Implementing QoS Management for Workflow Systems

Jorge Cardoso, Amit Sheth and Krys Kochut
Large Scale Distributed Information Systems (LSDIS) Lab,
Department of Computer Science
University of Georgia, Athens, GA 30602 – USA

Abstract

Workflow management systems (WfMSs) have been used to support various types of business processes. As organizations adopt new working models, such as e-commerce, new challenges arise for workflow systems. One such challenge is that of quality of service (QoS) management. QoS management includes mechanisms that specify, compute, monitor, and control the quality of service of the products or services to be delivered. A good management of QoS directly impacts the success of organizations participating in e-commerce activities by better fulfilling customer expectations and achieving customer satisfaction. In this paper we present an implementation of a comprehensive QoS model for workflows we have specified earlier. While the implementation is being carried out for the METEOR workflow management system, the ideas presented here can also be applied to other workflow systems. In this work we describe the components that have been changed, or added, and discuss how they interact to enable the specification, computation, and monitoring of QoS.

1 Introduction

Organizations are constantly seeking new and innovative information systems to better fulfill their missions and strategic goals. The use of workflow Management Systems (WfMSs) allows organizations to streamline and automate business processes and reengineer their structure, as well as increase efficiency and reduce costs. Workflow systems are also a valuable asset for managing e-commerce applications that span multiple organizations (Sheth, Aalst et al. 1999). As the number of online services increases, workflow systems are needed to coordinate and manage the interaction among web-services (Berners-Lee 2001; Fensel and Bussler 2002).

Organizations operating in modern markets, such as e-commerce, require systematic design, planning, control, and management of business processes. One particular important aspect is the quality of service (QoS) management. Products and services with well-defined specifications must be available to customers. This is especially important since when using the Internet to trade goods, customers do not have a tangible access to

the products to be delivered. A good management of quality leads to the creation of quality products and services, which in turn fulfill customer expectations and achieve customer satisfaction. The customer's expectations and satisfaction can be translated into the quality of service rendered. Equally importantly, QoS is needed as a basis for contracts that govern e-commerce activities between trading partners.

Workflow systems should be viewed as more than just automating or mechanizing tools. They can also be used to analyze, reshape, and reengineer the way business is done. One way to achieve these objectives is through QoS analysis involving such QoS metrics as, time, cost, reliability, and fidelity. At runtime, if the monitoring of a workflow indicates the presence of unsatisfactory QoS metrics, strategies can be employed to redesign, reengineer, or dynamically adapt the workflow.

For organizations, being able to characterize workflows based on their QoS has three direct advantages. First, it allows organizations to translate their vision into their business processes more efficiently, since workflow can be designed according to QoS metrics. Second, it allows for the selection and execution of workflows based on their QoS in order to better fulfill customers' expectations. Third, it also makes possible the monitoring and control of workflows based on QoS, setting up compensation strategies when undesired metrics are identified, or use it as a tool to manage contract commitments.

The requirement of process QoS management is a new challenge for workflow systems. While QoS has been a major concern for networking, real-time applications, and middleware, few research groups have concentrated their efforts on enhancing workflow systems to support workflow quality of service (QoS) capabilities or a subset of them. Most of the research carried out to extend the functionality of workflow systems QoS has only been done in the time dimension, which is only one of the dimensions under the QoS umbrella. Furthermore, the solutions and technologies presented are still preliminary and limited (Eder, Panagos et al. 1999).

Our work in this area started with the definition of a QoS model for workflows (Cardoso, Sheth *et al.* 2002). The model includes four dimensions: time, cost, reliability, and fidelity. These dimensions allow for the specification of non-functional QoS metrics and for the computation of overall workflow QoS based on individual task QoS.

This paper enumerates and describes the enhancements that need to be made to workflow systems to support processes constrained by QoS requirements, such as e-commerce workflows. The enhancements include the development and support of a comprehensive QoS model and the implementation of methodologies (a mathematical model and simulation) to compute and predict workflow QoS. We have developed a stochastic workflow reduction algorithm (SWR) for the step-by-step computation of QoS metrics. Our work has been carried out for the METEOR system to allow the specification, computation, and management of QoS. The support of QoS requires the modification and extension of several workflow system components, and the development of additional modules. While the implementation was made for the METEOR system, and the development is based on a specific conceptual model, the main ideas presented in this study can be applied to the vast majority of workflow systems available (Aalst, Barros et al. 2002).

This paper is structured as follows. In section 2, we present the related work that has been done in the context of QoS management. In section 3, we briefly describe our QoS

model and each of its dimensions. These descriptions will allow for a better understanding of QoS implementation. Section 4 is extensive and describes the modification of existing workflow system components and the creation of new modules that have been developed to support the workflow QoS concept. Each of workflow components and new modules are analyzed individually. Section 5 explains how QoS can be computed, as based on QoS tasks. We briefly present the idea behind one algorithm that we have developed, and we also describe how simulation techniques can be used to compute workflow QoS. In section 6 we explain and describe the role of workflow adaptation and dynamic changes to support a successful QoS management. Finally, section 7 presents our conclusions.

2 Related Work

While QoS has been a major concern for networking (Cruz 1995; Georgiadis, Guerin et al. 1996), real-time applications (Clark, Shenker et al. 1992) and middleware (Zinky, Bakken et al. 1997; Frlund 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 (QoS) specifications and management.

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 an early contribution by considering time and cost. In their approach, a continuous-time Markov chain (CTMC) is used to calculate the time and cost associated with workflow executions. While the research on QoS for WfMS is limited, the research on time management, which is one component of workflow QoS, has been more active and productive. Eder (1999) and Pozewaunig (1997) extend CMP and PERT by annotating workflow graphs with time. At process build-time, instantiation-time, and runtime the annotations are used to check the validity of time constraints. A significant limitation of their approach is that only direct acyclic graphs (DAG) can be modeled, especially because many real-world workflows have cyclic graphs. Cycles are in general used to represent re-work actions or repetitive activities within a workflow. Reichert (1998) and Dadam (2000) also recognize time as an important aspect of workflow execution. In their approach, it is possible to specify a deadline involving minimal and maximal durations for execution of each task. At runtime, the workflow system monitors the specified deadlines and notifies users when deadlines are missed. The system also checks if minimal and maximal time distances between tasks are followed according to initial specifications. Marjanovic and Orlowska (1999) describe a workflow model enriched by modeling constructs and algorithms that check the consistency of workflow temporal constraints. Their work mainly focuses on how to manage workflow changes, while at the same time accounting for temporal constraints. Son and Kim (2001) present a solution for the deadline allocation problem based on queuing networks. Their work also uses graph reduction techniques, but applied to queuing networks.

