Modeling Quality of Service for Workflows and Web Service Processes

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

[1]LSDIS Lab, Department of Computer Science
[2]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 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, the underlying workflow system 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. To this end, we present a model that specifies QoS and describe an algorithm and a simulation system in order to compute, analyze and monitor workflow QoS metrics.

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.

Our experience with real-world applications (CAPA 1997; Kang, Froscher *et al.* 1999; Hall, Miller *et al.* 2000; Anyanwu, Sheth *et al.* 2002) has made us aware that existing workflow systems, both products and research prototypes, provide a set of indispensable functionalities that manage and streamline business processes. Yet, organizations operating in e-commerce and in global economies that include competitive and constantly changing markets have a new set of requirements that have not been answered by current workflow technologies. One important missing requirement is the management of Quality of Service (QoS), or technical aspects of Service Level Agreements (SLAs). Organizations operating in modern markets, such as e-commerce activities and distributed Web service interactions, require QoS management. Products and services with well-defined specifications must be available to customers. 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 workflow Quality of Service management. 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. To enhance QoS management, we present a comprehensive framework based on a QoS model for tasks and a set of functions to compute the overall QoS of workflows.

For organizations, being able to characterize workflows based on QoS has four distinct advantages. First, 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. Second, 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. For e-commerce processes, it is important to know the QoS an application will exhibit before making the service available to customers. Third, 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. Fourth, it allows for the evaluation of alternative strategies when adaptation becomes necessary. The unpredictable nature of the surrounding environment has an important impact on the strategies, methodologies, and structure of business processes. Thus, 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. For example, when a workflow becomes unavailable due to the malfunction of some of its components, it is indispensable to evaluate the adaptive strategies that can be applied to correct the process. It is essential that the services rendered follow customer specifications to meet their expectations and ensure satisfaction. Customer expectations and satisfaction can be translated into the quality of service rendered. Organizations have realized that quality of service management is an important factor in their operations. Quality models, such as ISO9000 (ISO9000 2002), have been created to help organizations and their individual performers meet customer needs.

This paper presents a comprehensive model for the specification of workflow QoS as well as methods to analyze and monitor 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 real and usable workflow QoS model. Once the QoS and associated dimensions are selected, it is necessary to develop algorithms and to select methods to compute QoS. In 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. We present an algorithm and also show how a workflow system can be coupled with a simulation system in order to predict QoS.

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, tasks or web-service only have to adhere to the QoS model.

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. In Section 5, we present an algorithm to compute and estimate workflow QoS, and we also describe how simulation techniques can be used for QoS estimation. Section 6 presents an example of how to compute the QoS for the workflow introduced in our initial scenario. Section 7 discusses the related work in the QoS area; section 8 presents future work on workflow QoS. Finally, section 9 presents our conclusions.

2 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.* 2000). 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.

2.1 Workflow Structure

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

A workflow is composed of tasks 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*.

2.2 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.

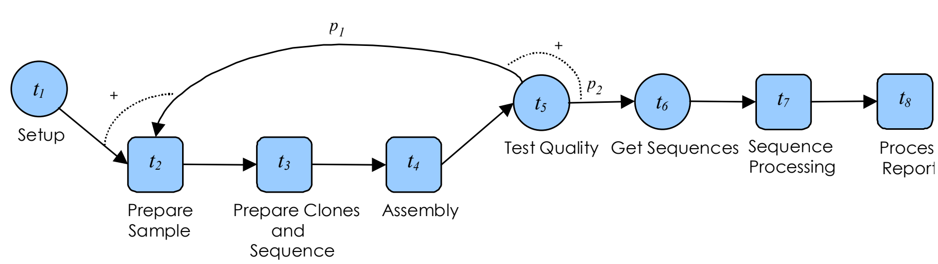


Figure 1– DNA Sequencing workflow

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.

2.3 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, fidelity, quality, 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.
- The error rate of the task *Prepare Clones and Sequence* has to be at most ε, and the data quality of the task *Sequence Processing* has to be at least α.
- 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 four non-functional requirements: time, cost, fidelity, and reliability. While the specification of such quality requirements is important, current WfMSs do not include the functions to delineate their specification or management.

2.4 Current WfMSs Limitations

The lack of a mechanism to specify workflow QoS is a current limitation of WfMSs. However, this is not the only missing element; once a workflow QoS model is defined, three additional components need to be developed: *estimation algorithms and methods*, *monitoring tools*, and *mechanisms to control* the quality of service. Only the development of integrated solutions composed of those four modules (specification, estimation, monitoring, and control) can result in a sophisticated quality management framework. The objectives and functionalities of each module include the following:

- A quality of service model must be developed to allow for the *specification* of workflow Quality of Service (QoS) metrics. This model allows suppliers to specify the duration, quality, cost, fidelity, *etc.*, of the services and products to be delivered. Specifications can be set at design-time, when designers build workflow applications, or they can be adjusted at run-time.
- Algorithms and methods must be developed to *estimate* the quality of service of a workflow both before instances are started and during instance execution. The estimation of QoS before instantiation allows suppliers to ensure that the workflow processes to be executed will indeed exhibit the quality of service requested by

customers. The analysis of workflow QoS during instance execution allows workflow systems to constantly compute QoS metrics and register any deviations from the initial requirements.

- Tools must be available to *monitor* the quality of service of running workflow instances. Workflow users and managers need to receive information about the QoS status and possible deviations from the desired metrics that might occur. In our scenario, let us assume that for some unknown reason the *matching factor* of the DNA Sequencing data drops below a threshold expressed by the customer. The *matching factor* reflects the degree of similarity between the query sequence ("probe") and the compared ("subject") sequence stored in a sequence database. The use of workflow QoS monitoring tools can automatically detect this variation in fidelity and automatically notify interested users.
- Mechanisms must be available which *control* the quality of service of workflow instances. Control is necessary when instances do not behave according to initial requirements. Let us consider the following example: workflow instances are running correctly and the quality of service specifications are being followed when a task fails. The task *Prepare Clone and Sequence* stops its processing because one of the associated machines has a mechanical problem. As a consequence, workflow QoS specifications of time are no longer satisfied, and the WfMS raises a warning, an alert, or an exception. The faulty task needs to be replaced by an equivalent task to restore the soundness of the system. This replacement can be accomplished by applying dynamic changes to the workflow instances, either manually or automatically (Cardoso, Luo *et al.* 2001).

While these four areas of research are important and indispensable for adequate quality of service management, in this paper we focus on the specification, estimation, and monitoring of workflow QoS.

3 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*. Kobielus (1997) suggests that dimensions such as time, cost, and quality should constitute the criteria that workflow systems should include and might benefit from. 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.

3.1 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*. These three dimensions have been a major concern for organizations. Garvin (1988) associates eight dimensions with quality, including performance and reliability. Software systems' quality of service has also been extensively studied. Major contributions can be found 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; Hiltunen, Schlichting *et al.* 2000). For middleware systems, Frolund and Koistinen {Frolund, 1998 #106} present a set of practical dimensions for distributed object systems' reliability and performance, which include TTR (time to repair), TTF (time to failure), availability, failure masking, and server failure. For data networks, the QoS generally focuses on domain-specific dimensions such as bandwidth, latency, jitter, and loss (Nahrstedt and Smith 1996).

