Palaniswami, et al. (1998). "WebWork: METEOR2's Web-based Workflow Management System." Journal of Intelligence Information Management Systems: Integrating Artificial Intelligence and Database Technologies (JIIS) **10**(2): 185-215.

Musa, J. D. (1993). "Operational Profiles in Software-Reliability Engineering." IEEE Software **10**(2): 14-32.

Musa, J. D. (1999). Software reliability engineering: more reliable software, faster development and testing. New York, McGraw-Hill.

Nelson, E. C. (1973). A Statistical Basis for Software Reliability, TRW Software Series.

Pearson, W. R. and D. J. Lipman (1988). Improved tools for biological sequence comparison. Proceedings of the National Academy of Science of the USA. pp. 2444-2448.

Pozewaunig, H., J. Eder, et al. (1997). ePERT: Extending PERT for workflow management systems. First European Symposium in Advances in Databases and Information Systems (ADBIS), St. Petersburg, Russia. pp. 217-224.

Reichert, M. and P. Dadam (1998). "ADEPTflex - Supporting Dynamic Changes of Workflows Without Losing Control." Journal of Intelligent Information Systems - Special Issue on Workflow Managament **10**(2): 93-129.

Rommel, G. (1995). Simplicity wins: how Germany's mid-sized industrial companies succeed. Boston, Mass, Harvard Business School Press.

Sadiq, W. and M. E. Orlowska (1999). Applying Graph Reduction Techniques for Identifying Structural Conflicts in Process Models. Proceedings of the 11th International Conference on Advanced Information Systems Engineering (CAiSE '99), Lecture Notes in Computer Science, Springer-Verlag, Berlin. pp. 195--209.

Shegalov, G., M. Gillmann, et al. (2001). "XML-enabled workflow management for e-services across heterogeneous platforms." The VLDB Journal **10**(1): 91-103.

Stalk, G. and T. M. Hout (1990). Competing against time: how timebased competition is reshaping global markets. New York, Free Press.

Zinky, J., D. Bakken, et al. (1997). "Architectural Support for Quality of Service for CORBA Objects." Theory and Practice of Object Systems **3**(1): 1-20.