Recently, researchers have been interested in QoS in the area of Web services. In the DAML-S (DAML-S 2001) specification, use of an ontology allows and facilitates process interoperability between trading partners involved in e-commerce activities. The specification includes tags to specify the quality of service parameters, such as quality guarantees, quality rating, and degree of quality. While DAML-S has identified

specifications for Web service and business processes as a key specification component, the QoS model which should be adopted needs to be significantly improved to supply a realistic solution to its users. One current limitation of the DAML-S' QoS model is that it does not provide a detailed set of classes and properties that represent QoS metrics. The QoS model needs to be extended to allow for a precise characterization of each dimension. Furthermore, a model to compute overall QoS of process specified as composition of Web Services is not provided. The addition of concepts that represent the minimum, average, maximum, and distribution functions associated with dimension, such as cost and duration, will allow for the implementation of algorithms for the automatic computation of QoS metrics of processes, as based on sub-processes' QoS metrics.

3 Workflow Quality of Service

In the work presented here, workflow QoS represents the quantitative and qualitative characteristics of a workflow application which is necessary to achieve a set of initial requirements. Workflow QoS addresses the non-functional issues of workflows, rather than workflow process operations. 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 be a 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, and therefore it should be embedded in workflow specifications and WfMSs.

Quality of service can be characterized along various dimensions. We have investigated related work to decide which dimensions would be relevant in composing our QoS model. Our research targeted two distinct areas: operations management in organizations (Garvin 1988; Stalk and Hout 1990; Rommel 1995) and quality of service for software systems, which include networking (Cruz 1995; Georgiadis, Guerin et al. 1996; Nahrstedt and Smith 1996), middleware areas (Zinky, Bakken et al. 1997; Frlund and Koistinen 1998; Hiltunen, Schlichting et al. 2000), and real-time applications (Clark, Shenker et al. 1992). The study of those two areas is important, since workflow systems are widely used to model organizational business processes, and since workflow systems are themselves software systems.

3.1 QoS Model

Weikum (1999) divided information services QoS into three categories: system centric, process centric, and information centric. Based on previous studies and on our experience in the workflow domain, we have constructed a QoS model that includes system and process categories. Our model is composed of four dimensions: *time*, *cost*, *fidelity*, and *reliability*.

Time (T) is a common and universal measure of performance. For workflow systems, it can be defined as the total time needed by an instance in order to transform a set of inputs into outputs. Task response time (T) corresponds to the time an instance takes to be

processed by a task. The *task response time* can be broken down into major components which include: process time, queuing delay, setup delay, and synchronization delay.

Cost (C) represents the cost associated with the execution of workflow tasks. During workflow design, prior to workflow instantiation, and during workflow execution it is necessary to estimate the cost of the execution to guarantee that financial plans are followed. Task cost is the cost incurred when a task *t* is executed; it can be broken down into major components, which include realization cost and enactment cost.

We view **Fidelity** (F) as a function of effective design; it refers to an intrinsic property or characteristic of a good produced or of a 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 can be subjective. Nevertheless, the fidelity of workflows should be predicted when possible and carefully controlled when needed. Workflow tasks have a fidelity vector dimension composed by a set of fidelity attributes ($F(t).a_i$) to reflect, qualify, and quantify task operations. Each fidelity attribute refers to a property or characteristic of the product being created, transformed, or analyzed. Fidelity attributes are used by the workflow system to compute how well workflows, instances, and tasks are meeting user specifications. For automatic tasks (Kochut, Sheth et al. 1999) the fidelity can be set automatically. For a human task, we must really on the person in charge of the task realization to set the fidelity attributes.

Task **Reliability** (R) corresponds to the likelihood that the components will perform when the user demands them; it is a function of the failure rate. Depending on the workflow system and task conceptual model, tasks instances can be placed into different states, typically described by a state transition diagram (task structure) during their execution. Two final states exist. One represents the success of a task realization, and the other represents the failure of a task realization. The reliability dimension is a function of the number of times the success state is reached and the number of times the failure state is reached.

4 Workflow QoS Implementation

The QoS model that we have developed is being implemented for the METEOR workflow management system. The METEOR project is represented by both a research system (METEOR 2002), and a suite of commercial systems that provide an open system based, high-end workflow management solution, as well as an enterprise application integration infrastructure. The work discussed in this paper is part of the research system and is not part of any commercial product yet.

METEOR's architecture includes four services: Enactment, Manager, Builder, and Repository. The enactment service includes two systems: ORBWork (Kochut, Sheth et al. 1999) and WebWork (Miller, Palaniswami et al. 1998). The task of the enactment service is to provide an execution environment for processing workflow instances. Both ORBWork and WebWork use fully distributed implementations. WebWork, an entirely Web-based enactment service, is a comparatively light-weight implementation that is well-suited for less complex applications that involve limited data exchange and do not need to be dynamically changed. ORBWork is targeted for more demanding, mission-

critical enterprise applications requiring high salability, robustness and dynamic adaptation. The current version of ORBWork has been designed to address a variety of shortcomings found in today's workflow systems. It supports interoperability standards such as JFLOW (OMG 1998) and SWAP (Swenson 1998). Although we started with the use of open standards such as Java and CORBA to make it a good candidate for interoperating with existing systems from a variety of distributed and heterogeneous computing environments, recently a Java-only version (replacing CORBA with RMI) has also been completed. With recently added modules, it also includes a repository for reuse (Song 2001), dynamic changes (Chen 2000; Tripathy 2000) at the instance level and an exception-handling mechanism (Luo 2000). ORBWork has been used in prototyping and deploying workflow applications in various domains, such as bio-informatics (Hall, Miller et al. 2000), healthcare (Anyanwu, Sheth et al. 1999), telecommunications (Luo 2000), defense (Kang, Froscher et al. 1999), and university administration (CAPA 1997).

In this section we describe the components that make up the METEOR system and the components that have been modified, extended, and created to enable QoS management. Changes have been made to four services: the Enactment, the Manager, the Builder, and the Repository. These components and their relationship to the overall workflow system are illustrated in Figure 1.