Our past work on deploying workflow applications has made us aware of the need for workflow process QoS management. Additionally, we have realized that workflow processes have a particular set of requirements which are domain dependent and that need to be accounted for when creating a QoS model. Based on previous studies and our experience in the workflow domain, we have constructed a QoS model composed of the following dimensions: *time*, *cost*, *reliability,* and *fidelity*. According to Weikum (1999), information services QoS can be divided into three categories: system centric, process centric, and information centric. Our model specifies quality dimensions that include the system and process categories. QoS specifications are set for task definitions. Based on this information, QoS metrics are computed for workflows (see section 5).

3.2 Task Time

Time is a common and universal measure of performance. For workflow systems, it can be defined as the total time needed by an instance to transform a set of inputs into outputs. 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, since the products are more rapidly introduced into the market. Additionally, reducing the time taken to execute a set of tasks in a workflow process makes it possible for an organization to be more responsive to customers' needs. Therefore, it is important to enhance WfMS to include time-based process execution.

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 mates 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.

3.3 Task Cost

Task cost represents the cost associated with the execution of workflow tasks. Cost is an important factor, since organizations need to operate according to their financial plan. It is fundamental for organizations that wish to reduce their expenditures on internal processes and wish to control product and service cost. 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 workflow instances monitoring. 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.

3.4 Task Reliability

In an early work on workflow modeling, Krishnakumar and Sheth (1995) represented the execution behavior of each task, using task structures. Each workflow task structure has an initial state, an execution state, and two distinct terminating states. One of the states indicates that a task has failed (for non-transactional tasks) or was aborted (for transactional and open 2PC tasks), while the other state indicates that a task is done or committed (Figure 2). 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. Based on this task model structure, we introduce the *reliability* dimension. This QoS dimension 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.

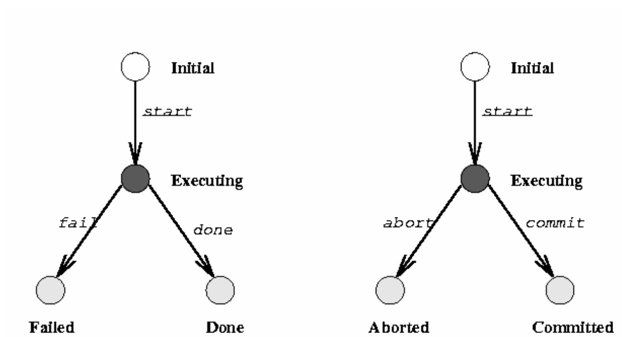


Figure 2 - Two task structures (Krishnakumar and Sheth 1995)

Task Reliability (R) corresponds to the likelihood that the components will perform for its users on demand; it is a function of the failure rate. 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), where failure rate is given as the ratio of *successful executions/scheduled executions*.

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

Table 1 – Task reliability

For each task, the WfMS keeps track of the number of times the task has been scheduled for execution and how many times the task has been successfully executed. R(t) is a stable model, since when software failure occurs no fault removal is performed.

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. Goel (1985) classified reliability models into four kinds: input domain-based models, times-between-failures models, failure-count models, and fault seeding models. 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.

3.5 Task Fidelity

We view fidelity as a function of effective design; it refers to an intrinsic property(ies) or characteristic(s) of a good produced or service rendered. Fidelity reflects how well a product is being produced and how well a service is being rendered. Fidelity is often difficult to define and measure because it is subject to judgments and perceptions. Nevertheless, the fidelity of workflows should be predicted whenever feasible and carefully controlled when needed (Kolarik 1995; Franceschini 2002).

3.5.1 Fidelity and Fidelity Attributes

Workflow tasks have a fidelity (F) vector dimension composed of a set of fidelity attributes ($F(t).a_r$), that reflect and quantify task operations. Each fidelity attribute refers to a property or characteristic of the product being created, transformed, or analyzed. During the development of a workflow, the domain expert and the business analyst are responsible for the identification of the various fidelity attributes to be associate with tasks. After the execution of a task, its fidelity attributes can be used by the workflow system to compute how well workflows, instances, and tasks are meeting user specifications.

The fidelity attributes are set during the execution of tasks. Depending on the task type, a task uses different strategies to set fidelity attributes. Three scenarios can be drawn: automatic tasks controlling hardware, automatic tasks controlling software, and human tasks. For an automated task controlling a hardware device, the fidelity attribute can be set after reading the output status line of the device. For example, the task *Sequencing* controls DNA sequencing, which is carried out automatically by a sequencer. When the sequencing finishes, the machine generates several output files to describe how the process was executed. These values can be passed on to the task, which automatically updates its fidelity attributes. For automated tasks controlling a software application, the same procedure can be applied. For example, the task *Sequence Processing* executes various algorithms on the sequences received. One of the algorithms used is BLAST (Altschul, Gish *et al.* 1990). This algorithm searches DNA sequences in a database to identify macromolecules with related structures and functions. Once the search is concluded, the algorithm returns a value indicating the confidence of the matching. For this task, the returned value from the execution of the algorithm will be used to describe the fidelity of the task's execution. For human tasks, the procedure has to be manual. Therefore, it is the responsibility of the user to manually input information relative to the fidelity of the task executed. In the case of the task *Prepare Sample,* the lab technician sets the fidelity attribute quality of clones manually, after a visual identification. For quality assurance reasons the attributes should be set or checked by a person other than the one who carried out the task execution. If evaluating the fidelity of a task cannot be accurately done by a human, an option is to place – when possible – an automatic task after the human task to automatically check the fidelity.

3.5.2 Local Normalized Fidelity

The fidelity attributes of a task give a vector of information characterizing how well a task has carried out its execution. But sometimes it is useful to characterize the task execution using a unique value, and not a vector of values. To achieve this, we introduce the concept of local normalized fidelity.

Each task is associated with a fidelity function F(t), which represents the local normalized fidelity:

$$F(t) = \sum_{i=1}^{n} z_i (F(t).a_k) * w_i$$

The formula weights the fidelity attributes, which can be transformed to more appropriate values using the normalization functions z_i. The sum of the weights w_i is equal to 1.

For example, in our experiments (section 6) the *Test Quality* task checks the probability that the sample being sequenced is contaminated with *E. coli* using the BLAST algorithm. Let us assume that the *Test Quality* task is modified to also use the FASTA algorithm to check the contamination of a sample. In this case, the *Test Quality* task has two fidelity attributes: $F(t).a_{E.coli\ matching\ BLAST}$ and $F(t).a_{E.coli\ matching\ FASTA}$.

Let us set function z_1 to normalize the results of the BLAST algorithm,

$$z_1(x) = \frac{1}{\mathrm{e}^{-15}} * \min(\mathrm{e}^{-15}, x)$$

where e^{-15} is the upper bound for which there is some indication that the sample contains *E. coli*. Let us define a similar function to normalize the results of the FASTA algorithm,

$$z_2(x) = \frac{1}{0.01} * \min(0.01, x)$$

We obtain the following local normalized fidelity, where $w_1 = w_2 = 0.5$,

$$F(t) = z_1(F(t).a_{E.coli\ matching\ BLAST})*0.5 + z_2(F(t).a_{E.coli\ matching\ FASTA})*0.5$$

At runtime, the function F(t) will give information about the quality of the sample being analyzed. For example, if both algorithms return values very close to zero after the analysis of a sample, which means that there is a high probability of *E. coli*, function F(t) will yield a value also close to zero. Indicating that the sample under analysis has a poor quality. On the other hand, if the values returned from the two algorithms are very high

(>10 for example) then function F(t) will return a value very close to 1, or even 1. Thus, indicating that the sample does not have any *E. coli*.

