AN EVALUATION OF METHODOLOGICAL ISSUES IN WORKFLOW MANAGEMENT

A THESIS

SUBMITTED TO THE DEPARTMENT OF

COMPUTER ENGINEERING AND INFORMATION SCIENCE

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Anastasia Soinikova

August, 1998

AN EVALUATION OF METHODOLOGICAL ISSUES IN WORKFLOW MANAGEMENT

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER
ENGINEERING AND INFORMATION SCIENCE
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Anastasia Sotnikova
August, 1998

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Özgür Ulusoy(Principal Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Cevdet Aykanat

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Uğur Güdükbay

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet Baray, Director of Institute of Engineering and Science

ABSTRACT

AN EVALUATION OF
METHODOLOGICAL ISSUES IN
WORKFLOW MANAGEMENT

Anastasia Sotnikova
M.S. in Computer Engineering and Information Science
Supervisor: Assoc. Prof. Dr. Özgür Ulusoy
August, 1998

Workflow management is a diverse and rich technology being applied over an increasing number of industries. Despite this fact, workflow management systems (WFMSs) still have a long way to go before they can be regarded as mature technology. In this thesis, we try to analyze methodological aspects of WFMSs and contribute to the workflow management theory in terms of new functionality and structures of workflow schemas. A confirmation of our ideas is provided by simulation results of a workflow application which we have designed. Bringing the simulation stage in between design and implementation stages would let a schema designer assess a workflow system in terms of optimal system throughput, required facility capacities, and an efficient transactional representation of activities. Also, by allowing a schema designer to choose an effective structure of a workflow system, simulation results help to avoid possible future losses at the early stages of the workflow schema design.

Key words: Workflow Systems, Advanced Transaction Models, Performance Evaluation.

ÖZET

İŞAKIŞI YÖNETİMİNDEKİ METODOLOJİK KONULARIN BİR DEĞERLENDİRMESİ

Anastasia Sotnikova
Bilgisayar ve Enformatik Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Doç. Dr. Özgür Ulusoy
Ağustos, 1998

İşakışı yönetimi teknolojisi, endüstride pek çok alanda uygulanma imkanı bulmasına rağmen, işakışı yönetim sistemlerinin henüz belirgin bir olgunluğa eriştiği söylenemez. Bu tezde hedeflerimiz, işakışı yönetim sistemlerinin metodolojik özelliklerini analiz etmek ve işakışı yönetim teorisine fonksiyonel ve yapısal açıdan katkılarda bulunmak olmuştur. Önerdiğimiz fikirlerin doğrulanması amacıyla bir işakışı uygulanması tasarımı yapılmış ve simülasyon tekniği kullanılarak tasarlanan uygulama test edilmiştir. Tasarım ve geliştirme adımları arasında simülasyon tekniğinin kullanılması, tasarımcıya işakışı sisteminin çeşitli yönlerden değerlendirilebilmesi imkanını verecektir. Ayrıca, simülasyon sonuçları tasarımcının işakışı sistemi için en uygun yapıyı seçmesini sağlayarak, ileriki aşamalarda olması muhtemel kayıpların önlenmesi için de yardımcı olacaktır.

Anahtar sözcükler: İşakışı Sistemleri, Gelişmiş İşlem Modelleri, Performans Ölçümü.

To my grandparents

ACKNOWLEDGMENTS

First of all, I would like to express deep gratitude to my advisor Assoc. Prof. Dr. Özgür Ulusoy for his careful reading and timely constructive comments during the study. I would also like to thank my colleague Oleg Gusak for his friendship and technical support.

I would like to thank the committee members Assoc. Prof. Dr. Cevdet Aykanat and Asst. Prof. Dr. Uğur Güdükbay for their valuable comments, and everybody who has in some way contributed to this study by lending moral and technical support.

Finally, I am very thankful to my family especially to my grandparents who had grown me up in a scientific atmosphere. Owing to their efforts I was able to identify rightly my personal strivings and research interests.

Contents

Chapter 1

INTRODUCTION

Workflow management is a discipline that studies the coordination, communication and control of organizational processes by means of information technology for the purpose of improving these processes [Joo96]. An organizational process contains the set of activities involved in handling the arbitrary number of similar actions, which are typically stretched across boundaries of departments and organizations. A workflow process is an automated organizational process, which means that the coordination, communication and control within a process are performed using information technology, but the activities within the process are either manual, or automated, or a mixture of these two. Workflow management is relatively a new term, however the ideas and concepts associated with it have been around for a considerable period of time. Workflow management can be seen as a logical expansion of a number of different fields. The Workflow Management Coalition (WFMC) [Wor96] suggests no less than eight areas that have had a direct influence on the development on workflow management:

- image processing,
- document management,
- electronic mail and directories,
- groupware,

- transactional systems,
- project support applications,
- business process re-engineering,
- structured system design tools.

As we can see from this list, workflow management challenges questions of interdisciplinary nature. It comprehends many other fields, but mostly information systems (e.g., database systems, data communication, software process modelling, software engineering, programming) and organizational science (e.g., decision theory, administrative organization, and management science). From the field of information systems workflow management borrows and extends the developed applications and techniques to support workflow processes. Organizational science defines interfaces for an organization and workflow technology interaction. It defines necessary workflow management system (WFMS) functionality and an efficient way for conducting a workflow-oriented investigations in an organization. It also points out the impact of different types of organizations on the structure and performance of a planned WFMS. At the intersection of both disciplines there is a methodology. Methodological research addresses definitional issues, in an attempt to help architects of workflow tools and designers of workflow processes by means of methodologies and tools.

Our study is devoted to the methodological issues in workflow management, in particular, to the required functionality of WFMS and the transactional support in workflows.

Current WFMSs do not exploit simulation techniques and do not provide workflow designers with a choice of system implementation as priority or non-priority. We propose a place and a way of incorporating these functionalities into WFMSs. The second drawback that we noticed in most of the WFMSs is that workflows are not based on the transactional concept. Even WFMSs that represent workflows as transactional processes, use insufficient sets of transaction models which cannot provide an appropriate support and flexibility for all possible workflow tasks. In the thesis, we present a set of transaction models that can be used in a workflow implementation and show on an example a

possible employment of these models. We hope that the ideas presented in this thesis will find their place in practical implementation of WFMSs.

The outline of this thesis is as follows. In Chapter 2, we describe the research problem. Chapter 3 provides basic concepts from the fields our study is related to. Also in this chapter we briefly describe our workflow model. In Chapter 4, we survey from the literature the state of the art in the workflow management. Chapter 5 describes our model in detail and presents the deadline assignment strategies we employ in our simulation experiments. Chapter 6 is devoted to the experimental part of our study and presents the results of the simulation. Chapter 7 concludes the thesis.

Chapter 2

Problem Description

2.1 Functionality of a WFMS

In order to give an idea to the reader about a general representation of a WFMS we present the reference model of it provided by the WFMC [Wor96] (Figure 2.1). The model provides the general architectural framework for the work of the WFMC. It identifies interfaces covering, broadly, five areas of functionality between a WFMS and its environment.

- The import and export of process definitions.
- Interaction with client applications and worklist handler software.
- The invocation of software tools or applications.
- Interoperability between different WFMSs.
- Administration and monitoring functions.

The place for enlargement of functionality is in the process definition tools. In addition to the provided functionality, i.e., visual definition of the schema of the workflow processes, it should supply the schema designer with a list of possible transaction models and deadline policies, and a simulation tool, which would analyse a developed schema and give some performance evaluations for

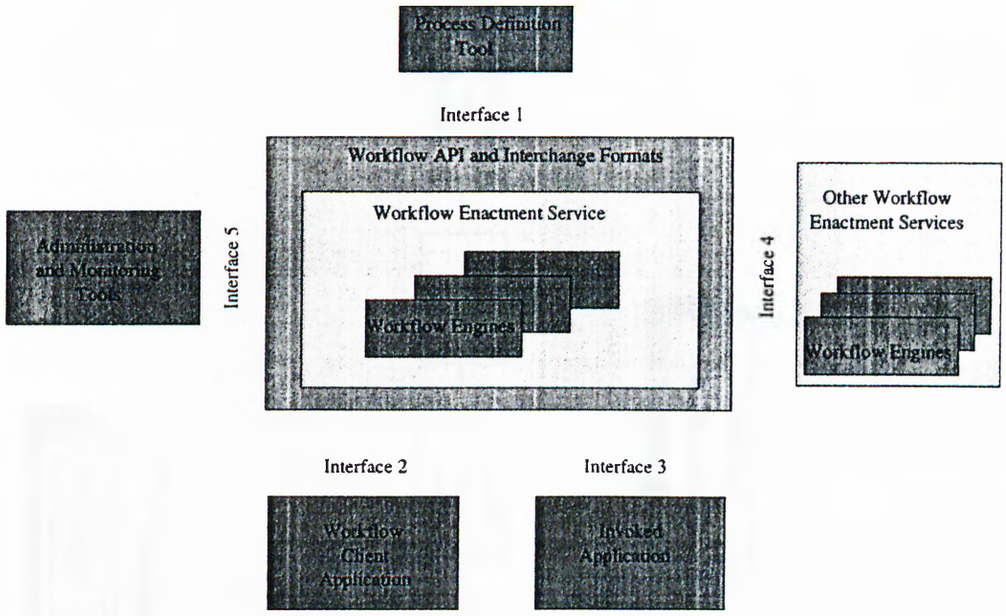

Figure 2.1: The Generic WFMS Schema

it. Thus, allowing the schema designer to choose an effective structure it avoids possible future losses at the early stages of the workflow schema development. Based on our workflow schema in Chapter 5 we describe the possible workflow settings, and in Chapter 6 we apply the set of transaction models and conduct performance analysis.

