Proceedings of the 1995 Winter Simulation Conference
ed. C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman

SIMULATION MODELING WITHIN WORKFLOW TECHNOLOGY

John A. Miller
Amit P. Sheth
Krys J. Kochut
Xuzhong Wang
Arun Murugan

Department of Computer Science/LSDIS Lab
The University of Georgia
Athens, Georgia 30602-7404, U.S.A.

ABSTRACT

This paper presents an approach for integrating simulation modeling and analysis capabilities within the Workflow Management System (WFMS) being developed in the Large Scale Distributed Information Systems (LSDIS) Lab at the University of Georgia. Simulation modeling can be used for studying the efficiency of workflow designs as well as studying the general performance and reliability of WFMSs. We also discuss the importance of using sophisticated monitoring and animation capabilities, and the use of workflow management technology to advance simulation technology itself. Finally, we demonstrate a sample simulation where tasks and task managers are simulated.

1 INTRODUCTION

Competition and economic pressures force modern business corporations to look for new information technologies to support their business process management. Since workflow technology provides a model for business processes, and "a foundation on which to build solutions supporting the execution and management of business processes" (Hsu and Kleissner 1995) it has been receiving much attention in the past few years. A workflow is simply a set of tasks that cooperate to implement a business process. Workflow also provides a way to integrate legacy systems and make good use of past investments in an enterprise in a way that matches the demands of today's rapidly evolving and unpredictably fluctuating enterprises.

In this paper, we focus on the use of simulation modeling and analysis within workflow technology. A workflow model can be used to design automated or semi-automated solutions for certain business processes within an enterprise. Workflow models tend to be more computer-oriented than traditional business process models. Consequently, they better facilitate automatic generation of substantial portions of actual solutions (i.e., executable workflows). Just as the ability to produce simulations from business process models has been found to be useful, we believe that this ability is more important for workflow models. In particular, if a spectrum of simulation approaches are provided, one could perform simulations analogous to those done using business process models today as well as perform a variety of ever more detailed simulations leading all the way up to an actual workflow implementation. Once a workflow is in place, simulation will remain useful for reengineering the workflow and for exploring what-if questions. In addition, the associated monitoring/animation facilities can assist in debugging and tuning workflows as well as understanding and administering workflows. This paper states the case and illustrates through simple examples how this could be done.

The history of workflow technology dates back to office automation and batch processing in the late 70's (Kappel, Schott, and Retschitzegger 1995). In recent years, workflow technology has gained in popularity due to the trend of business process reengineering and many emerging related technologies such as middleware and object-oriented technology, which make the development of a realistic workflow management system possible. An overview of workflow management system is provided in (Georgakopoulos, Hornick, and Sheth 1995) (see Sheth's tutorial on http://optimus.cs.uga.edu/). After several years of development, many workflow products are now available (e.g., FlowMark/IBM, ObjectFlow/DEC, Staffware/Staffware Corp., FloWare/Recognition Int., Action Workflow/Action Tech., and MultiTrac/Post Industrial Computing Systems.

A workflow is composed of multiple *tasks*. There are two types of tasks – simple tasks which represent individual indivisible activities, and compound tasks which represent some activities which can be divided into sub-activities (simple tasks or even other compound tasks). An entire workflow can be regarded as a large compound task. A simple task may be a program which can run on *processing entities*, which include application systems, servers supported by client-server systems or Transaction Processing Monitors (TP-Monitors), DataBase Management Systems (DBMSs), etc. Tasks are operations or a sequence of operations that are submitted for execution at the processing entities using their interfaces. A Workflow Management System (WFMS) provides "the ability to specify, execute, report on, and dynamically control workflows involving multiple humans and HAD (Heterogeneous, Autonomous, and Distributed) systems" (Krishnakumar and Sheth 1995). For workflow execution, a workflow scheduler is necessary to enforce inter-task dependencies, and therefore, to coordinate the execution of tasks in the workflow. Also, *task managers* are designed to start tasks and to perform a supervisory role in forward recovery.