In view of the fact humans often feel awkward in handling and interpreting such quantitative values (Tversky and Kahneman 1974), we provide an option to the designer (who would usually take the help of a domain expert), to map the value resulting from applying the fidelity function to a qualitative scale (Miles and Huberman 1994). This qualitative indicator is used to detect areas of a workflow with anomalies and undesired behavior. An example of a mapping scale for quantitative and qualitative values is shown in Table 2. The workflow designer is responsible for the creation of the mapping table. The table is created by first selecting a set of qualitative terms that characterize the fidelity. The use of qualitative terms may facilitate the human understanding of the fidelity concept exhibited by workflows in some cases.

Qualitative Fidelity	Quantitative Fidelity
Unacceptable	[0.00.. 0.20]
Poor	[0.21.. 0.40]
Satisfactory	[0.41.. 0.60]
Good	[0.61.. 0.80]
Excellent	[0.81.. 1.00]

Table 2 – Example of a fidelity-mapping table

The fidelity information can be used to effectively monitor workflow executions. Typically, during the lifetime of an instance, qualitative information describing task fidelity is displayed on graphical monitors as the tasks are executed. Managers can easily identify tasks which exhibit unsatisfactory fidelity metrics.

3.6 QoS Model Discussion

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.

We recognize that 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 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.

4.1 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 3): 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 3 and Table 4 show the QoS dimensions for an automatic task (the *SP FASTA* task) and for a manual task (the *Prepare Sample* task; see section 2.2 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
Fidelity.a_i	0.63	0.81	0.92	Trapezoidal(0.7,1,1,4)

Table 3 – 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
Fidelity.a_k	-	-	-	-

Table 4 – 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 (see section 5.2). 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, fidelity, 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. 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 $\text{Average}_{\text{Dim}}(t)$); b) the average of a task *t* dimension is calculated based on all its executions independently of the workflow that executed it (Multi-Workflow $\text{Average}_{\text{Dim}}$ (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 $\text{Average}_{\text{Dim}}(t, w)$); and d) the average of the dimension of all the times task t was executed in instance i of workflow w (Instance $\text{Average}_{\text{Dim}}(t, w, \text{i})$). Scenario d) can only occur when loops exist in a workflow.

While the formulae presented only show only 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 5.

a)	$\text{QoS}_{\text{Dim}}(t)$	Designer $\text{Average}_{\text{Dim}}(t)$
b)	$\text{QoS}_{\text{Dim}}(t)$	wi_1* Designer $\text{Average}_{\text{Dim}}(t)$ + wi_2* Multi-Workflow $\text{Average}_{\text{Dim}}(t)$
c)	$\text{QoS}_{\text{Dim}}(t, w)$	wi_1* Designer $\text{Average}_{\text{Dim}}(t)$ + wi_2* Multi-Workflow $\text{Average}_{\text{Dim}}(t)$ + wi_3*Workflow $\text{Average}_{\text{Dim}}(t, w)$
d)	$\text{QoS}_{\text{Dim}}(t, w, i)$	wi_1* Designer $\text{Average}_{\text{Dim}}(t)$ + wi_2* Multi-Workflow $\text{Average}_{\text{Dim}}(t)$ + wi_3* Workflow $\text{Average}_{\text{Dim}}(t, w)$ + wi_4* Instance Workflow $\text{Average}_{\text{Dim}}(t,w, i)$

Table 5 – QoS dimensions computed at runtime

The workflow system uses the formulae from Table 5 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. 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.

4.2 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 5).

5 Workflow QoS Computation

Once QoS estimates for tasks and for transitions are determined, we can compute overall workflow QoS. We describe two modeling techniques that can be used to compute QoS metrics for a given workflow process: mathematical modeling and simulation modeling. The selection of the method is based on a tradeoff between time and the accuracy of results. The mathematical method is computationally faster, but it yields results which may not be as accurate as the results obtained by simulation. (Note that our mathematical models could be extended to queuing network models (Lazowska, Zhorjan *et al.* 1984), but this requires making some simplifying assumptions).

5.1 Mathematical Modeling

Our Stochastic Workflow Reduction (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.

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).

To compute QoS metrics, we have developed the *SWR*(*w*) algorithm (Cardoso 2002), which uses a set of six distinct reduction rules: (1) sequential, (2) parallel, (3) conditional, (4) fault-tolerant, (5) loop, and (6) network.

Additional reduction rules can be developed. We have decided to present the reduction concept with only six reduction rules, for two 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.* 2002) 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.

5.1.1 Reduction Systems

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}.

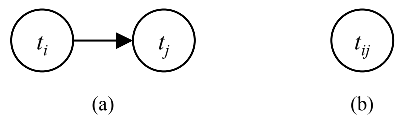


Figure 3 - Sequential system reduction

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:

$$T(t_{ij}) = T(t_i) + T(t_j)$$
$$C(t_{ij}) = C(t_i) + C(t_j)$$
$$R(t_{ij}) = R(t_i) * R(t_j)$$
$$F(t_{ij}).a_r = f(F(t_i).a_r, F(t_j).a_r)$$

The fidelity attributes of task t_{ij} are derived from the fidelity of task t_i and t_j. Since the fidelity of a task is domain dependent, domain experts and business analysts are responsible for the definition of function *f*. A detailed discussion on fidelity can be found in section 5.1.3.

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.

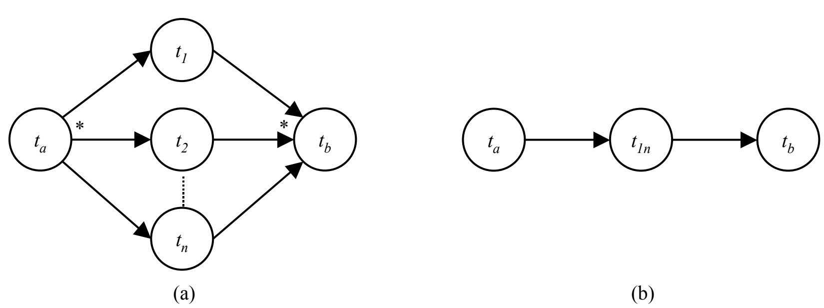


Figure 4 - Parallel system reduction

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

$$T(t_{1n}) = \mathrm{Max}_{I\in\{1..n\}} \{T(t_i)\}$$

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

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

$$F(t_{1n}).a_r = f(F(t_1).a_r, F(t_2).a_r, \ldots, F(t_n).a_r)$$

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.