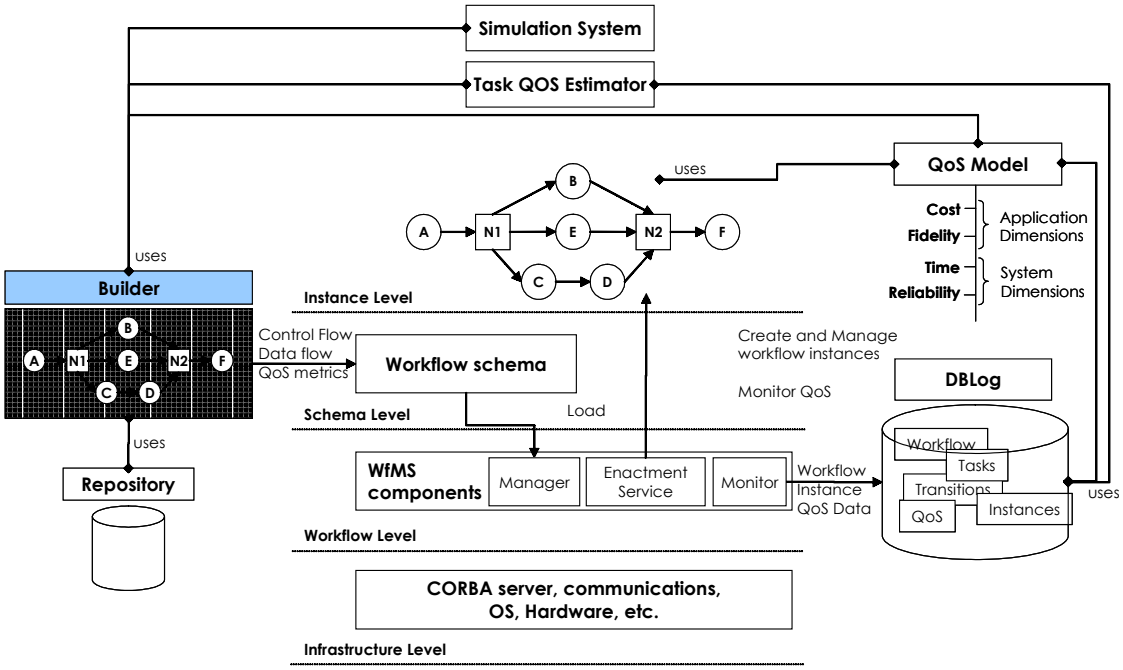

Figure 1 – QoS Management Architecture

4.1 Enactment Service

In this section we describe the modifications that have been made to the ORBWork enactment system. The components analyzed include task schedulers, task managers, and monitors.

In ORBWork enactment service, task schedulers, task managers, and tasks are responsible for managing runtime QoS metrics. From the implementation point of view, we divide the management of the QoS dimensions into two classes: the system and the application dimensions. The system dimensions (time and reliability) are the responsibility of task schedulers, while the application dimensions (cost and fidelity) are the responsibility of task managers and tasks. Since task schedulers decide the starting time of task execution and are notified when tasks are complete, they set the time dimension of the QoS. Additionally, the supervision of tasks completion puts them in charge of managing the reliability dimension. These two dimensions are called system dimensions because it is a system component (the enactment system) that is responsible for registering the time and reliability metrics at runtime. For the cost and fidelity dimensions, task managers are the candidate components since they include the necessary functions to initialize tasks with estimated QoS metrics. The cost and fidelity dimensions are called application dimensions since they are manipulated and modified by a task realization.

4.1.1 Task Schedulers

ORBWork follows a fully distributed scheduling strategy. The scheduling responsibilities are shared among a number of participating task schedulers, according to workflow definitions. The distributed schedulers maintain a workflow data specification that has been received during workflow installation. Each task scheduler provides a well-constrained subset of the HTTP protocol and thus implements a lightweight, local Web server. The scheduler accesses workflow specifications through the HTTP protocol, directly from specification files or from the repository. Each set of task specifications includes input dependency (input transitions), output transitions with their associated conditions, and date objects that are sent into and out of the task. As discussed previously, task schedulers are responsible for managing the time and reliability dimensions. We discuss each one of these separately in the following sections.

Managing Time

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

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

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

To efficiently manage the time dimension, workflow systems must register values for each of the functions involved in the calculation of task response time (T). Currently,

we register values for all the functions, except for the setup delay. The time dimension has its values set according to the task structure illustrated in Figure 2. Each state has been mapped to one of the functions that compose the time dimension. ORBWork system follows this task structure to represent workflow task execution behavior (Krishnakumar and Sheth 1995). To more effectively support QoS management, the original structure has been extended, with the inclusion of the *Pre-Init*, as shown in Figure 2.

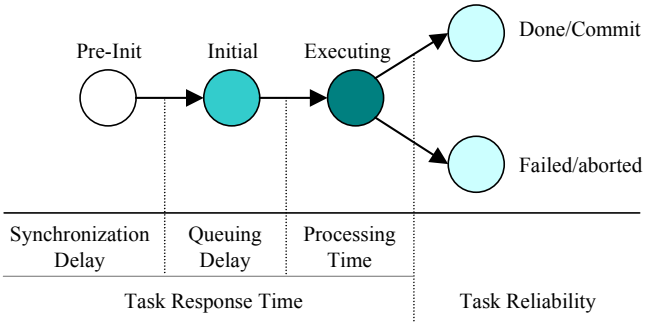

Figure 2 – Revised task structure (extended from (Krishnakumar and Sheth 1995))

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

As for the synchronization delay time, the queuing time is the difference between the time a task leaves and enters the *initial* state. A task in the *initial* state indicates that the task is in a queue waiting to be scheduled (by its task scheduler). ORBWork task schedulers treat their queues with a FIFO policy. One interesting queuing policy variation is associated with the scheduling of human-tasks. For a human-task instance, being in the *initial* state means that the task has been placed in a worklist for human processing. A user can select any human-task in a worklist, as long as the user role matches the task role. In this case, the queuing policy is SIRO (Serve In Random Order). Depending on the workflow system, other useful queuing policies can be used, such as priority queues. When a task instance enters a queue a time-stamp is attached to it. When the task is removed from the queue for scheduling, another time-stamp is attached to it so that the total queuing time can be calculated later.

When a task is ready to be executed it transits to the *executing* state. As with the previous calculations, the time a task remains in this state corresponds to the processing time.

Managing Reliability

During a task realization, a number of undesirable events may occur.. Depending on the successful or unsuccessful execution of a task, it can be placed in the done or fail state

(for non-transactional tasks) and commit or abort (for transactional tasks). The former state indicates that the task execution was unsuccessful, while the latter state indicates that a task is executed successfully (Krishnakumar and Sheth 1995).

When an undesirable event occurs, an exception is generated. An exception is viewed as an occurrence of some abnormal event that the underlying workflow management system can detect and react to. If an exception occurs during the invocation of a task realization, its task enters the fail/abort state. In our implementation, it is the responsibility of task schedulers to identify the final state of a task execution in order to subsequently set the reliability dimension. Later this information is used to compute the failure rate, which is the ratio between the number of times the failed/aborted state is reached and the number of times state done/committed is reached. To describe task reliability we follow a discrete-time modeling approach. Discrete-time models are adequate for systems that respond to occasional demands such as database systems. We use the stable reliability model proposed by Nelson (1973), for which the reliability of a task t is $R(t) = 1$ - failure rate.

4.1.2 Task Managers and Tasks

When a task is ready to execute, a task scheduler activates an associated task manager. The task manager oversees the execution of the task itself. Task managers are implemented as an object and are classified as transactional or non-transactional, depending on the task managed. Human tasks do not have an associated task manager.