To build a workflow management system that supports the integration and interoperability of heterogeneous, autonomous, and distributed systems, utilization a communication mechanism operating at a higher-level than Sockets or Remote Procedure Calls (RPC) would be beneficial. Distributed Object Management (DOM) is intended to support this kind of integration and interoperability. OMG (Object Management Group)'s CORBA (Common Object Request Broker Architecture) (Object Management Group 1993) is a rapidly maturing standard for DOM. The CORBA specification defines the architecture of an Object Request Broker (ORB), whose job is to enable and regulate interoperability between objects and applications. The CORBA 1.0 specification was released in October 1991. It was followed by CORBA 1.1 released in March 1992 and CORBA 1.2 in December 1993. CORBA 2.0 was announced in the end of 1994 (Object Management Group 1993) (Betz 1995). There are already almost a dozen commercial ORBs or CORBA-like products available in the market (e.g., DOE/Sun Microsystems, ORBeline/PostModern Computing Technologies, Orbix/IONA Technologies, ObjectBroker/DEC, (D)SOM/IBM, HyperDesk/HyperDesk Corp., ORBplus/HP, and XShell/Expertsoft Corp.

In Section 2, we address the interplay between simulation and workflow technology. An overview of the architecture of our two prototype workflow management systems is given in Section 3, while task structures and task models are presented in Section 4. Section 5 details several different types of workflow simulations, and includes a discussion of related monitoring issues. Finally, an example simulation study comparing the two architectures is given in Section 6.

2 SIMULATION AND WORKFLOW

We now consider the interplay between simulation and workflow technology. First, workflow technology and concepts may be used in the development of simulation environments. Second, simulation modeling and analysis capabilities may be integrated with workflow technology.

Workflow technology can benefit simulation in a very important way. A recent trend in simulation environments is to make the components more independent (Standridge and Centeno 1994). The environment would consist of modular loosely-coupled components that are brought together for the purpose of dealing with some simulation analysis problem. Components may include GUI designers, simulation engines, animation packages, graphics packages, spreadsheets, editors, database management systems, forms packages, query tools, mathematical packages, and statistical packages. Instead of a single vendor providing a fixed monolithic environment, a workflow system would allow components to be selected and plugged into the workflow system as needed, and replaced when desired. Furthermore, if simulation vendors could agree on standard interfaces, users would be free to mix and match from multiple vendors. At the 1994 Winter Simulation Conference there was vigorous debate about the need for a complete IEEE sponsored standard for simulation environments. Vendors felt that this might limit innovation and handicap their ability to make autonomous decisions. The workflow approach would only require that the vendors agree on the form of narrow interfaces between different types of components. In addition, this standardization effort could be minimized by incorporating some of the standardization work already done by the Object Management Group (OMG) on CORBA (Object Management Group 1993) and by the Workflow Management Coalition (WfMC) (The Workflow Management Coalition 1994).

The main focus of this paper, however, is on how simulation can be useful for workflow. There are two principal ways in which simulation can be used in a workflow system: (1) Simulation can be used to design WFMS architectures and tune WFMS implementations. The performance and reliability of implementations based on different architectures can be tested. Later in this paper, we give a simple example of a performance study that is used for just this purpose. It compares the efficiency of two architectures, for which we have prototype implementations, under varying workloads and assumptions about the relative amount of work performed by tasks versus task managers. (2) Simulation can be used to study and refine workflow specifications. Because the workflow specification captures the implementation aspects of a business process model, their simulation and analysis can provide valuable feedback to the business process model evaluation.

Using the Graphical Workflow Designer (Murugan 1995), a workflow can be readily designed by a domain expert. For example, the HIIT project *http://www.scra.org/hiit.html* of which we are a part is charged with the task of automating portions of the healthcare delivery system. One important aspect of this is managed care. If patients can receive all of their care services in an efficient sequence, and resources (e.g., doctors, nurses, technicians, operating rooms, hospital rooms, lab tests, etc.) can be assigned so as to minimize queuing delays, then it is possible for patients to receive quality care in less time.