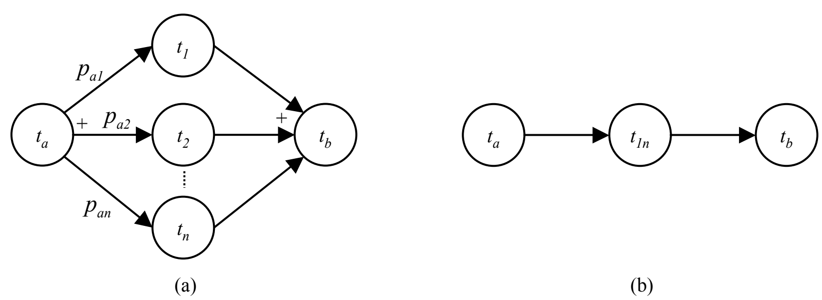


Figure 5 - Conditional system reduction

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

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

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

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

$$F(t_{1n}).a_r = f(p_{a1}, F(t_1).a_r, p_{a2}, F(t_2).a_r, \ldots, p_{an}, F(t_n).a_r)$$

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 $p_{lk} = \frac{\mathrm{p_{ok}}}{1 - \mathrm{p_i}}$, and $\sum_{k=1}^{n} p_{lk} = 1$.

Figure 6 – Simple loop system reduction

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

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

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

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

$$\mathrm{F}(t_{li}).\mathrm{a_r} = f(p_i, \mathrm{F}(t_i).\mathrm{a_r})$$

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 $p_{lk} = \frac{\mathrm{p_{ok}}}{1 - \mathrm{p_i}}$ and $\sum_{k=1}^{n} p_{lk} = 1$.

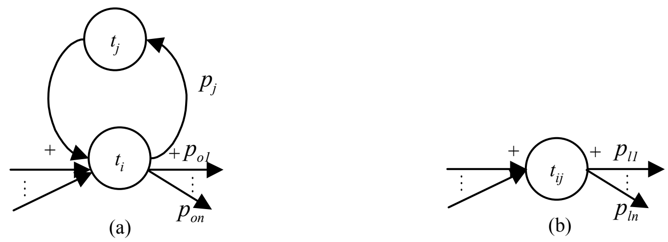


Figure 7 – Dual loop system reduction

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

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

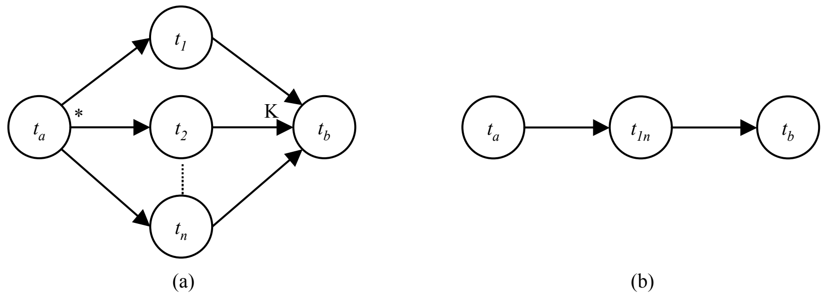
$$\mathrm{C}(t_{ij}) = \frac{\mathrm{C}(t_i) + \mathrm{C}(t_j) - (1-\mathrm{p_j})\mathrm{C}(t_j)}{(1-\mathrm{p_j})}$$

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

$$\mathrm{F}(t_{ij}).\mathrm{a_r} = f(\mathrm{F}(t_i).\mathrm{a_r}, p_j, \mathrm{F}(t_j).\mathrm{a_r})$$

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 $\{t_1, t_2, …, t_n\}$ 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 $\{t_1, t_2, …, t_n\}$ 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:

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

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

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

$$\mathrm{F}(t_{1n}).a_r = f(p_{a1}, \mathrm{F}(t_1).a_r, p_{a2}, \mathrm{F}(t_2).a_r, \ldots, p_{an}, \mathrm{F}(t_n).a_r, k)$$

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

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

The formula $\mathrm{R}(t_{1n})$ 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 $\mathrm{R}(t_{1n})$ 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 $\mathrm{R}(t_{1n})$, the summation over $i_1, \ldots, i_n$ generates all the possible probabilistic states. Each probabilistic state is represented with a binary sequence ($i_1 \ldots i_n$) 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 $g(2\text{-}2)*((1- i_1)+(2i_1\text{-}1)R(t_1))*((1- i_2)+(2i_2\text{-}1)R(t_2))*((1- i_3)+(2i_3\text{-}1)R(t_3))$ 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,* $X(t_j) = X(t_i)$, $X \in \{T, C, R, F\}$.

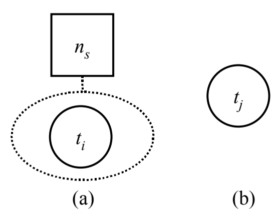


Figure 9 - Network system reduction

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

5.1.2 SWR Complexity

The running time of the SWR algorithm is $O(n^3+nl)$, where n is the number of nodes in a workflow, and l the number of loops. Having an initial workflow with n nodes, we need to make at most n tests to find a node n_t to determine if it belongs to a reduction system with nodes $\{n_i, \ldots, n_j\}$ so that a reduction can occur. Each test takes at most $O(k)$ time, where k is the number of nodes of the system being tested and it is at most equal to n.

When n tests have been completed, at least one reduction should have been applied. If it is not the case, then the workflow being reduced has an incorrect design. After applying a reduction, the number of nodes of the transformed workflow is at least n-1, except if the reduction has been applied to a basic loop system. Therefore, *n*-1+*l* tests still need to be carried out, where *l* is the number of loops in a workflow. The number of tests to carry out is at most:

$$\sum_{0<i<n}(n-i)+l=\frac{n(n+1)}{2}+l$$

Each test takes O(*k*) time to carry out. Once a test is positive, meaning that a reduction system can be applied, it takes O(*k*) to make the reduction. The overall runtime of the algorithm is:

$$\mathrm{O}\left(\frac{n(n+1)}{2}+l\right)*\mathrm{O}(k)+\mathrm{O}(k)=\mathrm{O}(n^3+nl)$$

Typically a workflow has a number of tasks which is much greater than the degree of parallel, conditional, and fault-tolerant systems, and the number of loops. Therefore, when $n >> k$ and $n >> l$ we have,

$$\mathrm{O}\left(\frac{n(n+1)}{2}+l\right)*\mathrm{O}(k)+\mathrm{O}(k)\approx\mathrm{O}(n^2)$$

5.1.3 Time, Cost, Reliability, and Fidelity Computations

Time and Cost. The operations used to compute the time and cost dimensions are fairly intuitive. The results from the computation of the time and cost dimension are valid when a workflow succeeds. The inclusion of repair actions (maintenance) in the QoS computation, when a workflow fails, are discussed in the future work section.

Reliability. For the reliability dimension we have used concepts from system and software reliability theory (Hoyland and Rausand 1994; Ireson, Jr. *et al.* 1996; Musa 1999). The reliability functions used when applying workflow reduction systems assume that tasks behave independently. While this assumption is widely employed when modeling hardware systems, it is considered by some to be inappropriate for software systems since they tend to violate the independence supposition of the individual software systems.

Mason and Woit (1998) show that an application's structure has an influence on the dependability derived from the reliability of its components. Their work presents a theory based on a set of rules which when applied to the construction of an application can result in systems which do not violate the underlying assumptions of the typical reliability models, *i.e,* system independence. In order to understand the dependence of software components it is necessary to understand the difference between the terms "uses" and "invokes" (Parnas 1974; Parnas 2001). The utilization of "use" methodology creates a dependency between modules or procedures. This is because if a module A calls a module B, then the state of A depends on the results of B. Using the "invokes" methodology this problem does not arise, since when module A calls module B, module A does not wait or depend on B's execution results. Based on this observation, Mason and Woit (1998) state that to reduce the dependence of modules in a system or application a, “uses” methodology should not be present to interconnect the components; instead, a “invokes” methodology should be present. Additionally, the module’s implementation details cannot affect the correctness of other modules in the system (state independence).