2.2 Application Area

There has been a growing interest for the use of workflow technology in numerous application domains. In literature, one can find a lot of studies related to different aspects of workflows. Although the workflow-related investigations require real world examples to assess the variety of performance parameters, many of the researches use abstract workflows in their studies. The spectrum of practiced or modeled workflow examples still does not cover the possible application areas and therefore a generalization of WFMSs requirements is not feasible yet. The application scope that is encompassed by the research community follows. The authors of [KS95, PR97, PR98b, PR98a] use a service provisioning process as an example. An example of a loan request workflow is

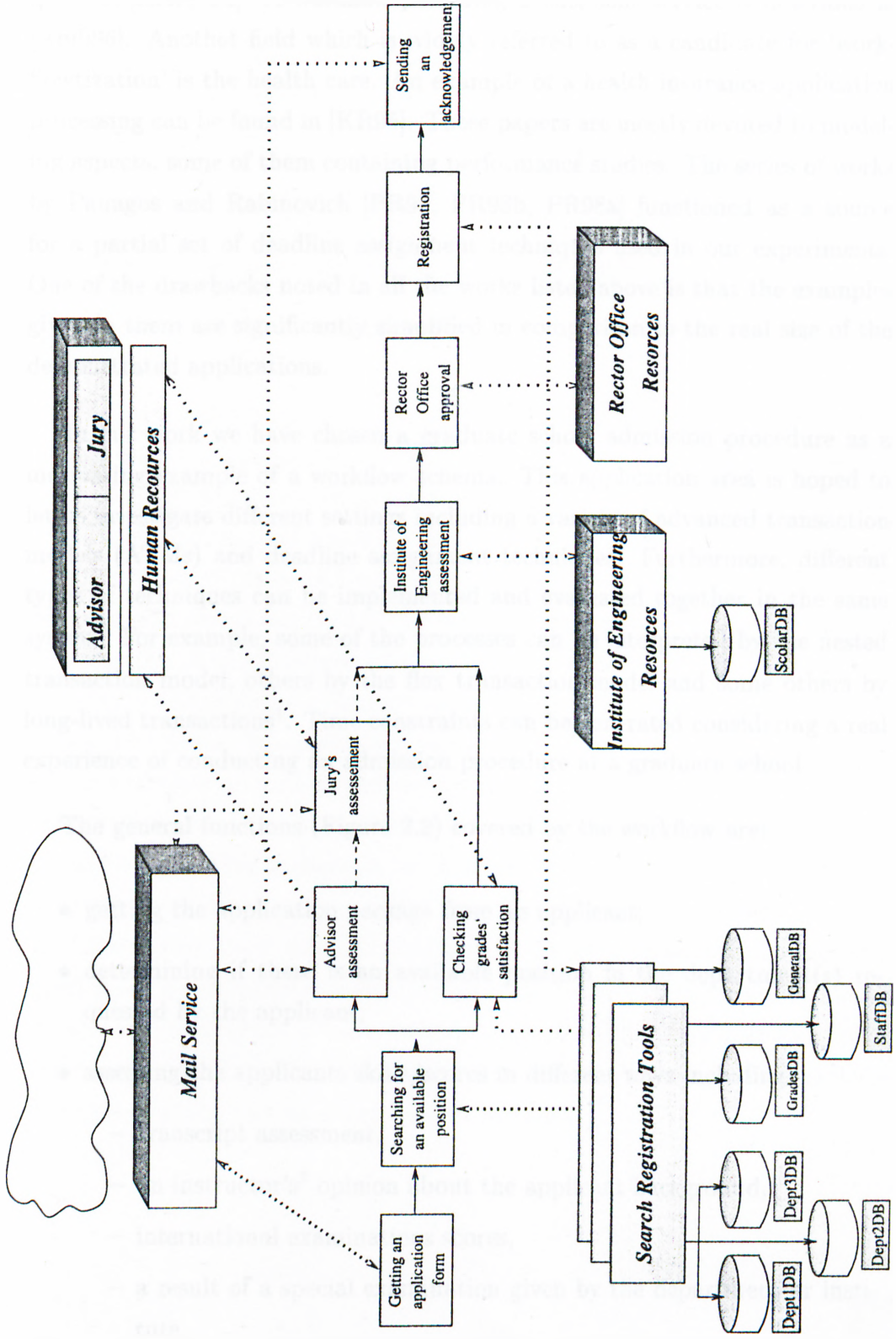

Figure 2.2: Generic workflow schema

given in [AAA+95]. A workflow providing a telephone service is described in [Amb96]. Another field which is widely referred to as a candidate for 'workflowtization' is the health care. An example of a health insurance application processing can be found in [KR95]. These papers are mostly devoted to modeling aspects, some of them containing performance studies. The series of works by Panagos and Rabinovich [PR97, PR98b, PR98a] functioned as a source for a partial set of deadline assignment techniques used in our experiments. One of the drawbacks noted in all the works listed above is that the examples given in them are significantly simplified in comparison to the real size of the demonstrated applications.

In our work we have chosen a graduate school admission procedure as a motivating example of a workflow schema. This application area is hoped to let us investigate different settings including a variety of advanced transaction models (ATMs) and deadline assignment techniques. Furthermore, different types of techniques can be implemented and evaluated together in the same system. For example, some of the processes can be interpreted by the nested transaction model, others by the flex transaction model and some others by long-lived transactions[1]. Time constraints can be generated considering a real experience of conducting an admission procedure at a graduate school.

The general functions (Figure 2.2) covered by the workflow are:

- getting the application package from an applicant;
- determining if there is an available position in the department(s) requested by the applicant;
- assessing the applicants skills/scores in different ways including:
  - transcript assessment,
  - an instructor's[2] opinion about the applicant background,
  - international examinations scores,
  - a result of a special examination given by the department or institute,

[1] Definition of the transaction models are provided in Section 3.2.
[2] who is assumed to be the advisor for the applicant.

  - decision regarding the applicant taken by a jury of faculty members;

- taking an approval from a higher lever administrative office (e.g., Institute of Engineering) with an attempt of fulfilment of financial support, if it was requested;
- final approving (e.g., Rector Office approval);
- registration of the applicant as a graduate student;
- sending an acknowledgement.

Figure 2.2 also presents the resources that are used at each particular step of the workflow application we consider. From now on, all the services and recourses will be referred to as facilities. The system facilities include mailing facility, registration and searching facilities, the department's human facilities, a facility from Institute of Engineering, and a Rector Office facility. The registration and search facility and the facility from Institute of Engineering have the corresponding repositories for processing the activities, which require retrieving or saving data for their execution. GeneralDB is a database that stores application forms. Examination grades like GRE, TOEFL, GMAT scores, entrance examination results, and some additional data are stored in GradesDB. Information about faculty members, their major research topics and available positions for graduate students are maintained in StaffDB which is used for choosing an advisor for an applicant. Dept1DB, Dept2DB, and Dept3DB store the information about available positions in the departments. If an applicant gives a list of desirable departments then the search is performed on all the databases. ScholarDB is used to keep track of scholarships and to provide information about available scholarships for new applicants. A more detailed description of the schema is provided in Section 5.1.

2.3 Multitransactional Support vs. Workflow Transaction Model

One of the issues in workflow management that our work is related to is the transactional aspect of workflows. The workflow concept seems to be in evolutionary development, starting with conventional DBMSs going through real-time, heterogeneous, active, distributed and mobile DBMSs. In addition to the properties and issues imposed by their frames, WFMSs pose the challenges of human invocation and legacy applications management. Without the latter concepts, WFMSs would progress as transactional systems, inheriting the quite well studied transactional management issues such as:

- flexibility
- interoperability
- availability
- concurrency control
- recovery
- scalability

With the need of humans' control, incorporation of already existed home-grown applications, and modeling workflow schemas using inter-activity dependencies and conditional execution, workflows become a superset of the establishments and rules in the database area. The question of whether workflows should be of transactional nature is discussed among workflow researches whereas, the commercial camp, nonwilling to wait for a decision has taken the non-transactional side.

> ... no commercial workflow products are based on the on-line transaction processing or database technology ... [AAAM98]

> ... database community has had so far very little impact in this (workflow management system) area. [Alo98]

While database management is of the *data-centric* nature, workflow management is recognized/confessed as to be of *process-centric* origin. The challenge that the workflow community have been facing for the last few years is how to combine these two approaches to produce a sophisticated environment which would satisfy the wide spectrum of workflow needs.