Because of the multitude of factors involved in designing efficient workflows for healthcare delivery and managed care, simulation becomes very useful. Let us consider the following example in which a patient comes to the Emergency Room (ER) of a hospital with a specific complaint. The patient first registers with the receptionist, who then assigns the patient to an examining room in the emergency wing. Once an ER doctor becomes available, he/she examines the patient and comes up with an initial diagnosis. The initial diagnosis may call for lab tests (X-Rays or a biopsy) which will then be analyzed, or it may simply lead directly to a final diagnosis. The doctor at this time determines if the patient should be treated on an inpatient or outpatient basis. After inpatient treatment, the patient may continue treatment as an outpatient or terminate treatment. If the outpatient's progress is not satisfactory, then the patient may return to the hospital for another round of diagnosis; otherwise, they may terminate treatment. Specific treatment plans (both inpatient and outpatient) would in practice expand into their own subworkflows. Figures 1 and 2 are workflow models produced with our Graphical Workflow Designer depicting this example (models were adapted from Hsu and Kleissner 1995).

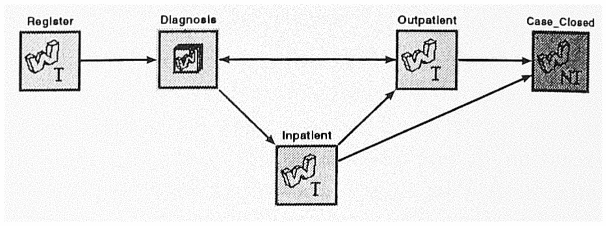

Figure 1: Patient Workflow Model

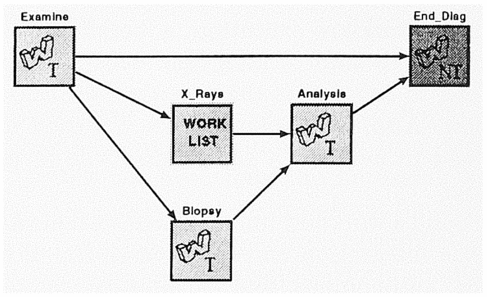

Figure 2: Diagnosis Subworkflow Model

3 WFMS ARCHITECTURES

A Workflow Management System (WFMS) consists of a model repository from which workflows may be selected for execution. Workflows or components of workflows may be added to the model repository by specifying them in a workflow language (e.g., WFSL/TSL, Krishnakumar and Sheth 1995) or by designing them with a Graphical Workflow Designer (Murugan 1995). The Workflow Specification Language (WFSL) (Krishnakumar and Sheth 1995) is a declarative rule-based language to describe the conceptual workflow specification, while the Task Specification Language (TSL) (Krishnakumar and Sheth 1995) is a language to specify simple tasks that run in a HAD information systems environment. Once a workflow is selected from the repository, several translation steps are carried out that instantiate a workflow instance that is able to run within the execution environment (some manual coding/recoding is also necessary). In Subsections 3.1 and 3.2, we give an overview of two architectures for the execution environment.

For the purposes of simulation, a simple flexible architecture is the most suitable. In addition, we wish to be able to conveniently implement full function monitoring and animation. It is therefore useful to have all the relevant information centrally located. This is precisely what our first two prototype WFMS architectures do. We will briefly discuss them here. These architectures and three other architectures are discussed in more detail in Wang (1995).

The main components in the execution environment are the Workflow Scheduler, Task Managers (TMs), and Tasks. Tasks are the run-time instances of an enterprise's applications. Today they typically run independently or are tied together in ad hoc ways. WFMSs tie these tasks together in a loosely coupled fashion. This is achieved by making minor modifications to existing applications code or enforcing standards for new application development. The modifications provide hooks into the task that allow the transitions between major steps to be observed and in some cases controlled by the task manager for the task. To establish global control as well as facilitate recovery and monitoring, the task managers communicate with a scheduler. It is possible for the scheduler to be either centralized or distributed, or even some hybrid between the two (Wang 1995).

3.1 Highly Centralized Architecture

This architecture incorporates task managers into the scheduler's process. This process is multithreaded and has a thread for the scheduler proper, a thread for the scheduler's dispatcher, and a thread for each task manager. Task managers communicate with tasks through a CORBA IDL interface. The architecture is shown in Figure 3, where each box represents a process, while subdivisions within a box represent threads (light weight processes).