Once activated, the task manager stays active until the task itself completes. Once the task has completed or terminated prematurely with a fault, the task manager notifies its task scheduler. The task manager is responsible for creating and initializing a QoS data structure from QoS specifications (for the cost and fidelity dimensions) for the task overseen. When the supervised task starts its execution, the data structure is transferred to it. If the task is a non-transactional one (typically performed by a computer program), a set of methods is available to programmatically change the initial QoS estimates. No methods are supplied to change the time and reliability dimensions since the task schedulers are responsible for controlling these dimensions. For transactional tasks (*i.e.* a database operation), only the time and reliability dimensions are dynamically set at runtime. The cost and fidelity dimensions, once initialized from the QoS specifications, cannot be changed. This is because database systems do not make available information evaluating the cost and the fidelity of the operations executed. Once the task completes its execution, the QoS data structure is transferred back to the task manager, and later from the task manager to the task scheduler. The only responsibility of the task scheduler will be to incorporate the metrics registered for the time and reliability dimensions (see section 4.1.1) into the QoS data structure and send it to the monitor to be processed (see next section).

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

completes its realization, the task scheduler parses the interface page(s) and retrieves the new QoS metrics that the user may have modified.

4.1.3 Monitor

When workflows are installed and instances are executed, the enactment system generates information messages (events) describing the activities being carried out. The monitor is an independent component represented by an object that records all of the events for all of the workflows being processed by the enactment service. Depending on the system setup parameters, the ORBWork monitor can display the events it receives to the console or store them in a readable log file. To extend the functionality and usability of the monitor two distinct APIs have been developed: the HTTPlog and the DBlog.

The first one uses the HTTP protocol to send status information from the ORBWork monitor to remote clients. The information can be viewed remotely, using a monitor client. This is particularly suitable for administrators that need to periodically check the status of running instances. The second API, the DBlog, has been developed to store the status and QoS events generated in a relational database. When a workflow is installed and executed, task QoS estimates, runtime QoS metrics, and transition frequencies are stored in the database. The stored information will be later utilized to create a QoS profile for the tasks and to enable the computation of the workflow QoS.

4.1.4 DBlog

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

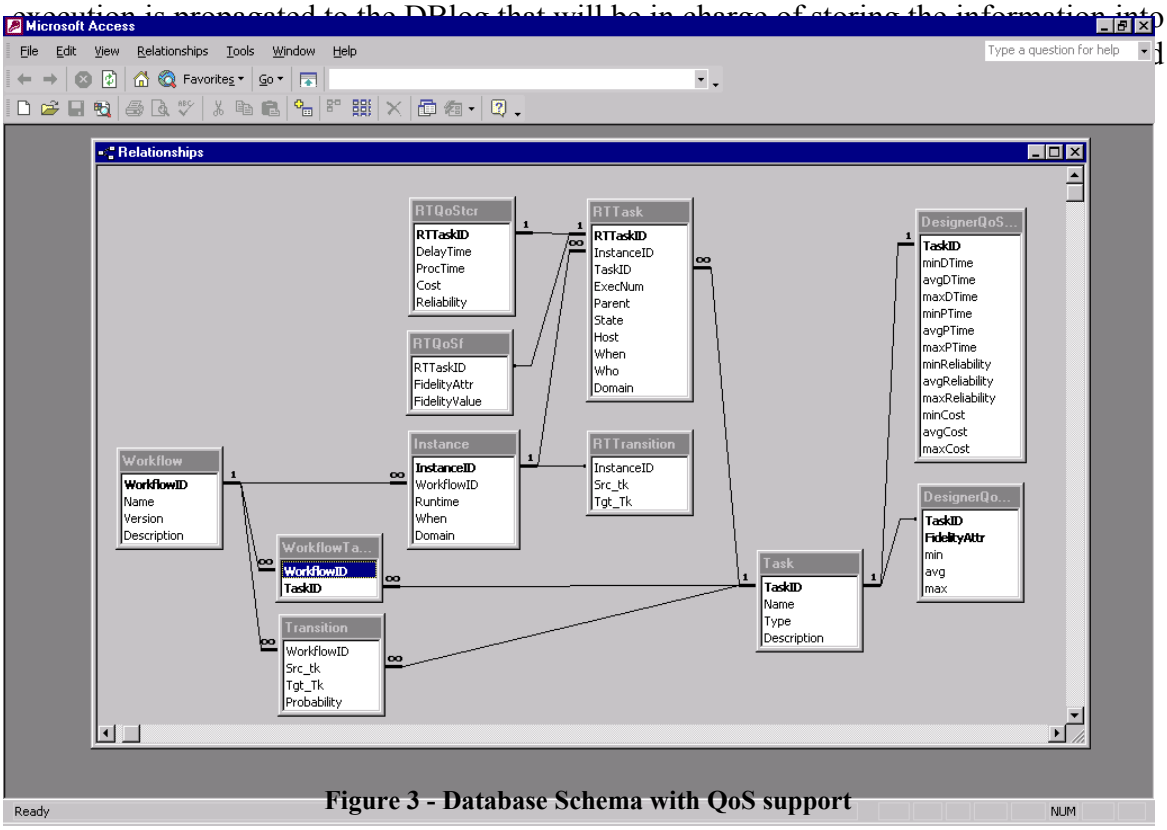

Figure 3 - Database Schema with QoS support

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

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

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

4.2 Manager

The manager is used to install and administer workflow definitions (schema), and to start workflow instances. When a workflow is installed, the manager activates all of the necessary task schedulers to carry out the execution of instances. The manager is implemented as an object and has an interface that allows clients to interact with it. The manager does not participate in any task scheduling activities. It is only necessary at the time a new workflow is installed or modified. When a workflow is installed, trace messages are send to the monitor indicating the workflow installed and its associated tasks. The information send to the monitor also includes the initial QoS estimates that the user has set during the workflow design. When the monitor receives this information (workflow topology, tasks, and QoS estimates), it uses the DBlog interface to store it in a database for later QoS processing.

4.3 Workflow Builder

The workflow builder tool is used to graphically design and specify a workflow. In most cases, after a workflow design no extra work is necessary and it can be converted automatically to an application by a code generator. The builder is used to specify workflow topology, tasks, transitions (control flow and data flow), data objects, task invocation, roles, and security domains (Kang, Park et al. 2001). During the design phase, the designer is shielded from the underlying details of the runtime environment and

infrastructure, separating the workflow definition from the enactment service on which it will be installed and executed. To support workflow QoS management the designer must be able to set estimates for transition probabilities and QoS estimates for tasks. This information is later combined with historical data, which plays a larger role as more instances are executed, to create a runtime QoS model for tasks and a probability model for transitions.

The workflow model and the task model have been extended to support the specification of QoS metrics. To support these extensions, the builder has been enhanced to allow designers to associate probabilities with transitions and to make possible the specification of initial QoS metrics for tasks (see section 4.3.1). Previously, the workflow model only included data flow mappings associated with transitions. The association of probabilities with transitions transforms a workflow into a stochastic workflow. The stochastic information indicates the probability of a transition being fired at runtime. The QoS model specified for each task and transitions probabilities are embedded into the workflow definition and stored in XML format.

4.3.1 Setting Initial Task QoS Estimates

At design time, each task receives information which includes its type, input and output parameters, input and output logic, realization, exceptions generated, *etc*. All this information makes up the task model. The task model has been extended to accommodate the QoS model. Task QoS is initialized at design time and re-computed at runtime when tasks are executed. During the graphical construction of a workflow process, each task receives information estimating its quality of service behavior at runtime. This includes information about its cost, time (duration), reliability, and fidelity.