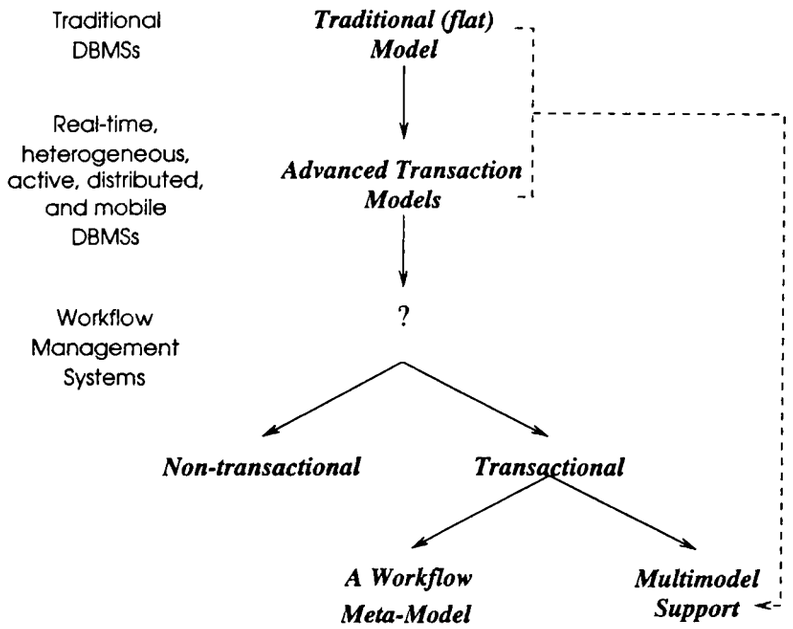

Figure 2.3: Transaction evolution.

Figure 2.3 shows the sequence of possible solutions for the transaction issue in workflow management. Traditional transactions being the first formalization of database accesses are defined as a collection of operations for which a DBMS quarantees certain properties regarding reliability and correctness of computations. On-line transaction processing (OLTP), as way to control transaction execution, is assumed to manage a large number of relatively short-lived transactions. Success of applying the latter concept in traditional DBMSs led to efforts of using the same methods in other application domains which require more flexibility and concurrency. Due to the autonomy property[3] of traditional transactions, this attempt failed already in the early investigations in the workflow area [WAN97].

[3]This aspect is discussed in detail in Section 3.2.

Relaxing some of the traditional transaction properties allows advanced transaction models (ATMs) to be used in the next generation DBMSs. However, these models still carry a significant drawback: a single ATM relaxes a single property (e.g., atomicity, isolation, etc.) whereas activities constituting a workflow may or may not require this particular feature. An appropriate solution could be found by designing a new model for each application or by having a general framework to describe and reason about transactional properties of complex applications.

A sound attempt was undertaken by developing several metamodels which worked for certain set of applications. But the backside of the significant success of workflow acceptance in many areas materialized an immature property of the proposed metamodels. Researches and practitioners have identified new applications that could conceptually use this technology, but current workflow products either do not address the emerging requirements or do so selectively [SK97]. Nevertheless, the valuability of the transactional approach is preserved in current research. Database control is still needed because DBMSs provide services like controlled persistency of data shared among workflow participants.

In our work we introduce the concept of multimodel support in workflow applications. The kernel of this approach is that a WFMS that maintains a set of transaction models (including the traditional model) and dependencies that are used to describe a workflow schema. When defining each activity's execution rules, the workflow designer chooses a model for it from the set supported by the WFMS. The merit of using this approach is that there is no need in inventing and formalizing a new transaction model. The list of ATMs presented in Section 3.2 is recognized [JK97, PKH88, Elm92] as sufficient for describing most of the applications. Moreover, with the progress of the ATMs a multimodel WFMS will become more sophisticated. Existence of several models in a single management system might sound infeasible but due to some of the properties of workflow applications presented in Section 5.2 it becomes quite possible. Each activity can be represented as an autonomous unit monitored/controlled by the chosen policy. The treatment of legacy applications and human invocation in frame of this approach is presented in Section 3.3.

Chapter 3

Background

3.1 Basic Concepts

A *workflow* is a collection of tasks organized to accomplish a business process. A workflow should reflect an organization processing structure, its material and information processes. Enterprises and organizations can introduce a *workflow management system* (WFMS) in their business processes aimed not only for automating documents rotation but also supporting human intervention in managing the business processes, human collaboration and co-decision. Therefore, a real world workflow application is a large-scale system that should combine ad hoc, administrative, collaborative and production management [GHS95]. The workflow community accepted the same classification for workflows: *ad hoc*, *administrative*, *collaborative* and *production* workflows. Although this classification is not very strict, it points out the principal differences of workflow-based applications. *Ad hoc* workflows perform office processes such as, product documentation or sales proposals, where there is not set pattern for moving information between the participants. Thus, the ordering and coordination of tasks in an ad hoc workflow cannot be fully automated and must be controlled by humans. Furthermore, the tasks ordering and coordination decisions are made while workflow is performed. *Administrative* workflows involve repetitive, predicable processes with simple task coordination, such as

patient registration in a health care organization. The ordering and coordination of tasks in administrative workflows can be fully automated. They do not encompass complex information processes and do not require access to multiple information sources used for supporting collaborative management. The name of *collaborative* workflows says for itself; it can be seen as an extension of administrative management where in order to complete a distributed business process, humans' collaboration is required. *Production* workflows are more complex and are the combination of ad hoc, in terms of human intervention and unpredictability in execution patterns, collaboration and administrative workflows. Automation of production workflows is complicated due to information processes' complexity and necessity to access multiple information sources and storages to accomplish the constituting tasks. The following paragraph introduces the basic workflow-related concepts and definitions.

Although it is widely popular to switch to distributed technologies and applications in providing definitions for core aspects and distinctive features of workflows and WFMSs, we keep the centralized way respecting the efforts taken by the *Workflow Management Coalition* (WfMC). WfMC was organized to lead the computer society out of labyrinth of opinions of what workflows are, what they consist of, what we can call a WFMS and what we cannot. Our further discussion will touch the workflows and WFMS definitions. Initially, let us represent the interconnection between basic concepts in the workflow area in a graphical form to prevent any misunderstandings in this still not well-structured field of Computer Science (Figure 3.1).

Definition of the terms used in Figure 3.1 and the others which we use in our study are adopted from WfMC Terminology and Glossary [Wor96]. Keeping in mind discrepancies in workflow terminology, we provide the exact definitions given by WfMC. Nevertheless, we will modify or extend some of them in further chapters.

- *Business Process* - a set of one or more linked procedures or activities which collectively realise a business objective or policy goal, normally within the context of an organisational structure defining functional roles and relationships.

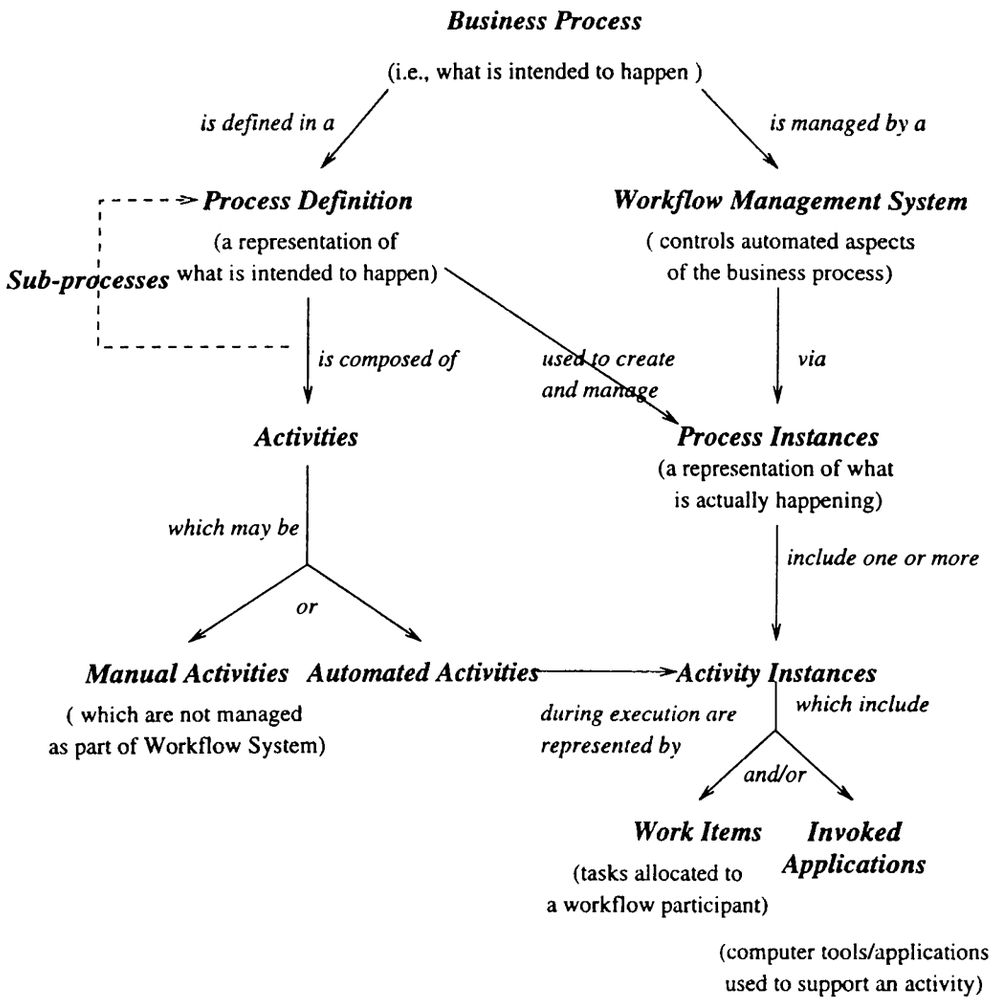

Figure 3.1: Relationship between basic terminology

- *Process Definition* - the representation of a business process in a form, which supports automated manipulation, such as modelling, or enactment by a workflow management system. The process definition consists of a network of activities and their relationships, criteria to indicate the start and termination of the process, and information about the individual activities, such as participants, associated *information technologies* (IT) applications and data, etc.