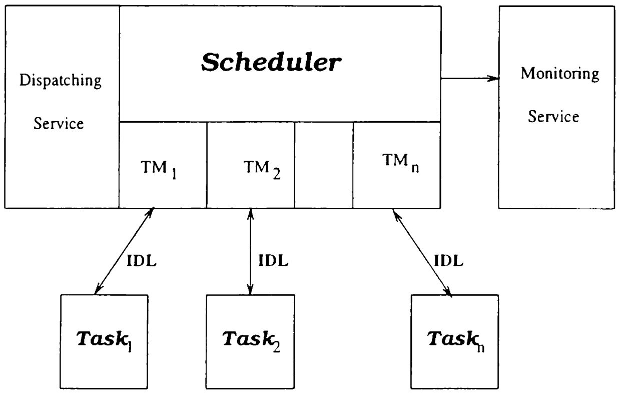

Figure 3: Highly Centralized Scheduler

3.2 Centralized Architecture

The main difference between this architecture and the previous one is that task managers are not threads any more and may reside at remote sites. However, the scheduler still has a thread for each task manager. The thread does nothing other than activate the task manager on a specified machine using another CORBA IDL interface. (The reason for keeping a thread for every task is that synchronous calls are still used to communicate between the scheduler and task managers in this architecture.)

In this architecture, CORBA IDL interfaces are used at two distinct levels: (1) the scheduler process contains threads which communicate with task managers using CORBA IDL interfaces; and (2) task managers communicate with tasks using CORBA IDL interfaces.

Since task managers have been separated from the scheduler process and may reside at other nodes, task managers can take advantage of multiple nodes to do work in parallel. Communication between task managers and tasks may be sped up since a task and its task manager may be co-located in the same process using ORBeline services.

4 TASK STRUCTURES & MODELS

A task structure indicates the generic form of a task. A given structure simply fixes its set of states and the permissible transitions between those states. A full task specification will fix the type of the task. Beyond the task structure, this also requires a specification of allowable operations. Following the object-oriented paradigm, each state will correspond to an operation (or method). A full specification requires that the parameters and return type of each operation be specified. In essence, this is analogous to a class specification in C++. However, to facilitate interoperability and distribution, the specification is in the form of a CORBA IDL interface specification. In addition to operations (methods), attributes and exceptions may be specified. Several different task structures have been developed (Attie et al. 1993, Krishnakumar and Sheth 1995, Wang 1995). The task structure for transactional tasks is shown in Figure 4 below.

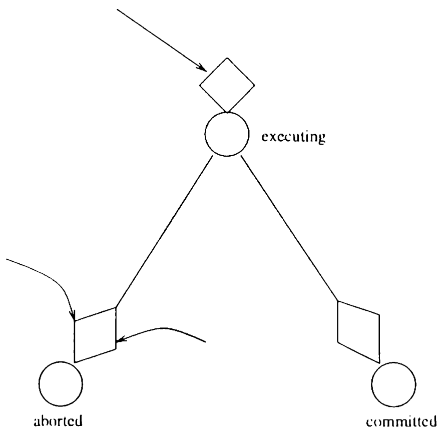

Figure 4: Transactional Task Structure

Clients may call IDL interfaces using any language for which a binding is provided (e.g., C, C++, SmallTalk). Servers (i.e., implementations) have this same flexibility. For our WFMS system, the base IDL interface Task is implemented on the server side as an abstract class (all member functions (methods) are pure virtual). For simplicity only CORBA in and out parameters are allowed. All in parameters (the inputs) are grouped together into a single structure derived from WFL_Data (similarly for the out parameters (outputs)). These structures may contain embedded Object References so that actual objects are not sent over the network, rather references to CORBA objects are passed.