The task QoS estimates are composed of two classes of information: basic and distributional. The basic class associates with each task QoS dimension the estimates of the minimum, average, and maximum values that the dimension can take. For example, for the cost dimension, it corresponds to the minimum, average, and maximum costs associated with the execution of a task. The second class, the distributional class, corresponds to the specification of a distribution function (such as Exponential, Normal, Gamma, Weibull, and Uniform) which statistically describes tasks behavior at runtime. For example, the time QoS dimension of a task can be describe by using the normal or uniform distribution function. Figure 4 illustrates the graphical interface that is used to specify the basic and distributional information to setup initial QoS metrics.

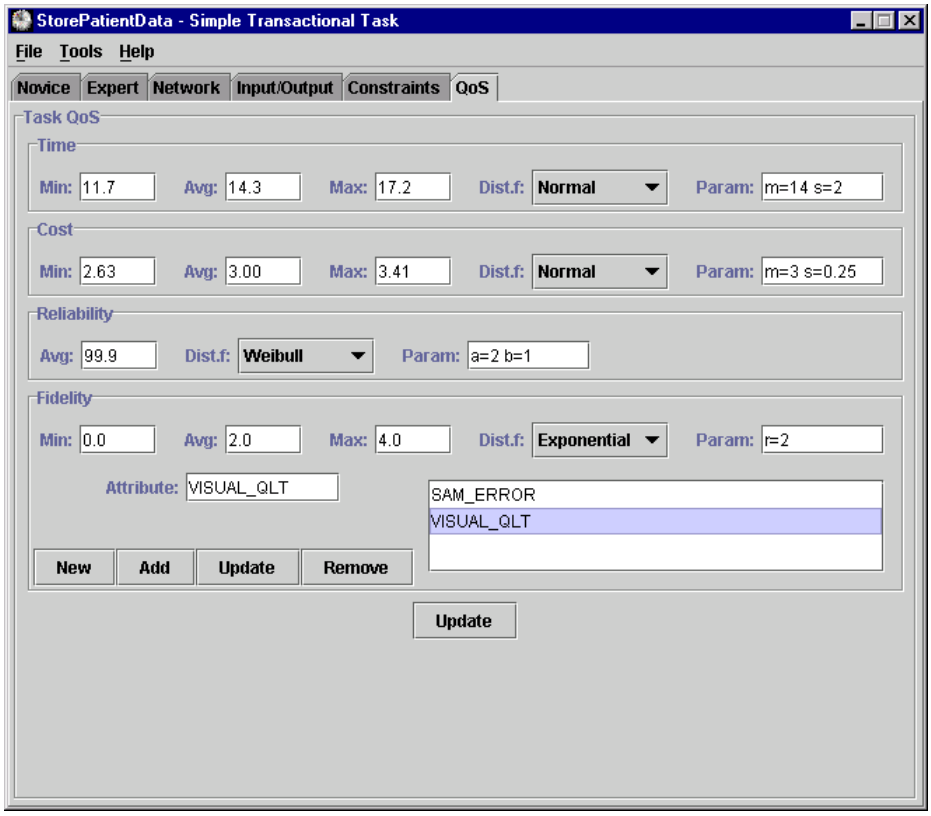

Figure 4 – Task QoS basic and distributional class

The values specified in the basic class are typically used by mathematical methods to compute and predict workflow QoS metrics (see SWR algorithm in section 9 and in Appendix), 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 user typically applies QoS models presented in Cardoso, Miller et al. (2002). We recognize that the specification of cost, time, fidelity, and reliability is a complex operation, and when not carried out properly can lead to the specification of incorrect values.

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

4.3.2 Re-Computing QoS Estimates

The initial QoS specifications may not be valid over time. To overcome this difficulty we re-compute task QoS values for the basic class, based on previous executions. The same applies for transitions. The distributional class also needs to have its distribution re-computed. This involves the analysis of runtime QoS metrics to make sure that the QoS distribution functions associated with a task remain valid or need to be modified.

The re-computation of QoS estimates for tasks and for transition probabilities is done based on runtime data generated from past workflow executions that have been stored in the database log (section 4.1.4). We have developed a QoS Estimator module that lies between the builder and the database log. The QoS Estimator creates a QoS model for tasks based on the information stored in the DBlog. It also calculates transition probability functions based on the transitions enabled at runtime. Figure 5 illustrate the architecture of the QoS Estimator module. When a workflow is being designed, if the tasks selected to compose the workflow have been previously executed, then their QoS metrics are re-computed automatically using the QoS Estimator module.

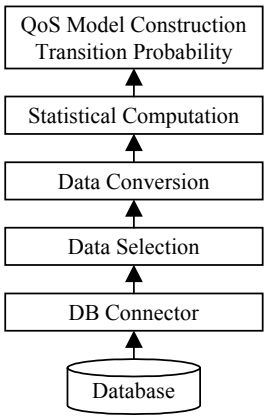

Figure 5 – QoS Estimator Module

DB connector

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

Data Selection

The data selection component allows for the selection of task QoS metrics, as defined by the designer and tasks previously executed. Four distinct selection modes exist, and for each one a specific selection function has been constructed. The functions are shown in Table 1. The component can select tasks QoS metrics from information introduced by the user at design time, from tasks executed in the context of any workflow, from tasks executed in the context of a specific workflow *w*, and from tasks executed from a particular instance *i* of workflow *w*.

Selection function	Description
UD_Select(*t*)	Selects the designer defined QoS metrics of task *t* specified by the designer in the basic class.
RT_Select(*t*)	Selects the runtime QoS metrics of all the executions of task *t*.
RT_Select(*t*, *w*)	Selects the runtime QoS metrics of all the executions of task *t* in any instance of workflow *w*.
RT_Select(*t*, *w*, *i*)	Selects the runtime QoS metrics of all the executions of task *t* in instance *i* of workflow *w*.

Table 1 – Select functions of the Data Selection Component

Data Conversion

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

Statistical Computation

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

Four distinct functions have been developed to compute estimates for the tasks selected in the previous step; these are shown in Table 2. Each function is to be used when computing QoS dimensions and corresponds to four scenarios that can occur. The first function is utilized to retrieve, for a specific task *t* and a particular dimension *Dim*, the average specified by the designer. This function is used when QoS estimates are needed and no runtime QoS information is available. The second function calculates the average of dimension *Dim* metrics for task *t*, based on all task *t* executions, independently of the workflow that has executed it. The third function calculates the average of a task *t* dimension *Dim*, based on all the times task *t* was executed in any instance from workflow *w*. Finally, the last function (d) calculates the average of the dimension *Dim* of all the task *t* executions, from instance *i* of workflow *w*. This scenario can only occur when loops exist in a workflow, and they often do.