- *Workflow Management System* - a system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications.

- *Workflow* - the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.

- *Process* - a formalised view of a business process, represented as a coordinated (parallel and/or serial) set of process activities that are connected in order to achieve a common goal.

- *Activity* - a description of a piece of work that forms one logical step within a process. An activity may be a manual activity, which does not support computer automation, or a workflow (automated) activity. A workflow activity requires human and/or machine resource(s) to support process execution; where human resource is required; an activity is allocated to a workflow participant.

- *Automated Activity* - an activity that is capable of computer automation using a workflow management system to manage the activity during execution of the business process of which it forms a part.

- *Manual Activity* - an activity within a business process which is not capable of automation and hence lies outside the scope of a workflow management system. Such activities may be included within a process definition, for example to support modelling of the process, but do not form part of a resulting workflow.

- *Instance* (as in Process or Activity Instance) - the representation of a single enactment of a process, or activity within a process, including its associated data. Each instance represents a separate thread of execution of the process or activity, which may be controlled independently and will have its own internal state and externally visible identity, which may be used as a handle, for example, to record or retrieve audit data relating to the individual enactment.

- *Work Item* - the representation of the work to be processed (by a workflow participant) in the context of an activity within a process instance. An activity typically generates one or more work items, which together constitute the task to be undertaken by the user (a workflow participant) within this activity. (In certain cases an activity may be completely handled by an invoked application which can operate without a workflow participant, in which case there may be no work item assignment.)

- *Workflow Participant* - a resource that performs the work represented by a workflow activity instance. This work is normally manifested as one or more work items assigned to the workflow participant via the worklist.

- *Worklist* - a list of work items associated with a given workflow participant (or in some cases with a group of workflow participants who may share a common worklist).

- *Deadline* - a time based scheduling constraint, which requires that a certain activity (or work item) be completed by a certain time (the 'deadline').

- *Escalation* - a procedure (automated or manual) which is invoked if a particular constraint (such as the deadline) or condition is not met.

- *Parallel Routing* - a segment of a process instance under enactment by a workflow management system, where two or more activity instances are executing in parallel within the workflow, giving rise to multiple threads of control.

- *Sequential Routing* - a segment of a process instance under enactment by a workflow management system, in which several activities are executed in

sequence under a single thread of execution. (No -split or -join conditions occur during sequential routing.)

- *AND-Split Point* - a point within the workflow where a single thread of control splits into two or more parallel activities.
- *AND-Join* - a point in the workflow where two or more parallel executing activities converge into a single common thread of control.
- *OR-Split* - a point within the workflow where a single thread of control makes a decision upon which branch to take when encountered with multiple alternative workflow branches.
- *OR-Join* - a point within the workflow where two or more alternative activity(s) workflow branches re-converge to a single common activity as the next step within the workflow.

From the previous experience of commercial WFMSs it has become clear that WFMSs should have underneath a database management system (DBMS) and should rely on it as a management technology, not just as a repository. While much research has been done in the area of advanced transaction models (ATMs) in DBMSs, non of the current WFMS products support the transaction concept [Moh97, AAAM98]. The reason lies in the fact that neither the traditional transaction model nor single ATM satisfies the wide spectrum of workflow management demands. To clarify the above statement let us highlight the limitations in the traditional transaction technology and present partial solutions for them provided by ATMs.

3.2 Transactional Support

The transaction model was originally designed for business oriented database applications, where transactions are generally short and atomicity of transactions is strictly necessary. A traditional or flat transaction must obey *atomicity*, *consistency*, *isolation* and *durability* (ACID) properties. The *atomicity*

property requires either all the effects of operations of a transaction to be successfully installed in the database, or non of them. Thus, the transaction is an indivisible, *atomic*, unit of work. If a transaction leaves the database in a consistent state, providing that it was consistent when the transaction started, then it satisfies the *consistency* property. The *isolation* property guarantees that concurrent execution of transactions does not introduce inconsistency to the database. For a single transaction, even if it is executed concurrently with other transactions, the database view is that this transaction is executed alone. To fulfil the *durability* requirement, all the effects of committed transactions must be permanent for a database, and guaranteed to survive any subsequent failures. With the recent use of databases for managing distributed, mobile, heterogeneous environments, transactions have been becoming an order of magnitude more complex. In such environments, transactions need to access many data items and reside in the system for a long period of time. Transactions of this kind are usually called *long-lived* transactions[1]. Long-lived transactions pose new challenges to the traditional transaction technology. A long-lived transaction is more easily interrupted by failures because of its long execution time. Because of atomicity requirement, when a failure occurs, it has to be rolled back and all the effects on the database must be undone. This might be reasonable for short transactions, which is composed of one or a few database operations; however, this is not acceptable for long-lived transactions due to the fact that much work might have been done and will be lost if the transaction aborts. Moreover, not all of the committed operations might be affected by the abortion. Isolation requirement causes unnecessary idle times (downtimes) in database applications which process long-lived transactions. A long-lived transaction access many data items and these data items, in frame of the isolation requirement, cannot be released until the transaction commits.

Another limitation that is caused by the isolation is a restriction of cooperation among processes. In some applications it might be a need for sharing uncommitted results of a long-lived transaction which is prevented by isolation. The durability requirement is violated in mobile environments. A *mobile management system* (MMS) deals with mobile computers whose location constantly

[1]Informally defined as those transactions that last for at least the same time magnitude as the mean time between failures of the computer system on which they run [KP92].

changes. MMS traces these changes and according to the location of mobile computers it assigns them to different fixed hosts or servers (i.e., servers process mobile computers' queries). In other words, location information changes without any intervention of mobile client's management system. In a mobile environment frequently submitted queries like weather or traffic conditions are not issued by mobile computers but instead broadcasted by the server to its mobile clients. Therefore, the DBMS resided in a mobile computer does not implicitly place a query but in certain time window its content changes. These particularities of mobile environments contradict with the durability definition.

Advanced transaction models (ATMs) were introduced to combat the enslavement caused by conventional ACID transactions which prevented DBMS from meeting availability and robustness requirements. Merging of ATM models and workflow management would give birth to a family of workflow applications, which more closely reflects the demands of enterprise-wide infrastructure and security. But, it is not sufficient to support an extended transaction model in a WFMS as no single extended transaction model is likely to satisfy the transactional requirements of all the applications. The list of ATM models which WFMS developers can choose from is composed but not limited to *Nested*, *Open Nested*, *Saga* and *Flex* models.

A *Nested Transaction* [Elm92, JK97] consists of a top-level transaction T and a set of component transactions S referred to as subtransactions. T may contain any number of subtransactions, and each subtransaction, recursively, may contain any number of subtransactions, thus forming a transaction tree. A child transaction may start after its parent has started and a parent transaction may terminate only after all its children terminate. The model was proposed to overcome two main limitations of the flat (single level) transactions, i.e., limited parallelism and inflexible failure control. If a parent transaction is aborted, all its children are aborted. However, when a child fails, the parent may choose its own way of recovery. It can restart the child or start another transaction, or even ignore the failure in the case of non-vital subtransaction. Therefore, at the subtransaction level nested transactions allow a user to define finer units of recovery than that in the flat model. The subtransactions of a nested transaction can be executed concurrently ensuring execution atomicity.

Open Nested [Elm92] model relaxes the isolation requirements by making the result of committed subtransactions visible to other concurrently executing nested transactions. Applying such a visibility rule open nested transactions achieve a higher degree of concurrency. To preserve the consistency property for open nested transactions, only commutative subtransactions are allowed to use the results of committed subtransactions.

The main contribution to ATMs made by proposing *Saga* transaction model [GMS87] is that sagas can deal with long-lived transactions. Sagas use the concept of compensating transactions for handling failures. For a transaction T, a compensating transaction C is a transaction that can semantically undo the effects of T after T has been aborted. A *saga* is a set of relatively independent (component) transactions $T_i, i = 1 \ldots n$ which can interleave in any way with component transactions of other sagas. Component transactions are executed in a predefined order within the saga. Each component transaction T_i is associated with a compensating transaction C_i. Both component and compensation transactions behave like atomic transaction preserving ACID properties. Component transactions can commit without waiting for any other component transactions or the saga to commit. However, a saga commits only if all its component transactions commit in a prespecified order. When a saga aborts, compensating transactions are executed in the reverse order of commitment of component transactions. A compensating transaction can commit only if its corresponding component transactions commit but the saga, to which it belongs, aborts. Due to their ACID properties, component transactions make their changes to objects effective in the database at their commitment times.

Flexible Transactions or *Flex* transaction model [ELLR90] has been proposed as a transaction model suitable for a multidatabase environment. A *multidatabase system* is a facility that allows users to access data located in multiple autonomous DBMSs. Multidatabases typically integrate information from pre-existing, heterogeneous local databases in a distributed environment [Bob96]. A *flex* transaction is a set of tasks, with a set of functionally equivalent subtransactions for each task and a set of execution dependencies between subtransactions, including failure-dependencies, success-dependencies, or external-dependencies. The latter two define the execution order on the

subtransactions, whereas the former defines the dependencies of the subtransaction execution on the events that do not belong to the transaction. The execution of a flex transaction succeeds if all its tasks are accomplished. A flex transaction is resilient to failures in the sense that it may proceed and commit even if some of its subtransactions fail. The transaction designer is allowed to specify acceptable states for termination of the flex transactions. To relax the isolation requirement on the subtransaction level, flex uses compensating transactions.