The architecture of workflow systems directly follows the two points that allow for a reduction of task dependencies. Workflow systems such as OrbWork (Kochut, Sheth *et al.* 1999) use a message-passing architecture and thus exhibit “invokes” characteristics. Additionally, tasks are independent from the implementation point of view, and therefore they are state independent. Due to the architecture of typical WfMSs, workflow applications have a reduced dependency factor among tasks; we make the assumption that the dependencies can be ignored in many cases.

Fidelity. While time, cost, and reliability are common measurements, the fidelity fully depends on the intrinsic properties and characteristics of the goods produced– it is not a universal measurement. This means that for each reduction rule presented previously, it is not possible to specify a general and universal formula to compute fidelity. Thus, for each reduction system (except for network systems) and for each fidelity attribute, a specific formula needs to be specified. For example, the Swiss watchmaker TAG Heuer conducts a series of sixty tests of their watches during the manufacturing process. Specific tasks carry out the tests, which are placed at strategic locations in the process. Each testing task can have a fidelity attribute associated with it that represents the number of tests that have been passed when the task was executed. In this case, the following fidelity function can be specified for the sequential reduction rule:

$$F(t_{ij}).a_{\text{number of tests passed}} = f(F(t_i).a_{\text{number of tests passed}}, F(t_j).a_{\text{number of tests passed}}) \text{ where}$$

$$f(x, y) = x + y$$

In this example, the function *f* is additive and simply adds the number of tests passed by each task. In other cases, the function *f* can be multiplicative, and therefore can be similar to the functions employed to compute metrics for the reliability dimension.

It is the responsibility of the business analyst and domain expert to set for each fidelity attribute involved in a workflow the fidelity functions (*f*) to be used when computing workflow QoS.

5.2 Simulation Modeling

While the values specified in the basic class are employed by mathematical methods (see previous section) to effectively compute workflow QoS, another alternative is to utilize simulation systems calibrated with the information specified in the distributional class (see section 4.1). Our analytic model computes average results/predictions, simulation can give extreme cases, averages, quartiles, standard deviation and confidence intervals.

Simulation can play an important role in tuning the quality of service metrics of workflows by exploring ”what-if“ questions (Miller, Cardoso *et al.* 2002). When the need to adapt or to change a workflow is detected, deciding what changes to carry out can be very difficult. Before a change is actually made, its possible effects can be explored with simulation. To facilitate rapid feedback, the workflow system and the simulation system need to interoperate. In particular, workflow specification documents need to be translated into simulation model specification documents so that the new model can be executed/animated on-the-fly.

In our project, these capabilities involve a loosely-coupled integration between the METEOR WfMS and the JSIM simulation system (Nair, Miller *et al.* 1996; Miller, Nair *et al.* 1997; Miller, Seila *et al.* 2000). Workflow is concerned with scheduling and transformations that take place in tasks, while simulation is mainly concerned with system performance. The JSIM simulator supports the constructs appearing in the METEOR WfMS, facilitating interoperation of the two systems. Entities flow along directed graphs where delays at nodes will simulate task execution time. Routing along edges may be either conditional or probabilistic. In addition, XOR as well as AND splits/joins are supported. interoperation is facilitated. In order to carry out a simulation, the appropriate workflow model is retrieved from the repository and translated into a JSIM simulation model specification. The distribution functions specified in the distributional class (section 4.1) of workflow tasks are retrieved and associated with the nodes of the simulation model. The simulation model is displayed graphically and then executed/animated. Statistical estimates for workflow QoS metrices are collected and displayed.

In order to simulate METEOR workflows, we have enhanced the JSIM Web-Based Simulation System. In JSIM, simulation entities flow through a digraph consisting of the following types of nodes.

Source	Produces entities with random times
Server	Provides service to entities
Facility	Inherits from server, adds a waiting queue
Signal	Alters number of service units in a server(s)
Sink	Sink consumes entities and records statistics

Table 6 – Nodes in JSIM

These nodes are connected together with transports, which move entities from one node to the next. These edges provide a smooth motion of entities when a simulation model is animated. These edges are labeled with branching probabilities which are retrieved from the stochastic workflow model (see section 4.2).

The mapping of a workflow digraph to a simulation digraph is straightforward. A METEOR *start*, *stop* task will be mapped to a JSIM Source and Sink node, respectively. A METEOR human task will be mapped to a JSIM Facility, with the number of service units equal to the number of human participants carrying out the task and feeding of the same worklist. A METEOR transactional/non-transactional task will be mapped to a JSIM Facility, with the number of service units equal to the number of processors available to execute the task. These default mappings can be customized (*e.g.*, a non-transactional task that does not allow requests to be queued should be mapped to a JSIM Server). Each edge in the METEOR digraph will be mapped to a corresponding edge in the JSIM digraph. In METEOR, edges are labeled with the data type of objects flowing along the edge. In the case of *xor* nodes, they are also labeled with Boolean expressions. (The first one that evaluates to true will be the edge selected.) A Boolean expression can be mapped to the probability that the condition will evaluate to true and that none of the preceding conditions will evaluate to true. For more details on mapping workflow specifications into simulation models specifications, see Chandrasekaran *et al.* (2002).

5.3 Workflow QoS Metrics of Interest

In this section, we list the workflow QoS metrics which are of interest to compute. The computation can be done at either design time or runtime. At design time, QoS computations help the designer to compose workflows that will exhibit QoS metrics that satisfy initial requirements. At runtime, the computation of QoS allows the manager and administrator to identify workflow instances that have ceased to meet initial QoS requirements. This situation may occur when tasks fail, break down, or when necessary services are unavailable. The metrics presented can be automatically computed using the SWR algorithm.

Important measurements for workflow execution include *workflow response time*, *workflow delay time*, *minimum workflow response time*, *workflow response time efficiency, workflow cost, workflow fidelity, and workflow reliability*. The following three workflow metrics are shown as an example.

The **workflow response time** (T) is the total amount of time that a workflow instance spends within a workflow process before it finishes. The response time in a workflow is equal to the sum of the response times at the individual tasks, less any time that two or more tasks are superimposed on one another. Two or more tasks superimpose their response time when they are executed in parallel.

$$\mathrm{T}(w) = \mathrm{T}(SWR(w))$$

The **minimum workflow response time** (min T), sometimes called the “service time” of a workflow, is the time required for a workflow instance to be processed, not accounting for any task delay time. Thus, it includes only the task response time, ignoring completely the impact of the task delay time. The minimum workflow response time is equal to the sum of the process time at the individual tasks, less any time that two or more tasks superimpose.

$$\min \mathrm{T}(w) = \min \mathrm{T}(SWR(w))$$

The **workflow response time efficiency** (E) is the ratio of the minimum workflow response time and the workflow response time. It is instructive to compare these two measures, since instance efficiency measurement provides an indication of the time an instance is delayed during its execution and also indicates the degree a workflow process can be improved by reducing its response time.

$$\mathrm{E}(w) = \frac{\min \mathrm{T}(w)}{\mathrm{T}(w)}$$

6 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.