In our system, diagrammatic workflow models (Krishnakumar and Sheth 1995), (Murugan 1995) are inherently hierarchical. At the top level is a model for the overall workflow (see Figure 1), below this are subworkflows (or compound tasks) (see Figure 2) which themselves may be made up of subworkflows or simple tasks. Each task is further modeled via a state transition diagram (see Figure 4). Optionally, the gates (see Section 4.1) guarding states may displayed. Time elapses either when a task is in a state or blocked in a gate waiting to enter a state. Thus tasks are similar to processes in the process-oriented simulation world-view while states correspond to activities, the beginning and end of which are events (instantaneous occurrences). Note, from an aggregate point of view an activity could correspond to an entire task. These graphical workflow and task models along with supplementary specifications (e.g., conditions, inputs, outputs, etc.) both created using the Graphical Workflow Designer are the basis for partial automated code generation for actual workflows as well as simulations. The graphical (or diagrammatic) models used for our workflow system are not unlike those used for discrete-event simulation (Ceric and Paul 1992) (e.g., Event Graphs (Schruben 1983), Petri Nets (Peterson 1977), Activity Cycle Diagrams (Pidd 1992), Activity Diagrams (Birtwistle 1979), GPSS Block Diagrams (Schriber 1974), or SLAM Network Diagrams (Pritsker 1979)). Of these, our diagrams are most similar to Activity Diagrams.

4.1 Intertask Dependencies and Enable Arcs

Coordination between tasks is accomplished by specifying intertask dependencies using `enable` clauses. An `enable` clause may have any number of predicates. Each predicate is either a task-state vector or a Boolean expression. For the database community, this represents an Event-Condition-Action (ECA) rule (Chakravarthy et al 1989).

A successful enable will cause one leaf node in the gate guarding say [task-i, state-j]) to become true. If the gate's OR-AND tree now evaluates to true, the gate will open (it is fully enabled) and the method implementing [task-i, state-j] will be scheduled for execution. If state-j is the root of the DAG representing task-i's task structure, then enabling causes task initiation. The execution may be scheduled to occur immediately (this is the default) or at a given time in the future, or after a given time delay. In the case of traditional simulation using virtual time, such scheduling is straightforward. However, for scheduling using real time, one could potentially address many complex issues. Our WFMS is at present not intended for use in domains where hard real time constraints are present; we simply incorporate deadlines into the calculation of dynamic priorities. The dynamic priority will affect placement on WFMS dispatch queues, and optionally the actual execution priority for operating systems that allow such influence.

The time dimension will correspond to real time when actual implementations are performed. The same is true when components are replaced with simulated versions. However, in the case of traditional discrete-event simulation virtual time will be used.

5 TYPES OF SIMULATIONS

For the high level workflow model shown in Figures 1 and 2, simulation would be primarily useful for allocating resources (doctors, nurses, lab technicians, rooms, equipment, etc.) to meet expected demands. Major opportunities to apply the principles of managed care would occur when detailed treatment plans are redesigned.

As illustrated in the example in Section 2, it is important to be able to run numerous simulations before full implementation of a workflow is attempted. Furthermore, it is useful to provide a variety of different simulation modes from abstract to near implementation. The degrees of freedom in this spectrum correspond to replacing actual components with simulated components. These simulated components should try to mimic the actual behavior of real components as closely as possible in terms of resource contention and consumption, and outputs generated. These aspects would be simulated by stochastic means. Depending on the characteristics of the task involved different resources may be deemed important. Examples of potentially important resources include CPUs, memories, disks, networks, files, databases, repositories, and warehouses.

At the near implementation end of the spectrum, one may envision simply replacing (or more likely deferring the coding of) a few of the larger more complex tasks with simulated versions. Moving a little further from the implementation end, one could replace all the task with simulated versions. This would be particularly useful when all of the tasks have yet to be coded or may need to be substantially recoded.

Simulating tasks can be very useful since tasks which represent the application code can be thousands of lines of code. It would be advantageous to be able to test aspects of a proposed workflow before implementing or retrofitting an application. It also allows what-if questions to be more easily addressed.

The next step is to replace some or all of the task managers with simulated versions. Task managers and tasks work together to achieve two goals, namely, to carry out some application and to coordinate with the rest of the workflow. The task is mainly composed of application-oriented code, while the task manager is composed of code to observe/control the execution of task(s), deal with failure and recovery issues, and communicate with the scheduler or other task managers. Sometimes the distinction between the two roles may be blurred. For example, it might be the case that a task is written to provide interactive SQL access to a database. The task manager might be charged with the responsibility of locating, connecting to, and opening the appropriate database. Because of the important interplay between task managers and tasks, one will often want to simulate both as was done for our sample simulation (see Section 6). With this type of simulation, one is still running a distributed program.