The activities that require to be represented by the transaction models described above are generalized as *open-ended* activities [KP92]. Open-ended activities are characteirzed by:

- *Uncertain duration* - from hours to days;
- *Unpredictable developments* - actions are not foreseeable at the beginning;
- *Interaction with other activities.*

In a workflow, open-ended activities can be structurally presented by *split-transaction* and *join-transaction* models.

A *split-transaction* divides an ongoing activity into two or more activities. In particular, resources of the original activity are divided among all the new resulting activities. Thereafter, each activity proceeds independently with its own resources. In the general case, all the new transactions continue and may commit or abort independent of each other as if they had always been distinct. The new activities are thus not subactivities of the original, but instead effectively replace the original [KP92].

The inverse of the split-transaction is called the *join-transaction.* It merges two or more activities into a single activity and all their work is either committed or aborted together [KP92]. Corresponding to the split points' classification given in Section 3.1, split-transactions are categorized as AND-split, OR-split, AND-join, and OR-join transactions.

Open-ended activities as well are restricted by the ACID properties of the

traditional transaction model and being used in workflow applications arises the need for the employment of ATMs. Although ATMs greatly relax traditional transaction model, few or non of the current commercial products have incorporated transactional support for workflows' management [AAAM98]. One of the reasons for such a limited success is the inadequacy of ATMs. Advanced transaction models are too centred on database concepts, which limits their possibilities and scope as many non-transactional legacy applications are inherited by WFMSs. In fact, WFMSs bear a strong resemblance to advanced transaction models, although addressing a much different and often richer set of requirements. Nevertheless, there are undoubtedly many ideas from the transactional world that can be translated and successfully applied in a workflow environment.

3.3 Legacy Applications

Workflow applications involve legacy tools, which were not developed to be used in transactional environments. In literature [GHS95, KS95] it is said that non-transactional activities lie outside WFMSs and are not controlled by them. Such a statement might create a view that WFMSs are not capable to cope with legacy tools and fail in sophisticated incorporation of non-transactional activities. Thinking about application areas of WFMSs we can see that many of them use a mailing system as a coordination medium. If a mail-based WFMS assumes that mailing is not a controllable task then there would no need in using this WFMS at all. We try to show a possible solution for 'transferring' non-transactional activities to transactional ones.

Assume that an activity uses a legacy application, e.g., e-mail, and according to [KS95] it should be considered as a non-transactional task. The difference between transactional and non-transactional tasks is depicted in Figure 3.2. As it can be seen from the picture, we could wrap the functionality of a non-transactional activity with transactional activity functionality (not in all cases) and name this activity as a transactional one. The structure we would obtain after this wrapping is shown in Figure 3.3.

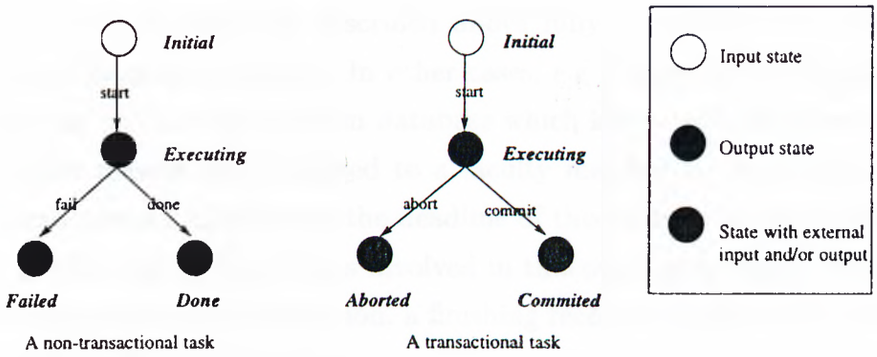

Figure 3.2: Tasks structure

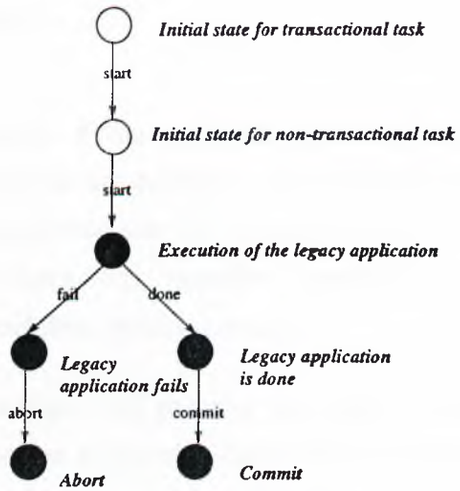

Figure 3.3: Wrapping non-transactional task with transactional properties

Therefore, extending a non-transactional activity by pre- and post-transactional operations (e.g., recording some starting and returning parameters from mailing application in our case), allows us to define abortion and commitment conditions for the legacy application. In the case of abortion we cannot roll back the activity since from the viewpoint of WFMSs it is executed in a 'black box'. Abortion of a non-transactional activity causes its restarting (may be in a certain time period) or cancelling. The action upon abortion is determined by time constraints. The above assumption lets us consider all the tasks as transactional and allows to apply the set of different advanced transaction models to them as well as to the 'pure' transactional tasks.

Human invocation can be treated in a similar way. If it is based on e-mail

communication, the way we described above fully suits the functionality of this kind of human invocation. In other cases, e.g., paper-based cooperation, a record can be made in a system database which keeps track of the workflow states, after papers are submitted to a faculty member for examining. The system can alarm a clock when the deadline of this activity is approaching or send a mail to the person who is involved in this operation. Again, when the papers are returned with a decision, a finishing record is made in the database making this activity committed.

3.4 Priorities

An important parameter of the activities (and the entire workflow) that is difficult to set up a priori is the deadline. The designer can estimate a deadline value but in real implementation the completion time can vary significantly leading to inefficient execution. Another possibility is to use simulation to tune deadlines in a workflow implementation.

The notions of deadline and priority are tightly coupled. In this section we provide an introduction to priority-based and non-priority-based execution. The next section describes the notion of deadline.

In a priority system, the order of execution is determined based on the deadlines assigned to the activities constituting a process. Therefore, in such systems the chain of executed tasks must be commutative to possible changes in the execution sequence; i.e., regardless of the route taken in the workflow schema, the final activity must observe the same data state in the WFMS. We can think of a priority assignment policy as a function that can take two different kinds of arguments: a single activity or a set of activities. When applied to a single activity, the result of the function is the priority of the activity, and when applied to a set of activities, the result is an ordered list of the activities. In a non-priority system, the execution schema is determined by the submission order of the tasks and it cannot be changed unless the inter-task dependencies allow such a reordering. Therefore, in the case of a non-priority system, we include deadline assignment algorithms to see how

well the timing constraints can be satisfied using different algorithms in an environment where the constraints do not affect the execution order. The decision whether a system should be represented as a priority or a non-priority system is taken according to the type of the business process that is going to be automated. In an administrative workflow, where the execution sequence and tasks interdependencies are predefined, the use of priorities cannot bear any benefits and even may not be feasible in general. On the other hand, if the core of a workflow is an ad hoc process, the usage of priorities leads to improvements in performance. In real life it is often difficult to ascribe an organization's management flow exactly to one of the listed management styles, especially in a large-scale enterprises. The consequence is that the WFMSs should be ranged by their types or should be flexible enough to cope with all the variety of business processes.

3.5 Deadlines

The primary technique for meeting real-time requirements in non-priority systems is choosing a suitable algorithm for distributing deadlines for subactivities based on a deadline of an activity, which is set by the workflow schema designer. Deadlines can be expressed in two different ways, either as the allowed execution time for an activity[2], or the time by which a (sub)activity must be completed. In the first definition, a deadline is a time period rather than a certain time point in the future. Therefore, the completion time is a floating parameter and it is calculated when an activity is submitted for execution. The people who have proposed various deadline assignment algorithms for database transactions have agreed on the fact that the subtransaction deadlines should be assigned on-line or a priori with further adjustment [KGM93, SST94, LHK97]. It is indisputable that these approaches are natural for WFMSs considering the range of activities that a single workflow can consist of. Regarding the interpretation of deadlines in the scope of workflows, the former technique (i.e., a deadline is an allowable execution time) seems to be more acceptable, in defiance of the definition of deadlines given by WfMC [Wor96].

[2]without specifying start and completion times.