	Function	Description
a)	Designer Average$_{Dim}(t)$	Average specified by the designer in the basic class for dimension Dim.
b)	Multi-Workflow Average$_{Dim}$ (t)	Computes the average of the dimension *Dim* of all the executions of task *t*.
c)	Workflow Average$_{Dim}(t, w)$	Computes the average of the dimension *Dim* of all the executions of task *t* in any instance of workflow *w*.
d)	Instance Average$_{Dim}(t, w, i)$	Computes the average of the dimension *Dim* of all the executions of task *t* in instances *i* of workflow *w*.

Table 2 – Designer, multi-workflow, workflow and instance average

Similar to the functions used to compute averages as shown in Table 2 we also support functions to compute the minimum and maximum for QoS dimensions.

QoS Model Construction

The QoS Model Construction component uses the information computed in the statistical computation component and applies the functions presented in Table 3 in order to re-compute a QoS model for tasks. The weights wi_j are set manually, and they reflect the degree of correlation between the workflow under analysis and other workflows for which a set of common tasks is shared.

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

Table 3 – QoS dimensions re-computed at runtime

Let us assume that we have an instance *i* of workflow *w* running, and 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 the 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 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.

The method used to re-compute transitions' probability follows the same lines as for 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, the

only information available. 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* and from the executed instances.

Figure 6 shows the graphical user interface available to set the QoS functions and their associated weights, and to visualize the QoS estimates automatically computed for workflows, instances, tasks, and transitions. The QoS computation is carried out using the SWR algorithm (see section 9).

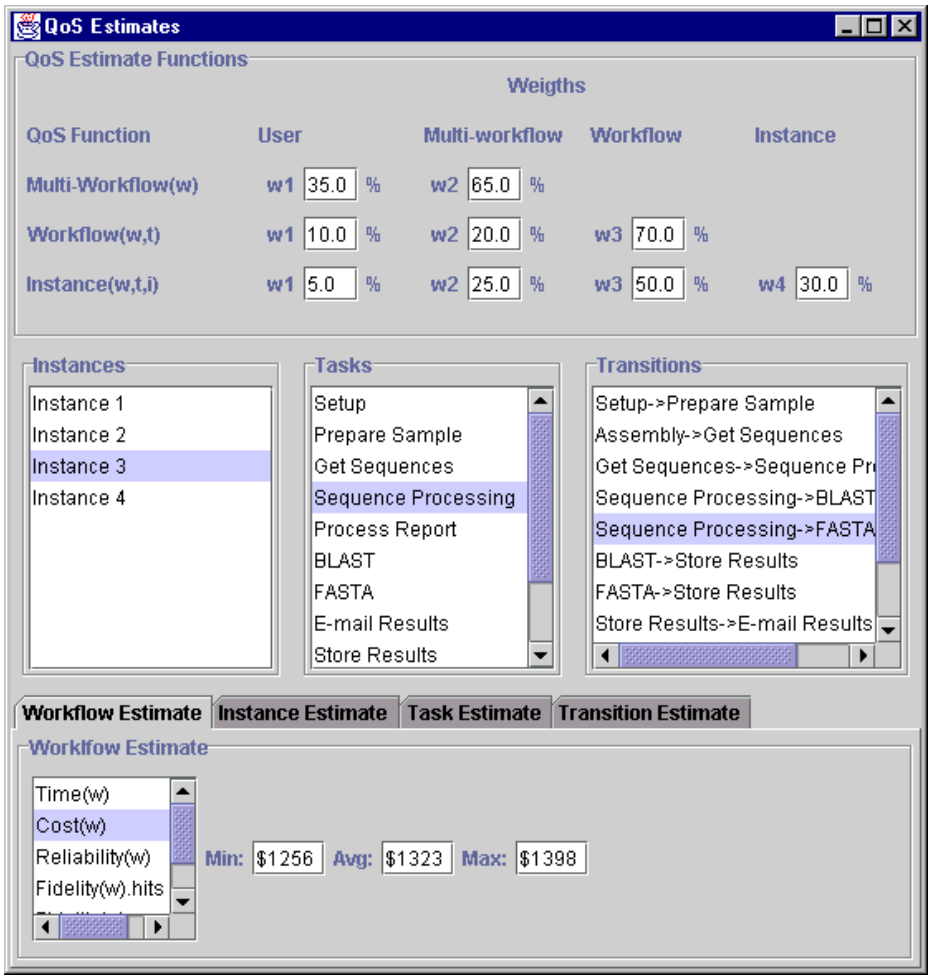

Figure 6 – GUI to calculate QoS estimates

4.4 Workflow Repository Service

Our workflow builder is coupled with a repository. The repository is responsible for maintaining information about workflow definitions and associated workflow applications. The repository tool allows users to retrieve, update, and store workflow definitions (Song 2001). A user can browse the contents of the repository and find already existing workflow definitions fragments (either sub-workflows or individual tasks) to be incorporated into a workflow being created. The repository service is also available to the enactment service; it provides the necessary information about a

workflow application to be started. The repository supplies a practical and efficient access to workflow definitions, based on queries. In order to query and search the repository based on QoS requirements the repository needs to be extended. This functionality is useful since it allows users to find tasks with specific QoS metrics when composing workflows with initial QoS requirements, such as low cost or high availability. While we have not implemented this feature yet, we consider it indispensable for QoS based workflow composition; and will support it in a future version of this system.

5 Workflow QoS Analysis and Simulation

Having made a graphical (abstract) representation of an organizational process model, a workflow contains information which can be used as a basis for analysis. The analysis focuses on workflow topology (tasks and transitions) and on the QoS metrics. Analyzing workflows allows us to gather information about workflow QoS metrics, which include processing time, delay time, cost, fidelity, and reliability. The QoS information makes workflow structures more transparent and quantifiable, allowing inefficiencies and performance problems such as bottlenecks, to be found.

We describe two methods that the builder can use 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 yields results which may not be as accurate as the ones obtained with simulation. Workflow modeling is a continuous activity, where processes are continuously improved to increase efficiency and meet organizational goals and strategies.

5.1 Mathematical Modeling

Comprehensive solutions to the challenges encountered in synthesizing QoS for composite services have been discussed in detail (Cardoso, Sheth *et al.* 2002). We have developed a stochastic workflow reduction algorithm (SWR) for step-by-step computation of aggregate QoS properties. The code, examples, and documentation for the algorithm can be found in Cardoso (2002). At each step a reduction rule is applied to shrink the workflow. Also at each step, the response time (T), cost (C), fidelity (F) and reliability (R) of the tasks involved is computed. Additional task metrics can also be individually computed, such as task queuing time and setup time. The reduction process is continued until only one atomic task (Kochut, Sheth et al. 1999) is left in a workflow. 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 composite service (*i.e.* workflow) corresponds to the set of inverse operations that can be used to construct a workflow of services. We have decided to allow only the construction of workflows based on a set of predefined construction rules to protect users from designing invalid workflows. Invalid workflows contain design errors, such as non-termination, deadlocks, and the split of instances (Aalst 1999). To compute QoS metrics, we use a set of six distinct reduction rules: (1) sequential, (2) parallel, (3) conditional, (4) fault-

tolerant, (5) loop, and (6) network. As an illustration, we will show how reduction works for a parallel system of tasks.