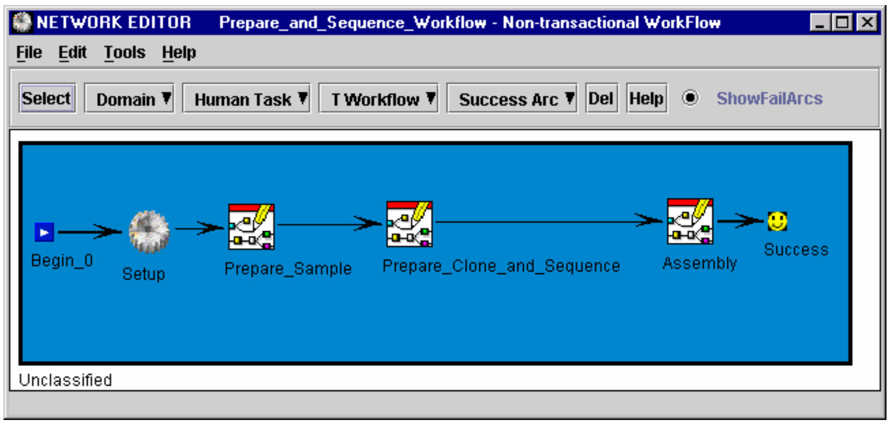


Figure 10 – Prepare and Sequence Workflow

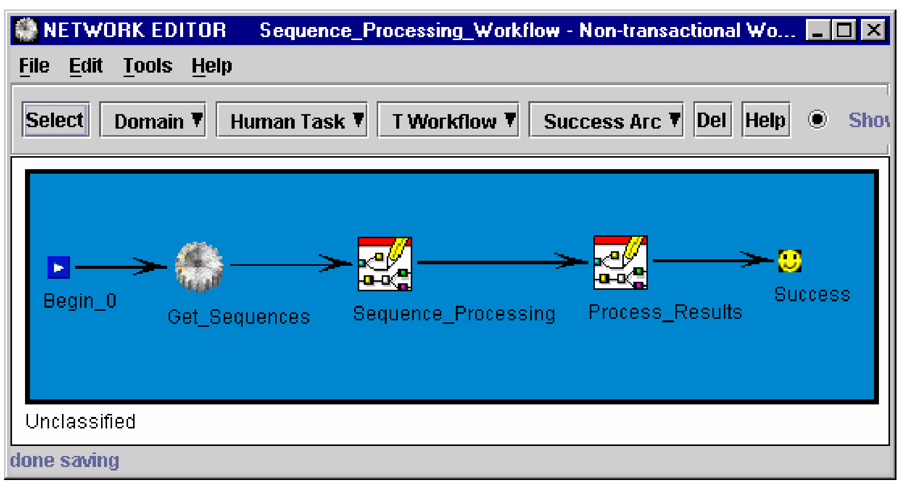


Figure 11 – Sequence Processing Workflow

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 10 and the *Sequence Processing* workflow from Figure 11,

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 12.

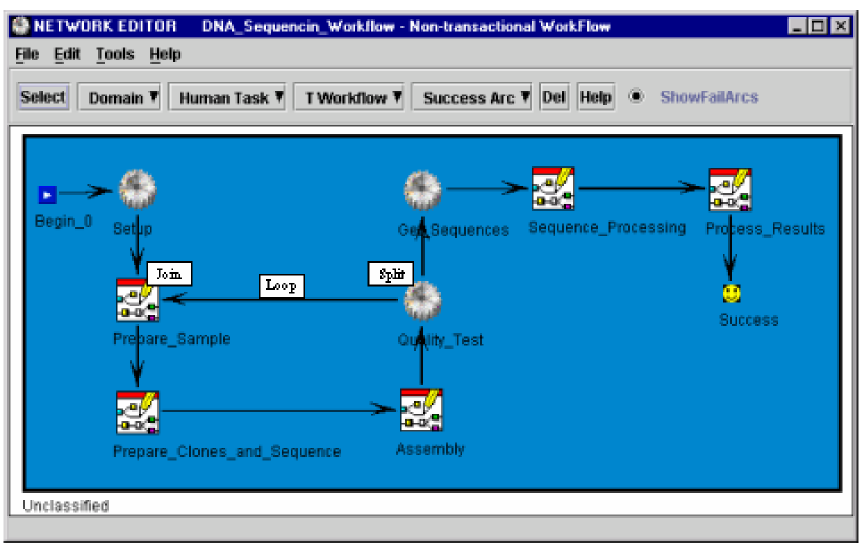


Figure 12 – DNA Sequencing Workflow

6.1 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, reliability, and fidelity 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:

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

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

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

The only task with a fidelity function is the *SP BLAST* task. The fidelity attribute *HITS* indicates the percentage of sequences processed with an E value lower than e^{-15}. The E value is an indication of the probability that the match between a query sequence and a sequence stored in a database occurred by chance. For close matches, this number is typically very small.

$$F(t_{BSP\ BLAST}).{}_{HITS} = \text{percentage of sequences with } E < e^{-15}$$

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 7.

Tasks	Designer Specifications T(t)	C(t)	R(t)	F(t)
Quality Test	0.01	$0.0	100%	n/a
SP FASTA	9.59	$0.0	100%	0.65

Table 7 – 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).)

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

As for the *SP BLAST* task, the following fidelity function has been utilized to characterize the quality of the results obtained by the task *SP FASTA*:

$$\mathrm{F}(t_{\mathrm{SP\ FASTA}}).{}_{\mathrm{HITS}} = \text{percentage of sequences with } E < 0.01$$

Generally, a value of 0.01 or below is statistically very significant, and a value between 0.01 and 0.05 is the borderline. The E-values reported from the BLAST and FASTA are significantly dissimilar since the two applications implement different algorithms.

To make the workflow QoS computation possible for the fidelity dimension, formulae have been defined for the reduction systems. As an example, for parallel systems and for the *HITS* fidelity attribute, the following function has been defined:

$$\mathrm{F}(t_{1n}).{}_{\mathrm{HITS}} = f(\mathrm{F}(t_1), \mathrm{F}(t_2), \ldots, \mathrm{F}(t_n)) = \frac{w_i \sum_{1 \le i \le n} \mathrm{F}(t_i).{}_{\mathrm{HITS}}}{\#\text{ of tasks with the fidelity attribute HITS}}$$

Using the above formula in the *DNA Sequencing* workflow will result in the application of the following function:

$$\mathrm{F}(t_{\mathrm{SP\ BLAST\ FASTA}}).{}_{\mathrm{HITS}} = (w_1 * \mathrm{F}(t_{\mathrm{SP\ BLAST}}).{}_{\mathrm{HITS}} + w_2 * \mathrm{F}(t_{\mathrm{SP\ FASTA}}).{}_{\mathrm{HITS}})/2$$

This function represents only a possible computation for the *HITS* fidelity attribute. It is shown here with the objective of illustrating how fidelity attributes are computed. Additional studies of the FASTA and BLAST applications would give more information on the processing of sequences that could be used to give a more precise definition of this function.

6.2 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)	$QoS_{Dim}(t)$	$0.2 * \text{Designer Average}_{Dim}(t) + 0.8 * \text{Multi-Workflow Average}_{Dim}(t)$
c)	$QoS_{Dim}(t, w)$	$0.2 * \text{Designer Average}_{Dim}(t) + 0.2 * \text{Multi-Workflow Average}_{Dim}(t) + 0.6 * \text{Workflow Average}_{Dim}(t, w)$

Table 8 – 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 5.