To illustrate this claim by an example, let us assume that a process (Figure 3.4) can be completed in two commutative ways. Following the first route $(T_0, T_1, T_2, T_4, T_5)$ takes 13 time units; otherwise $(T_0, T_1, T_3, T_4, T_5)$ it takes 61 time units, depending on the output of activity T_1. Assume that deadlines are

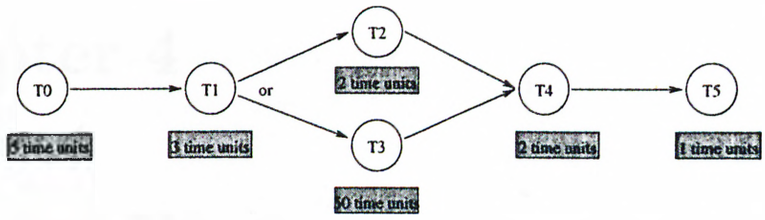

Figure 3.4: Sample workflow schema

absolute (i.e., deadline is interpreted as a time point in the future till which an activity must be finished). While assigning deadlines for activities and for the entire process it is not clear how to determine deadlines for activities T_4 and T_5 (Figure 3.5). Deadlines of T_4 and T_5 can be assigned in two different ways. Assigning a larger deadline (assuming that the second route will be chosen)

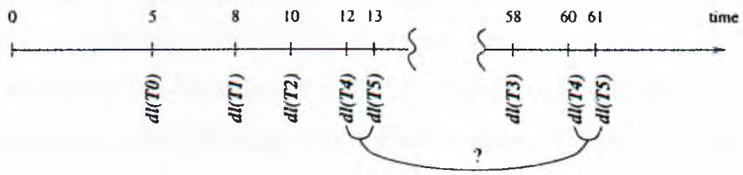

Figure 3.5: Absolute deadlines

may lead to a significant extension of execution time for the process providing that in reality the first route has been chosen. Assigning an earlier deadline for activity T_5 may lead to false abortion (e.g., if an early abortion algorithm is used to predict a possible deadline missing). On the other hand, if deadlines are relative, then activities become independent of each other meaning that whichever route is taken, deadlines remain adequate. In the frame of such an interpretation of deadlines, we do not determine a deadline for the entire process and therefore, do not try to meet the deadline of the whole process. Our aim is to meet the deadline for each activity. Later in the report we will describe and evaluate several deadline assignment algorithms.

Chapter 4

Related Work

4.1 Workflow Models

As it is depicted in Figure 2.3, the final state of the model evolution (that is seen so far) to be used in workflow applications can be either a workflow metamodel or a multitransactional archetype. From the metamodel side, two approaches are the most prominent - ACTA framework [CR94] and the Transactional Specification and Management Environment (TSME) [GHKM94].

ACTA was proposed as a framework for specifying the structure and behavior of complex applications and for reasoning about their transactional properties. ACTA is not a transactional model itself, rather it is a *metamodel*, intended to unify existing models and facilitate their analysis.

ACTA characterizes the semantics of interactions (i) in terms of different types of dependencies between transactions, and (ii) in terms of transaction effects on the objects accessed by the transaction. In ACTA, there can be two types of dependencies between transactions:

- *Commit-dependency:* if a transaction A has a commit-dependency on transaction B, then transaction A cannot commit until transaction B either commits or aborts. It does not imply that the two transactions should commit or abort together.

- *Abort-dependency:* if a transaction A has an abort-dependency on transaction B and if transaction B aborts, then transaction A should also abort. It neither implies that if transaction A aborts, B should abort, nor that if B commits, A should commit.

An object accessed by a transaction can be characterized by its state and its status. The state of an object is simply its contents. The state is changed when an operation invoked by a transaction modifies the contents of the object. The status of an object is represented by the synchronization information (e.g., concurrency control information) associated with the object. The status of an object changes when a transaction performs an operation on that object.

Transaction models are defined by a set of axioms. These axioms determine the rules of concurrent execution for a transaction and describe all the events invoked by the transaction, also indicating the partial order in which these events occur.

The ACTA framework may be useful for better understanding of the nature of interactions between transactions and the effects of transactions on their environment. It makes it possible to analyze particular applications and improve their concurrency properties. It also makes it easier the development and analysis for new extended transaction models suited for a particular environment. However, not all properties of transaction models can be captured and expressed in ACTA, and when an attempt is made to define a transaction with a particular set of properties, ACTA framework proves to be very difficult to use [CHRW98].

The TSME provides an implementation-independent language for transaction specification, as well as the environment in which transactions can be executed. The programmable transaction management mechanism is based on the event-condition-action (ECA) rules. A workflow in this framework consists of a set of constituting transactions and a set of dependencies among them. In addition, workflows have an execution structure that is defined by an ATM; the ATM defines the correctness criteria for the workflow. The TSME claims to support various ATMs to ensure correctness and reliability of various types of workflow processes.

Execution dependencies are based on ACTA notion of dependency. The differences are that TSME recognizes more states of a transaction (e.g., begin, prepare-to-commit), and that more dependencies are considered (e.g., strong-commit-abort dependency on transactions A and B meaning that A *must* abort when B commits).

The separation of transaction specification and transaction implementation in TSME has many advantages:

- It allows the designer to reason about correctness of the transaction model without considering its implementation.
- It allows for re-using existing transaction models.
- Developing new transaction models suited for a specific application is easier, especially if combining dependencies from existing models is possible.

However, the TSME relies very much on the properties of underlying systems for ensuring correct execution of the specified transaction. For this reason, it is possible to specify a transaction that cannot be executed due to the lack of functionality of the component systems. Also, due to the statical approach for defining the set of constituting transactions for a workflow, dynamic transaction models (e.g., split and join transactions) cannot be specified and considered within this framework.

A formalization of different ATMs through ECA rules is presented in [CA95]. In the paper the authors give their view on the required functionality of the underlying database system for a WFMS and provide definitions and implementations of several ATMs in an active database paradigm.

The set of *primitives* based on which the approach is realized, is identified to consist of nested transaction, saga and DOM transaction models (an extension of nested and flex transaction models). The authors keep the opinion that the set of functionalities provided by these models is sufficient to model the behavior of modern applications.

> ...it is unlikely that the approach of rolling new variants of transactions as applications emerge, will provide a realistic solution to the general problem [CA95].

As it was mentioned before, the research community has had almost no impact on the development of existing commercial products. The solutions presented above are still paper-based, not found their implementation. Nevertheless, the first generation of workflow engines has found wide acceptance since if there is a prototype for next generation managing systems this is a workflow management system. Nowadays there are several hundred commercial products that claim to be workflow tools. Some of the most relevant systems in the market include: **Action Workflow System**, of Action Technologies; IBMs **FlowMark**; **WorFlo Business System** of FileNet; **InConcert**, produced by XSoft, a division of Xerox Corp.; **OmniDesk**, of Sigma Imaging System Inc.; **ProcessIT**, of AT&T Global Information Solutions; and **StaffWare**, of StaffWare Corp. [AS96]. These WFMSs still have a long way to go before they can be regarded as mature technology for large-scale enterprise solutions. The most important limitations of current WFMSs were mentioned in Section 2.3.

Workflow on Intelligent Distributed database Environment (WIDE) makes an attempt to overcome some of the limitation [Wid], namely flexibility and centralized database support (if a centralized database lies in the foundation of a WFMS it quickly becomes a bottleneck of the system[1], crossing the scalability property).

The main objective of the WIDE project is to extend the technology of distributed and active databases, in order to provide added value to advanced, application-oriented software products implementing workflow techniques. In frame of the results reported recently on the 1st International Symposium on Advanced Database Support for Workflow Management [Int], the most relevant to our work are the following:

- A conceptual model integrating workflows and database technology has been developed.

[1] single point of failure

- A workflow description language, combining the specification of workflow with access to external databases, has been proposed.

The WFMS architecture is based on active rules. Rules are generated from work tasks (WT) specifications. Each WT have five major characteristics [CCPP95]:

- *Name:* identificator of the WT.
- *Description:* a few lines in natural language describing the WT.
- *Preconditions:* a boolean expression that must yield a truth valua before the action can be executed.
- *Action:* a sequence of operations executed when an appropriate precondition switches to a true value.
- *Exceptions:* is set to handle abnormal events. When an exception is realised a reaction is selected among a restricted set of options that includes END (termination of the task), CANCEL (the task is cancelled), NOTIFY (a message is sent to the person responsible for the task).

Tasks in the WFMSs can be represented using only two models. Global activities are represented by the saga transaction model and local activities by the nested transaction model. This approach imposes restrictions on implementation of dynamic structures (e.g., OR-split, VOTE-split, AND-split activities). The system as well as its predecessors does not concern about performance evaluation capabilities for designed workflows. Whereas, simulation is recognized to be a powerful tool for managerial decision making.

4.2 Simulation

The approach of merging workflow and simulation technologies is presented in [BMM96]. The authors propose an architecture of a Workflow Analysis and

Design Environment (WADE), which is described as a simulation-based tool of next generation workflow systems.

The niches for using simulation in workflow design are clearly outlined in the paper. They include:

- measuring the performance of existing systems,
- identifying improvement opportunities,
- evaluating the effect of alternative operational policies on system performance,
- comparing alternative system designs.

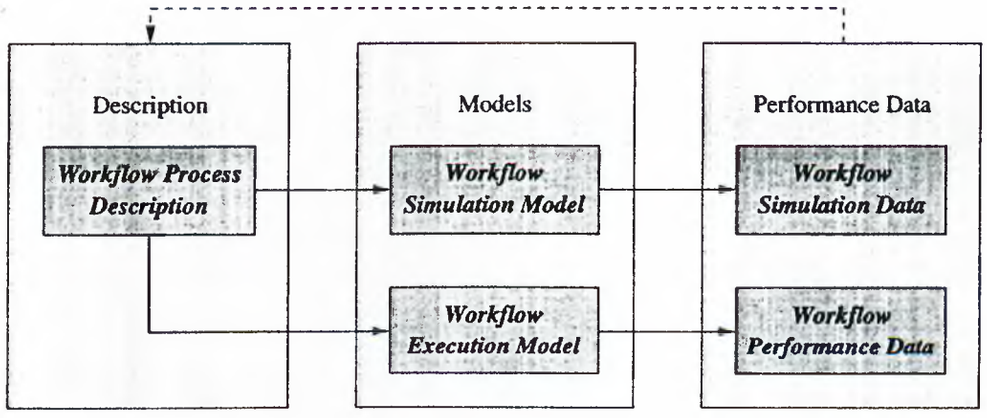

Figure 4.1: Relationship between simulation and execution models in WADE

WADE is intended to support the design and analysis of workflow systems in the context of continually evolving business processes. WADE provides automated support for generation of workflow simulation and execution models. The vision of relationship between these two models is shown in Figure 4.1.