One could further simplify the situation by entirely eliminating the tasks. Care must be taken in doing this, so as to not lose realism. For example, what happens to the communication delays, the delays associated with tasks carrying out their operations, and the possible contention for resources that is missing because tasks are no longer running. These aspects can be replaced by introducing queues, simulated resources, and operational delays. If the WFMS is implemented according to the Highly Centralized Architecture, then this is particularly convenient for simulation purposes, as all of the components will be threads executing within a single (heavyweight) process. This is similar to what happens within traditional discrete-event simulators which follow the process-oriented world view (Kreutzer 1986). Such simulators use either threads or coroutines to represent the active elements within the simulation.

There is still an important limitation to the modes of simulation so far discussed. The simulated components are to mimic the actual components as closely as possible in time and space. This is good from the point of view of model validation as well as getting feedback from potential users/participants of/in the workflow (e.g., sample screens may pop up on a nurse's workstation). Unfortunately, this will make it difficult to explore a variety of what-if questions for lengthy workflows. To deal with the above problem, some form of time compression in the simulation needs to be provided. So far the scheduler has been operating in real time. Traditional discrete-event simulators operate on simulated or virtual time. Rather than time smoothly advancing it jumps from event to event skipping the time in between. This allows for substantial time compression without losing model fidelity (e.g., the ability to detect bottlenecks accurately). To facilitate this type of simulation modeling, the scheduler's internal data structures maintain either real time or virtual time information. The choice is made by a compile time switch. Each choice has its advantages: Traditional discrete-event simulation has the advantage of time compression, while replacing components with simulated versions has the advantage that the simulation becomes an intermediate step to implementation.

The ability to monitor the execution of workflows, actual or simulated, is extremely important. Several types of monitoring/tracking facilities should be provided. In our current prototype WFMS, two monitoring facilities are provided.

The first monitor developed was simply a process that acts as a CORBA server to which the scheduler process sends messages. The scheduler sends a message to the monitor whenever an event occurs. Upon receiving a message, the monitor formats it appropriately and displays it in a window. Unfortunately, watching event messages scroll in a window is not a very exciting way to understand what is happening in a workflow. Still, this type of detailed textual information may be very useful for debugging. An event filter is being designed to allow only certain events to be displayed to improve the situation.

The second monitor is more interactive. It allows a user to specify a task name (or task id if he/she wishes) to discover status information about a task (e.g., what state it is in, values of inputs or outputs if available, when it started, whether any exceptions have been raised, etc.).

The next monitor which has been designed, but yet to be developed, will provide a GUI interface via the Graphical Workflow Designer to the second monitor. Using this monitor, a user may select a task icon using the mouse and click on it to see status information (e.g., the current state is shown by changing its color in the task structure diagram). The final monitor we plan to develop will be able to show animations of the workflow design diagram.

6 EXAMPLE SIMULATION STUDY

We close this paper by showing the results of a preliminary study used to compare architectures 1 and 2. It was our first cut at replacing components of workflows with their simulated versions. Task managers were partially implemented, while tasks themselves were entirely simulated. The only resource that was consumed by simulation replacement was CPU time. In that sense, the study is limited. Task managers were real in the sense that they communicated with tasks using a real API (CORBA IDL) interface. They, however, did not do any of the things that a full-blown task manager might do (e.g., connect and open a databases, check the format and possibly content of messages, handle failures, manage retries, etc.). To cover a wide range of possibilities, we simulated the consumption of CPU time (ranging from a few milliseconds to a couple of seconds for each major step taken by the task manager). Most commonly, the CPU requirements of task managers would be small. Tasks on the other hand were entirely simulated. The range of time taken by a task can vary widely from a few milliseconds to minutes or even hours. Due to this wide variation in time requirements we made the upper bound the same for tasks and task managers (naturally, a follow-on study should investigate what happens when this upper bound is increased). The environment on which the simulation was performed was an Ethernet (10BASE-T) LAN consisting of one Sun SPARCstation 20 and two Sun SPARCstation 5's all running Solaris 2.4. Communication was provided by PostModern Computing's CORBA 1.2 implementation, Orbeline.