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.

6.3 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 13. The diamonds indicate the QoS estimates (moving averages) given by the SWR algorithm and the squares indicate empirical runtime metrics. The four dimensions analyzed and presented in Figure 13 show that the predicted results are a good estimation for the measured ones.

Figure 13 – 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 numbered 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 results observed are strongly linked to the results obtained from the time analysis. Again, instances 3 and 4 have recorded actual costs that are considerably different from the values estimated. This is due to the existence of *E. coli* contamination in the sequences processed. When contamination is detected, the re-work necessary to carry out the sequencing double the cost of the instance. This is because the cost of an instance is totally determined by the tasks *Prepare Sample*, *Prepare Clone and Sequence*, and *Assembly*, which are involved in any necessary re-work. All the other tasks, which are mainly automated software tasks, are considered to have a zero cost. One characteristic of the DNA Sequencing workflow is the discrete linearity of its cost. When no re-work is necessary because no contamination is detected, the cost of the instance is c. If contamination is found, then re-work is needed, and the cost of the instance is $2c$. If contamination is found n times during the sequencing process, the cost of the instance will be nc. This property for the cost dimension can be observed from the chart, where instances with no re-work always have the same cost ($1,152), and instances that need re-working one time have a cost of $2,304.

The fidelity analysis shows the creation of very good estimates. It can be seen that the SWR algorithm constantly changes its estimation as a response to recently recorded QoS metrics. The runtime fidelity metrics are within a small range, as predicted from the estimates.

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 tend towards 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 the 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 8 for the weights used in the DNA Sequencing workflow). A higher weight associated with the multi-workflow function implies a faster adjustment when the SWR algorithm is applied. The same principal applies to the instance workflow function.

7 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 and fidelity as part of the QoS model. We provide two distinct solutions to compute the QoS of workflows: 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 allow the association of distributions with workflow activities. 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 the configuration tool 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.

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.

8 Future Work

The workflow QoS model presented in this paper can be extended in two additional dimensions which are useful for workflow systems with stronger requirements. The first dimension is *maintainability*. Maintainability corresponds to the mean time necessary to repair workflow failures; it is the average time spent to maintain workflows in a condition where they can perform their intended function. Maintenance actions mainly involve the correction of failures during workflow execution. Workflow systems record the period of time necessary for a faulty task to be repaired. The time spent to repair a workflow component depends on the type of error that has occurred. Reparative actions can be as simple as restarting a workflow scheduler that has crashed (Kochut, Sheth *et al.* 1999), or they can be more complex, involving the installation of an ORB infrastructure in a new machine to transfer workflow schedulers, for example. To increase maintainability, advanced mechanisms have been developed to allow workflow systems to automatically recover from errors. Luo *et al*. (2000) describe the architecture and implementation of an exception-handling mechanism. The system detects and propagates exceptions which occur during instances execution to an exception-handling module. The system, based on case-based reasoning theory, derives exception handlers to repair damaged workflows

(Luo, Sheth *et al.* 1998). The system has the ability to adapt itself over time. The knowledge acquired in past experiences is used in the resolution of new problems.

The second dimension that can be included is the *trust* dimension. The use of workflow systems to coordinate and manage Web services compels the development of techniques to appraise the global security level of workflows specifications. Workflow systems and applications face several security problems, and dedicated mechanisms are needed to increase the level of security. Major problems include the distributed nature of WfMSs, the use of non-secure networks (*i.e.,* the Internet), the use of Web servers to access workflow systems data, and the potential multi-organizational span of workflows. Systems security level is assessed through the existence of security mechanisms (such as authentication, access control, labels, audits, system integrity, security policy, *etc*.) and through the use of development techniques (such as formal specifications, formal proofs, tests, *etc*.). The importance of developing secure workflow systems has been recognized, and prototypes combining workflow and security technology have already been developed. We have extended workflow technology with the implementation of two security modules. The first one (Miller, Fan *et al.* 1999) and (Fan 1999) describes a workflow security architecture which targets the five security services (authentication, access control, data confidentiality, data integrity, and non-repudiation) recommended by the International Standards Organization for network-based information systems. The second one (Kang, Froscher *et al.* 1999) describes a multilevel secure (MLS) workflow system to enable distributed users and workflow applications to cooperate across classification levels. MLS workflow systems allow users to program multilevel mission logic, to securely coordinate distributed tasks, and to monitor the progress of the workflow across classification levels.

The functions used to compute the QoS dimensions at runtime (Table 5) have their terms weighted. The user is responsible for setting the weights (wi_1, wi_2, wi_3, and wi_4). These weights remain constant as the workflow system registers new workflow executions. Additional research would be useful to analyze the effect of replacing the constant weights with variable weights. The idea would be to allow the workflow system to automatically change the weights based on the number of workflow executions. As more instances are registered for a workflow *w*, the weights specified for the Designer and Multi-Workflow functions can be decreased and the weight associated with the Workflow function increased. This corresponds with the belief that over time the QoS metrics of the instances of the workflow *w* will give more accurate and fresh data to be used with the SWR algorithm. The use of Bayesian estimates (Bernardo and Smith 1994) are one of the solutions that can be investigated to enable the automatic adjustments of the weights.

9 Conclusions

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, which in turns represent an abstraction of cross-organizational business processes. The use of workflows and workflow systems to conduct and coordinate businesses in a heterogeneous and distributed environment has an immediate operational requirement: the management of workflow QoS. The composition of Web services, and therefore workflows, cannot be undertaken while ignoring the importance of QoS measurements. Trading agreements between suppliers and customers include the specification of QoS items such as products or services to be delivered, deadlines, quality of products, and cost of service. 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 presented a comprehensive QoS model. This model allows for the description of workflow components from a QoS perspective; it includes four dimensions: time, cost, reliability, and fidelity. 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 have developed 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. We also describe how a simulation system can be used with a workflow system to carry out efficient workflow QoS simulations.

To test the validity of our QoS model and of our mathematical model 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. The analysis of the data corroborates our initial hypothesis that our QoS model and mathematical model give a suitable framework to predict and analyze the QoS of production workflows.

Acknowledgements: WebSvcProc-Qos-vldb-rev2

10 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, A. H. M. t. Hofstede and B. Kiepuszeski (2002). Workflow patterns homepage. http://tmitwww.tm.tue.nl/research/patterns.

Allen, F. E. (1970). "Control Flow Analysis." SIGPAN Notices **5**(7): 1-19.

Altschul, S. F., W. Gish, W. Miller, E. W. Myers and D. J. Lipman (1990). "Basic local alignment search tool." Journal of Molecular Biology **215**: 403-410.

Ankolekar, A., M. Burstein, J. Hobbs, O. Lassila, D. Martin, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara and H. Zeng (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. P. Sheth, J. Cardoso, J. A. Miller, K. J. Kochut and K. Bhukhanwala (2002). "Healthcare Enterprise Process Development and Integration," LSDIS Lab, Department of Computer Science, University of Georgia, Athens, GA, Technical Report.

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.

Bernardo, J. M. and A. F. M. Smith (1994). Bayesian Theory, Wiley.