Although this approach is quite useful and applicable to workflow area, there are several shortcomings in it. The first and the most significant one is that this approach is superficial. It implicates only the structural aspects of designed workflows not taking into account the possible task implementation policies. Secondly, as in the majority of workflow products, the transactional

aspect is not mentioned in the system documentation. Thirdly, the system supports neither a priority notion nor deadline policies.

The first performance analysis techniques based on different deadline assignment strategies for workflow applications were proposed in [PR97, PR98b, PR98a]. Our study on the non-priority system setting is largely based on the techniques proposed by these authors. We refer the reader to Section 5.2.1 for more detailed description of deadline assignment algorithms.

Chapter 5

Simulation Model

5.1 Model Description

Our schema (Figure 5.1) consists of eleven activities each denoted by T_i where $i = 1 \ldots 11$. Activity T_1 starts the whole process by getting an application package from an applicant. It uses e-mail that is a legacy application, and according to our assumptions from Chapter 3, we can treat it as a transactional activity providing that its starting and finishing states are recorded in StaffDB database and controlled by the WFMS. We represent activity T_1 by the flat (single level) transaction model which preserves ACID properties.

Activity T_2 fills in the faculty database (GeneralDB) with the initial data, such as the information taken form the application form, the applicants' grades, and other data required for the acceptance procedure. T_2 is also represented by the flat transaction model. This activity is executed once for each applicant and it must be committed to make the process continue.

Activity T_3 accesses GeneralDB and StaffDB in order to determine if there is an available position in any of the departments listed by the applicant. We assume that the limits for the allowed number of positions are different for native and foreigner applicants. T_3 is represented by the flex transaction model and it consists of a set of subactivities $\{T_{31}, T_{32}, T_{33}, T_{34}\}$. T_{31}, T_{32}, T_{33} are the alternative paths for getting the information about available positions

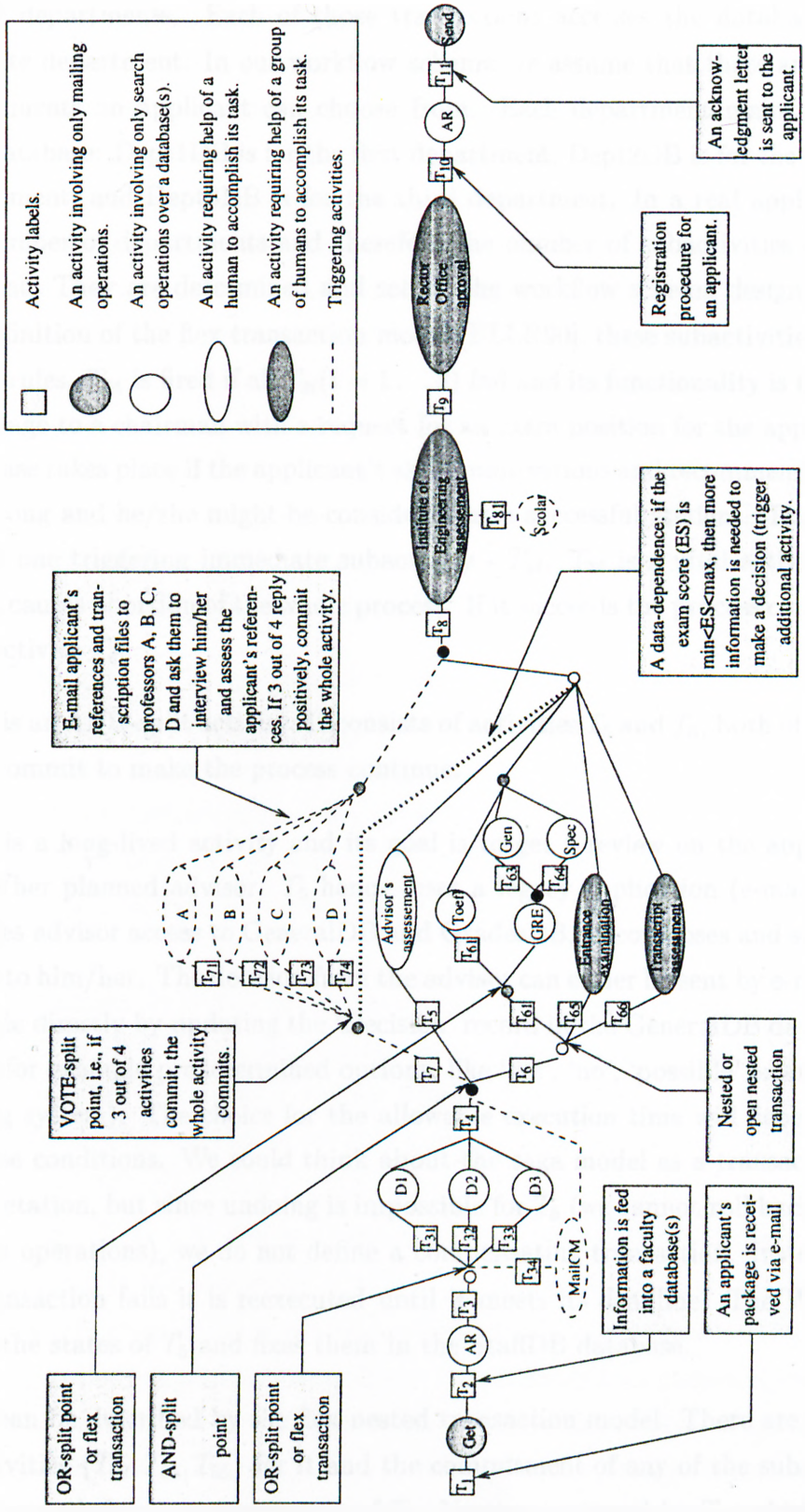

Figure 5.1: Workflow schema

in the departments. Each of those transactions accesses the database of a separate department. In our workflow schema we assume that there are three departments an applicant can choose from. Each department maintains its own database: Dept1DB is for the first department, Dept2DB is for the second department, and Dept3DB is for the third department. In a real application the number of departments and therefore the number of subactivities can be different. They are determined and set by the workflow schema designer. By the definition of the flex transaction model [ELLR90], these subactivities obey ACID rules. T_{34} is fired if all $T_{3i}(i = 1 \ldots 3)$ fail and its functionality is to send a message to a chairman with a request for an extra position for the applicant. This case takes place if the applicant's skills, motivations and recommendations are strong and he/she might be considered as a successful student. This way, T_3 has one triggering immediate subactivity - T_{34}. T_{34} is a vital activity; its failing causes abortion of the whole process. If it succeeds the process continues with activity T_4.

T_4 is an AND-split activity. It consists of activities T_5 and T_6, both of which must commit to make the process continue.

T_5 is a long-lived activity and its goal is to get a review on the applicant by his/her planned advisor. T_5 hence, uses a legacy application (e-mail). T_5 provides advisor access to GeneralDB and GradesDB, or composes and sends a report to him/her. The decision from the advisor can either be sent by e-mail or be made directly by updating the 'Decision' record in the GeneralDB database (using for example predetermined options, like 'yes', 'no', 'possible' or another grading system). The choice for the allowable execution time will depend on all these conditions. We could think about the saga model as a transactional interpretation, but since undoing is impossible for T_5 (we cannot roll back mail server's operations), we do not define a compensating transaction. Instead, if the transaction fails it is reexecuted until it meets its deadline. The WFMS tracks the states of T_5 and fixes them in the StaffDB database.

T_6 can be described by the flex nested transaction model. There are three subactivities $\{T_{62}, T_{66}, T_{65}\}$ for it and the commitment of any of the subactivities is enough for the commitment of T_6. Nesting is caused by T_{62} which has three subactivities organized in a tree. T_{61}, T_{62} and T_{64} access GradesDB and

make a conclusion about meeting the requirements of the official department acceptance policy. Throughout the simulation, T_6 will be implemented as a nested transaction, an open nested transaction, and a saga in three separate experiments. The commit time will depend on that choice in a heavy loaded system. For example, *Transcript assessment* (activity T_{66}) is a long procedure and if T_{66} 'knows' that T_{65} (the subactivity responsible for the entrance examination assessment) has already been committed, then this procedure can be aborted without affecting the result of T_6 or even can be not started at all. This discussion applies to the case we represent T_6 by a flex or OR-split transaction model. In some cases it might not be the case depending on the application, for example it is possible that only AND-split is allowed at this step. The flexibility and performance at this point depends on the locking technique used by the DBMS. Priorities for the subactivities are determined according to the graduate school policy.

If T_6 fails, which means that either the advisor has assessed the applicant as unsuccessful or the applicant's grades do not meet the department requirements, or both, then activity T_7 is triggered. Here we are coming out of the frames of basic definitions that are given in Section 3.1. Activity T_7 cannot be represented either by OR-Split or by AND-Split point. For this activity we should give a definition for a new type of split point named a VOTE-Split.