Reduction of a Parallel System. Figure 7 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 transitions 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. In a parallel system, the probabilities of $p_{a1}, p_{a2}, \ldots, p_{1n}$ and $p_{1b}, p_{2b}, \ldots, p_{nb}$ are all equal to 1.

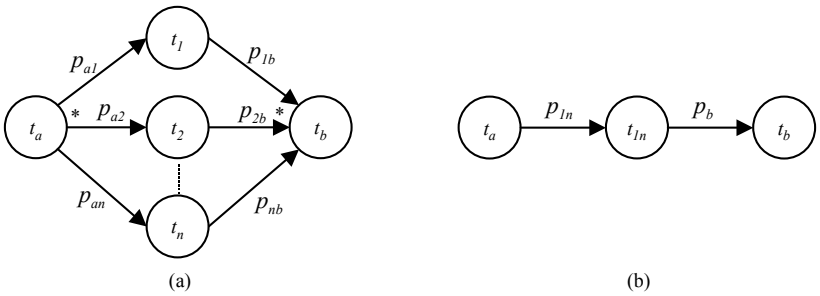

Figure 7 - Parallel system reduction

After applying the reduction, the QoS of tasks t_a and t_b remain unchanged, and $p_{1n} = p_b = 1$. To compute the QoS for this reduction the following formula are applied:

$$T(t_{1n}) = \text{Max}_{i \in 1 \le i \le n} \{T(t_i)\}$$

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

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

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

When a workflow needs to be analyzed, the builder converts the workflow data structure supported by the builder to one supported by the SWR algorithm. Once a workflow is in a suitable data format and each task has their QoS metrics and transition probabilities computed, it is transferred to the SWR algorithm. The algorithm outputs a single atomic task which contains the QoS metrics corresponding to the input workflow.

5.2 Simulation Models

While mathematical methods can be effectively used, another alternative is to utilize simulation analysis (Miller, Cardoso et al. 2002). Simulation can play an important role in fine-tuning tuning the quality of service metrics of workflows, by exploring “what-if" questions. 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 of 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. For modeling purposes, a workflow can be abstractly represented by using directed graphs (*e.g.*, one for control flow and one for data flow, or one for both). Since both models are represented as directed graphs, interoperation is facilitated. In order to carry out a simulation, the appropriate workflow model is retrieved from the repository, and the distribution functions defined in the QoS distributional class (see section 4.3.1) are used to create a JSIM simulation model specification. The simulation model is displayed graphically and then executed/animated. Statistical results are collected and displayed, indicating workflows QoS.

6 QoS Modeling and Adaptation

In order to complete a workflow according to initial QoS requirements, it is necessary to expect to adapt a workflow in response to unexpected delays, technical conditions, and changes in the environment. Long running workflow applications require support for dynamic reconfiguration since machines fail, services are moved or withdrawn and user requirements change. It is also understandable that the unpredictable nature of the surrounding environment has an important impact on the QoS of business processes. The environment can be characterized as heterogeneous and is affected, in a global perspective, by events such as changes in global markets, new company policies, and new laws and regulations.

When workflow adaptation is necessary, a set of potential alternatives is generated, with the objective of dynamically changing a process so it can continue to meet initial QoS 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. The evaluation of a set of different adaptive procedures can be carried out using the two methods described previously. A system administrator can rely solely on the designer tool (builder) to compose and evaluate adaptive strategies. In this case it will be using the mathematical model and *SWR* algorithm introduced in section 5.1. Or it can decide to use a simulation system (see section 5.2) to calculate and estimate QoS metrics.

For example, when a workflow becomes unavailable due to the malfunction of its components, QoS constraints, such as time, will most likely be violated if no intervention is carried out to adapt the workflow. Of course, more than one adaptation strategy may exist. Thus, it is indispensable to evaluate the alternatives that can be applied to correct the process such as it continues to exhibit the required QoS.

As Charles Darwin mentioned –"It is not the strongest species that survive, or the most intelligent, but the one most responsive to change". Adaptation characterizes the ability of a system to adjust to environmental conditions. The importance of adaptation

has been recognized in several areas, that include software (Margaret 1995; Heineman 1998), database systems and mobile systems (Zukunft 1997), and fault-tolerant systems (Hiltunen and Schlichting 1996). In the domain of workflow management systems, it is important to develop mechanisms to carry out adaptation, which are driven by QoS requirements. This permits workflow systems to be prepared to adapt themselves to a range of different business and organization settings and also to a changing context (Han, Sheth et al. 1998). Having these requirements in mind we have developed a module that allows ORBWork system to be an adaptable system.

We have implemented an interface that permits the realization of dynamic change of instances in a consistent manner (Chen 2000). The implementation guarantees that all consistency constraints that have been ensured prior to dynamic changes are also ensured after the workflow instances have been modified (Reichert and Dadam 1998). This interface provides indispensable functions to support the adaptation of workflow instances. An administrator or an application can access the interface to make the necessary change to a workflow to adjust its QoS metrics. Supporting dynamic changes significantly increases the ability of a WfMS to follow QoS constraints and user requirements, allowing the system to cope with all kind of unplanned events during the execution of the business process.

7 Conclusions

Organizations operating in global and competitive markets require a high level of quality of service management. The use of workflow systems to automate, support, coordinate, and manage business processes enables organizations to reduce costs and increase efficiency. Workflow systems should be viewed as more than just automating or mechanizing driving forces. They should be used to reshape and re-engineer the way business is done. One way to achieve continuous process improvement is to view and analyze processes from a QoS perspective. This allows workflows to be designed and adapted according to quality of service constraints drawn from organizational goals and strategies. A good management of QoS leads to the creation of quality products and services, which in turn fulfills customer expectations and achieves customer satisfaction. This becomes increasingly important when workflow systems are used in new organizational and trading models, such as in virtual organizations and e-commerce activities that span organizational boundaries.

While QoS management is of a high importance to organizations, current WfMSs and workflow applications do not provide full solutions to support QoS. Two research areas need to be explored. On one hand, a good theoretical QoS model is necessary to formally specify, represent, and calculate QoS metrics. On the other hand, experimental workflow systems need to be developed to identify the challenges and difficulties that the implementation of QoS management faces. We have already developed a QoS theoretical model, and in this paper we explain how the model was implemented in the METEOR system.

The support of QoS management requires the modification and extension of most of workflow system components. This includes the enactment system, the workflow builder (or designer), the monitor, the code generator, the repository, the workflow model, and the task model. Additionally, new components need to be implemented, such as a QoS

estimator module to create QoS estimates for tasks and probabilities for transitions. The monitor needs an additional interface so that runtime tasks QoS metrics are propagated and logged into a database for data processing purposes.