Bussler, C. (1998). Workflow Instance Scheduling with Project Management Tools. 9th Workshop on Database and Expert Systems Applications DEXA'98, Vienna, Austria, IEEE Computer Society Press. pp. 753-758.

CAPA (1997). "Course Approval Process Automation (CAPA)," LSDIS Lab, Department of Computer Science, University of Georgia, Athens, GA. July 1, 1996 - June 30, 1997.

Cardoso, J. (2002). Quality of Service and Semantic Composition of Workflows. Ph.D. Dissertation. Department of Computer Science, University of Georgia, Athens, GA.

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., Z. Luo, J. Miller, A. Sheth and K. Kochut (2001). Survivability Architecture for Workflow Management Systems. Proceedings of the 39th Annual ACM Southeast Conference, Athens, GA. pp. 207-216.

Chandrasekaran, S., G. Silver, J. A. Miller, J. S. Cardoso and A. P. Sheth (2002). Composite Web Service: Performance Evaluation and Simulation. Proceedings of the 2002 Winter Simulation Conference, San Diego, California. pp. 606-615.

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

Clark, D., S. Shenker and L. Zhang (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 and K. Kuhn (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, M. Duitshof and H. Ensing (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., E. Panagos, H. Pozewaunig and M. Rabinovich (1999). Time Management in Workflow Systems. BIS'99 3rd International Conference on Business Information Systems, Poznan, Poland, Springer Verlag. pp. 265-280.

Fan, M. (1999). Security for the METEOR Workflow Management System. M.Sc. Thesis. Department of Computer Science, University of Georgia, Athens, GA.

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/.

Franceschini, F. (2002). Advanced quality function deployment. Boca Raton, FL, St. Lucie Press.

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

Garvin, D. (1988). Managing Quality: The Strategic and Competitive Edge. New York, Free Press.

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

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

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

Goel, A. L. (1985). "Software reliability models: assumptions, limitations, and applicability." IEEE Transactions on Software Engineering **11**(12): 1411-1423.

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

Hall, D., J. A. Miller, J. Arnold, K. J. Kochut, A. P. Sheth and M. J. Weise (2000). "Using Workflow to Build an Information Management System for a Geographically Distributed Genome Sequence Initiative," LSDIS Lab, Department of Computer Science, University of Georgia, Athens, GA, Technical Report.

Hamlet, D., D. Mason and D. Woit (2001). Theory of Software Component Reliability. 23rd International Conference on Software Engineering ICSE'2001. pp. 361-370.

Hiltunen, M. A., R. D. Schlichting, C. A. Ugarte and G. T. Wong. (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. and R. Y. Moss (1996). Handbook of reliability engineering and management. New York, McGraw Hill.

ISO9000 (2002). ISO9000. International Organization for Standardization, http://www.iso.ch/iso/en/iso9000-14000/iso9000/iso9000index.html.

Kamath, M., G. Alonso, R. Guenthor and C. Mohan (1996). Providing High Availability in Very Large Workflow Management Systems. Proceedings of the 5th International Conference on Extending Database Technology, Avignon. pp. 427-442.

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

Kao, B. and H. GarciaMolina (1993). Deadline assignment in a distributed soft realtime system. Proceedings of the 13th International Conference on Distributed Computing Systems. pp. 428-437.

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

Knuth, D. E. (1971). "An Empirical Study of FORTRAN Programs." Software Practices and Experience **1**(12): 105-134.

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

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

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

Kolarik, W. J. (1995). Creating Quality: Concepts, Systems, Strategies, and Tools. New York, McGraw-Hill.

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.

Lazowska, E. D., J. Zhorjan, S. G. Graham and K. C. Sevcik (1984). Quantitative System Performance: Computer System Analysis Using Queueing Network Models, Prentice Hall.

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. Ph.D. Dissertation. Department of Computer Science, University of Georgia, Athens, GA.

Luo, Z., A. P. Sheth, J. A. Miller and K. J. Kochut (1998). Defeasible Workflow, its Computation, and Exception Handling. Proceedings of 1998 Computer-Supported Cooperative Work (CSCW 1998), Towards Adaptive Workflow Systems Workshop, Seattle, WA. http://ccs.mit.edu/klein/CSCW98/#anchor1609273.

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

Mason, D. and D. Woit (1998). Software system reliability from component reliability. Proceedings of 1998 Workshop on Software Reliability Engineering (SRE'98), Ottawa, Ontario.

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

Miles, M. B. and A. M. Huberman (1994). Qualitative data analysis: an expanded sourcebook. Thousand Oaks, California, Sage Publications.

Miller, J. A., J. S. Cardoso and G. Silver (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., M. Fan, S. Wu, I. B. Arpinar, A. P. Sheth and K. J. Kochut (1999). "Security for the METEOR Workflow Management System," Department of Computer Science, University of Georgia, Athens, GA, Technical Report, pp. 33.

Miller, J. A., R. Nair, Z. Zhang and H. Zhao (1997). JSIM: A Java-Based Simulation and Animation Environment. Proceedings of the 30th Annual Simulation Symposium, Atlanta, GA. pp. 786-793.

Miller, J. A., D. Palaniswami, A. P. Sheth, K. J. Kochut and H. Singh (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.

Miller, J. A., A. F. Seila and X. Xiang (2000). "The JSIM Web-Based Simulation Environment." Future Generation Computer Systems: Special Issue on Web-Based Modeling and Simulation **17**(2): 119-133.

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.

Nahrstedt, K. and J. M. Smith (1996). "Design, Implementation and Experiences of the OMEGA End-point Architecture." IEEE JSAC **14**(7): 1263-1279.

Nair, R., J. A. Miller and Z. Zhang (1996). A Java-Based Query Driven Simulation Environment. Proceedings of the 1996 Winter Simulation Conference, Colorado, CA. pp. 786-793.

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

Parnas, D. L. (1974). On a 'Buzzword': Hierarchical Structure. Proceedings of the IFIP Congress 1974, North Holland. pp. 336-339.

Parnas, D. L. (2001). Software fundamentals: collected papers by David L. Parnas. Boston, Addison-Wesley.

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 and W. Liebhart (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, S., O. Marjanovic and M. E. Orlowska (2000). "Managing Change and Time in Dynamic Workflow Processes." The International Journal of Cooperative Information Systems **9**(1, 2): 93-116.

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

Son, J. H., J. H. Kim and M. H. Kim (2001). "Deadline Allocation in a Time-Constrained Workflow." International Journal of Cooperative Information Systems (IJCIS) **10**(4): 509-530.

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

Tang, J. and J. Veijalainen (1999). "Using Fragmentation To Increase Reliability For Workflow Systems." Society for Design and Process Science **3**(2): 33-48.

Tversky, A. and D. Kahneman (1974). "Judgment under uncertainty: Heuristics and biases." Science **185**: 1124-1131.

Weikum, G. (1999). Towards Guaranteed Quality and Dependability of Information Service. Proceedings of the Conference Datenbanksysteme in Buro, Technik und Wissenschaft, Freiburg, Germany, Springer Verlag. pp. 379-409.

Wheater, S. M. and S. K. Shrivastava (2000). "Reliability Mechanisms in the OPENflow Distributed Workflow System," Department of Computing Science, University of Newcastle upon Tyne Technical Report 31, Esprit LTR Project No. 24962 - C3DS First year Report, pp. 269-288.

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