In this study, task CPU requirements, task manager CPU requirements, and the number of tasks were varied over the study domain shown in Figure 5. Lines A, C and E test the effect of varying task manager CPU requirements on turnaround time when the tasks have light, moderate and heavy CPU requirements, respectively. Line B goes from heavy task managers and light tasks to light task managers and heavy tasks, while Line D goes from light task managers and light tasks to heavy task managers and heavy tasks. At every point on these lines, the number of tasks varied from 4 to 160 (by 4). Some tasks can be executed in parallel, and others cannot. Tasks and task managers were distributed over the SPARCstation 5 workstations as evenly as possible. The scheduler was run on the SPARCstation 20.

Figure 5 illustrates the relative performance of the two architectures over the entire domain of the simulation study. Architecture 2 is more than 10 percent faster than architecture 1 in Area a, while architecture 1 is more than 10 percent faster than architecture 2 in Area d. Our results indicate that architecture 2 is faster than architecture 1 when tasks are less CPU intensive than task managers. In this case, task managers compete for CPU time with the scheduler in architecture 1. Conversely, the results show that architecture 1 is faster than architecture 2 when tasks are highly CPU intensive. An explanation is that the CPU contention between the scheduler and task managers diminishes because they end up waiting for slow running tasks.

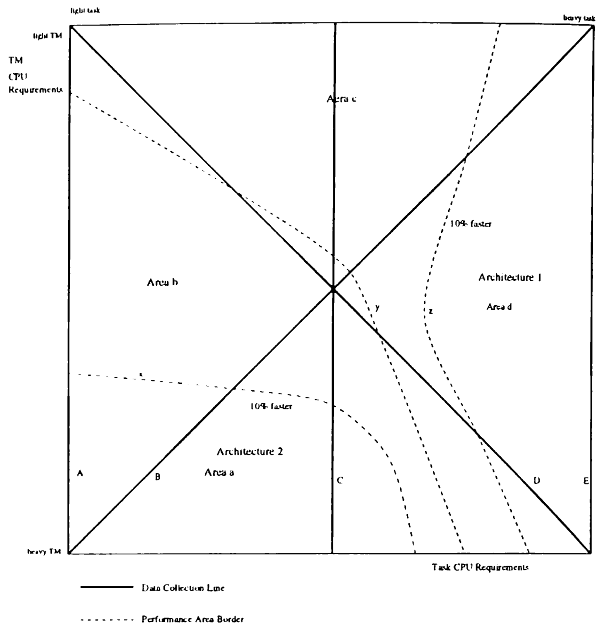

Figure 5: Throughput Contour Map

7 CONCLUSIONS

In this paper, we discussed how simulation modeling and analysis can be utilized for designing both workflows and workflow management systems. We argued that the best approach is to provide a spectrum of solutions, from high-level traditional discrete-event simulations to simulations that are near implementations. The importance of sophisticated monitoring and animation capabilities was also highlighted. An example simulation study was performed to compare the two WFMS architectures that we have developed, the highly centralized (architecture 1) and the centralized (architecture 2) architectures. Even though this was only a first cut simulation study, it did yield some interesting results. The results suggest that architecture 1 is superior when tasks have heavier CPU requirements, while architecture 2 is superior when CPU requirements for task managers are substantially greater than the CPU requirements for tasks.

REFERENCES

Attie, P., M. Singh, A. Sheth, and M. Rusinkiewicz. 1993. Specifying and enforcing intertask dependencies. In *Proc. of the 19th Intl. Conf. on Very Large Databases* 134–145. Dublin, Ireland.

Betz, M. 1995. OMG's CORBA. *Dr. Dobb's Journal* 9(16):8–13.

Birtwistle, G. M. 1979. *Discrete Event Modelling in SIMULA*. MacMillian, London.