Algorithms and methods are necessary to predict overall workflow QoS metrics. For this purpose, we present a mathematical model and explain how simulation can be used to calculate and predict workflow QoS. Both approaches enable a predictive computation of workflows QoS based on tasks QoS estimates. The mathematical method is computationally faster, but yields results which may not be as precise as the ones obtained with simulation. The choice of the method is based on a tradeoff between time and the accuracy of results.

8 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.

Anyanwu, K., A. P. Sheth, J. A. Miller, K. J. Kochut and K. Bhukhanwala (1999). "Healthcare Enterprise Process Development and Integration.," LSDIS Lab, Department of Computer Science, University of Georgia, Athens, GA, Technical Report.

Berners-Lee, T. (2001). Keynote presentation on web services and the future of the web. Software Development Expo 2001 Visionary Keynote, http://www.technetcast.com/tnc_play_stream.html?stream_id=616.

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). 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., A. Sheth and J. Miller (2002). Workflow Quality of Service. International Conference on Enterprise Integration and Modeling Technology and International Enterprise Modeling Conference (ICEIMT/IEMC'02), Valencia, Spain, Kluwer Publishers.

Chen, Y. (2000). Design and Implementation of Dynamic Process Definition Modifications in OrbWork Enactment System. M.Sc. Thesis. Department of Computer Science, University of Georgia, Athens, GA.

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.

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.

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

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.

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.

Han, Y., A. P. Sheth and C. Bussler (1998). A Taxonomy of Adaptive Workflow Management. Workshop of the ACM 1998 Conference on Computer Supported Cooperative Work, Seattle, WA.

Heineman, G. T. (1998). Adaptation and Software Architecture. Third Annual International Workshop on Software Architecture, Orlando, Florida. pp. 61-64.

Hiltunen, M. A. and R. D. Schlichting (1996). "Adaptive Distributed Fault-Tolerant Systems." International Journal of Computer Systems Science and Engineering **11**(5): 125-133.

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.

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. pp. 271-285.

Kang, M. H., J. S. Park and J. N. Froscher (2001). Access Control Mechanisms for Inter-organizational Workflows. Proceedings of 6th ACM Symposium on Access Control

Models and Technologies, Chantilly, VA.

Klingemann, J., J. Wäsch 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.

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

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

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.

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.

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.

Margaret, D. J. (1995). Adaptable, Reusable Code. Proceedings of the 17th International Conference on Software Engineering on Symposium on Software Reusability, Seattle, WA. pp. 38-46.

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

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

Miller, J. A., J. S. Cardoso 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., 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.

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.

OMG (1998). BODTF RFP #2 Submission, Workflow Management Facility, Revised Submission, ftp://ftp.omg.org/pub/docs/bom/98-06-07.pdf.

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.

Sheth, A. P., W. v. d. Aalst and I. B. Arpinar (1999). "Processes Driving the Networked Economy." IEEE Concurrency **7**(3): 18-31.

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.

Song, M. (2001). RepoX: A Repository for Workflow Designs and Specifications. M.Sc. Department of Computer Science, University of Georgia, Athens.

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

Swenson, K. (1998). SWAP - Simple Workflow Access Protocol.

Tripathy, S. (2000). WFTP: design and a RMI implementation of a workflow transport protocol for ORBWork. M.Sc. Thesis. Department of Computer Science, University of Georgia, Athens, GA.

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.

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.

Zukunft, O. (1997). Rule Based Adaptation in Mobile Database Systems. Proceedings of the 1997 ACM Symposium on Applied Computing, San Jose, California. pp. 310-317.

9 Appendix

The SWR (Stochastic Workflow Reduction) algorithm uses the set of reduction rules presented in (Cardoso, Sheth *et al.* 2002) to compute workflow QoS metrics. The algorithm iteratively applies the reduction rules to a workflow until only one atomic task remains. At each iteration, the response time (T), cost (C), reliability (R), and fidelity (F) of the tasks involved is computed. Additional task metrics can also be computed, such as task queue time and setup time. If at any point no more reduction rules can be applied and

the size of the workflow is greater than 1, then the initial workflow design was incorrect. An outline of the algorithm is presented in Listing 1.

```
QoS SWR (workflow wf) begin
    boolean changes = true;

    while changes begin
        changes = false;

        forall task in wf and  no changes begin

            changes = applySequentialRule(wf, task);
            if changes continue;

            changes = applyParallelRule(wf, task);
            if changes continue;

            changes = applyConditionalRule(wf, task);
            if changes continue;

            change = applyBasicLoopRule(wf, task);
            if changes continue;

            change = applyDualLoopRule(wf, task);
            if changes continue;

            change = applyNetworkRule(wf, task);
            if changes continue;

        end forall
    end while

    if workflow_size(wf) > 1 then error("invalid workflow schema")
    else begin
        atomic _task = getAtomicTask(wf);
        return atomic_task.QoS;
    end

end function
```

Listing 1 – The SWR algorithm

To check if a reduction rule can be applied a set of conditions are tested. In Listing 2 we illustrate the *applyConditionalRule* function. From line 3 to line 22, several conditions are tested to ensure that the conditional rule can be applied.

Once this is done, the QoS of the system being reduced is calculated (line 23 and 24) and the workflow is transformed (line 25 and 26). The transformation involves substituting the system being reduced (sequential, parallel, conditional, basic loop, dual loop, or network system) with a new task that has the QoS corresponding to the reduction.

```
1)  boolean applyConditionalRule(workflow wf, task tk) begin

2)  // check if the task tk is a "xor split" and if it is not a network task
3)  if isaXORsplit(tk) and not isaNetwork(tk) begin

4)      // get the tasks involved in the xor-split and xor-join system
5)      task[] next_tasks = wf.getNextTasks(tk);

6)      // check if all the tasks involved in the xor-split and xor-join system only have
7)      // one input and one output
8)      if not hasOneInputOneOutput(next_tks) return false;

9)      // get a task between the xor-split and xor-join task
10)     task a_next_tk = next_tks.getTask();

11)     // get the xor-join task
12)     task xor_join = wf.getNextTask(a_next_tk);

13)     // check if the xor_join task is indeed a "xor join", if the xor_join is not a network
14)     // task, and if the tasks involved in the xor-split and xor-join system are not
15)     // network tasks
16)     if not isaXORjoin(xor_join) or isaNetwork(xor_join) or isaNetwork(next_tks)
17)         return false;

18)     // check if the tasks following the xor-split are connected to the same xor-join
19)     if not sameDstTask(next_tks, xor_join) return false;

20)     // check if the xor-split degree is equal to the xor-join degree
21)     if wf.getNextTasks(tk).size != wf.getPrevTasks(xor_join).size
22)         return false;

23)     // compute the QoS for the conditional system
24)     QoS qos =  computeQoSConditionalSystem(wf, tk);

25)     // change the workflow structure and set the QoS for the new task created
26)     ….

27)     return true;
28) end if
```

```
29) return false;
30) end function
```

Listing 2 – The applyConditionalRule function