> **Definition** *VOTE-Split point is a point within the workflow where a single thread of control splits into three*[1] *or more activities. A quorum of committed activities defines the result of the original activity. This quorum is set by a workflow schema designer.*

Consequently T_7 is a VOTE-split activity. The commitment condition for it in our schema is the following: if 75% of its subtransactions are committed then the whole transaction is also committed (clearly, the proportion may vary according to the graduate school policy). T_7 consists of four subactivities each of which accomplishes a similar procedure. Under the control of these four subtransactions, i.e., $\{T_{71}, T_{72}, T_{73}, T_{74}\}$, the application package is sent to each of the members of a selected jury of faculty members. Then the votes of jury

[1]The case of two risen activities is equal to the or-split point.

members (in the form similar to the one for T_5 activity) are collected and the decision is taken according to the defined voting rules. Another possible voting technique is based on the summation of priorities of committed subactivities. Note that T_7 is a triggered activity for T_4. If T_4 fails; i.e., neither the grades are high enough nor the jury has voted 'yes' for the applicant, the whole process is aborted; i.e., the applicant is rejected. Otherwise, a file is composed and sent to a higher level; e.g., the Institute of Engineering Admission Office and then to the Rector Office.

Activity T_8 is responsible for getting an approval from the Institute of Engineering Admission Office and deciding for a possible scholarship. This way, T_8 has one triggering activity T_{81}, which accesses ScholarDB and tries to assign to applicant a scholarship according to the scholarship type suggested by the department.

Upon commitment of T_8, T_9 is started to get an approval from a higher level of administration hierarchy. The functionality of T_9 is similar to that of T_8. T_9 is represented by the flat transaction model. After the successful completion of T_9, the applicant is registered in the GradStudDB, which stores the information about accepted graduate students. Activity T_{10} accomplishes this process. But there is still a possibility for abortion in some cases (e.g., the applicant may not be satisfied with the scholarship).

The last activity, T_{11}, sends a resulting decision and some more information (if needed) to the applicant. Its structure is similar to that of T_1.

Facilities' usage for our workflow schema is depicted in Figure 2.2.

5.2 Deadline Assignment Algorithms

In this section we present a number of deadline assignment strategies for activities that will be examined by using our schema. As it was mentioned before, there are two major classes for transactional systems: priority and non-priority. Let us take a deeper look at the difference between a non-priority and a priority system from the point of view of deadlines. Remember that in the non-priority

set up for our workflow schema, we consider a deadline for an activity as allowed execution time and the workflow schema as a non-prioritizied set of activities. On the other hand, assuming a priority system, we refer to a time point in the future as the deadline for an activity. To explain this type of systems we refer to active and real-time DBMSs[2]. In contrast to an active DBMS or a real-time DBMS, a WFMS executes much broader spectrum of tasks and involves more heterogeneous legacy tools. In turn, these tools use their own DBMSs and data storages, and do have established execution policies. This fact lets us assume that in such a system possibility of a bottleneck situation is lower than that in any DBMS and it is an inner problem of the legacy tool. In many cases a WFMS cannot resolve the problem 'inside' a legacy tool and has to abort and reexecute the casual activity. In active and real-time DBMSs, priorities can be used for reducing the number of abortions. Priority execution can be explained as follows. A process can be treated as a set of independent or quasi-independent activities, which requests one or a few services and data storages. In order to avoid data or services' contention, the priorities are involved in execution of activities. A priority scheme reorders the execution sequence in such a way that the process wastes as little time as possible (reordering is allowed and does affect the final result of the process). On the other hand, in a WFMS a workflow schema predefines the execution sequence and often the order must be preserved. Nevertheless, the notion of a deadline is applicable to both a priority and a non-priority system.

Priority assignment algorithms are based on deadlines of activities. In order to be used as a base for priority assignment, deadlines should have a common absolute measure scale. For example, if we say that a deadline of one activity is 5 time units and a deadline of another is 10 time units, it must mean that the second activity is less urgent than the first. In other words, deadlines within a process should have a 'common denominator'. This is the case when a deadline is expressed as a point of time in future till which an activity must (should) be finished. As it was shown in Section 3.5 this principle is not easily applicable for WFMSs. Let us discuss the case when a deadline is expressed as an allowed execution time.

[2]Active and real time DBMSs are conceptual predecessors of WFMSs.

Saying that an activity, which has a deadline of 5 time units, is more urgent than another activity with 10 time units deadline is not always true. It just means that the first activity has 5 time units from its starting point to finishing point and the second has 10 time units and they cannot consume more time for execution. The order of execution for these two activities is defined by the underlying workflow schema and the current state of the process.

For the two different perceptions of the deadline concept that we discussed above, we use two different sets of deadline assignment techniques. In the first set of algorithms, the deadline of an activity is assigned based on the execution time of the activity and the available unused time left by previous activities. The second set of algorithms makes use of the arrival times of activities and a deadline for the entire process. We describe both sets of techniques later in this section.

We first introduce the parameters that describe each activity. The list of parameters has been chosen based on simulation experiment settings for real-time database systems [KGM93, KGM94, LHK97, SST94]. The basic parameters for a non-priority system study can be specified as follows:

- deadline of an activity T_k - $dl(T_k)$
- escalation cost of T_k - $esc(T_k)$
- failure probability of T_k - $q(T_k)$

Deadline of an activity T_k is the allowed execution time for it. Initially all the activities are assigned deadlines which can be changed during execution according to the deadline assignment policy employed. Escalation cost includes losses in terms of time, caused by abortion of an activity. For all the activities we introduce a failure probability. For example, for activity T_{63} (Figure 5.1) the meaning of the failure probability is that, GRE General Test score is lower then that needed to be accepted; for T_{81} the failure means absence of available financial support.

As derivative the following parameter can be considered:

- available slack time - avl_sl
- average execution time of T_k - $avg\_exec(T_k)$

The value of the available slack time for an activity is recalculated whenever its predecessor activity is finished. Therefore, the starting activity (T_1) does not receive any slack. After it finishes, the available slack is calculated as follows.

$$avl\_sl = dl(T_1) - exec(T_1)$$

where $exec(T_1)$ is the execution time of T_1.

The average execution time ($avg\_exec()$) is a statistical parameter calculated for each activity throughout multiple runs of that activity. In the performance experiments, each execution time is set as a random time period within a deadline. Therefore, at each life cycle the execution time of a particular activity can change.

Each activity is associated with one or more agents or facilities. Facilities represent applications which are involved by the WFMS. Systems that provide a single activity with few facilities are called *multiple node* systems. Therefore an activity is associated with a set of agents that can serve it. This set for an activity T_k is denoted by S_{T_k}. For such systems we add two more parameters:

- number of agents in the system - n $(A_1 \ldots A_n)$
- average length of the queue of an agent A_i - $avg\_q(A_i)$

For both priority and non-priority systems we consider the following two additional parameters:

- number of resource units accessed by T_k - $num\_ru(T_k)$
- number of data items accessed by T_k - $num\_di(T_k)$

By using these characteristics we consider our system as service and/or data oriented. In a *service oriented system*, there is a most frequently used facility

each activity requests a service from. In such an environment it is quite logical to prioritize the incoming activities by the number of times they will request this facility. In our system the mail facility is the subject of frequent requests.

We also should add one more parameter for priority system simulation settings, which is

- arrival time of an activity T_k - $ar(T_k)$

Using this parameter we are able to calculate the absolute deadline for an activity. Since we assume that deadlines are absolute only in a priority system, this parameter is used in the second set of deadline assignment algorithms.

Before describing the algorithms, we should explicitly underline the differences between prioritized and non-prioritized execution. The following statements are implemented in our simulation experiments.

In a non-prioritized system:

- The whole process is not associated with a deadline.
- Each activity is assigned a deadline and if the deadline is not satisfied, the activity fails.
- Failing of a vital activity causes failure of the whole process.

In a prioritized system:

- The whole process has a deadline and the process fails if the deadline is not satisfied.
- We do not assign initial deadlines for activities.
- If an activity misses its deadline (which is assigned dynamically), this does not affect the execution.

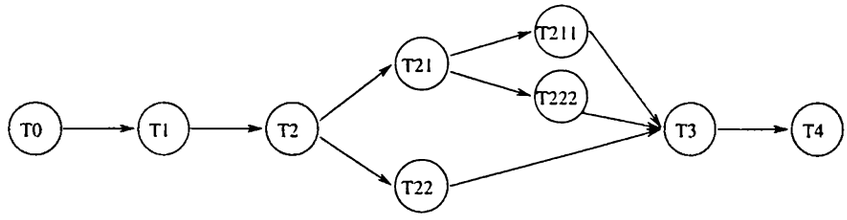

Figure 5.2: Sample workflow schema

5.2.1 Deadline Assignment Algorithms for Non-Priority Systems

Let us switch to the deadline assignment techniques that we use in the simulation. Deadline assignment techniques for a non-priority system simulation that we adopted mostly from [PR97] are listed bellow. We denote a deadline calculated using a deadline assignment algorithm for an activity T_k as $eff\_dl(T_k)$ and call it an *effective deadline.*

- No Adjustment (**NOA**): This policy does not allow any changes in the initially assigned deadlines. Therefore, the execution sequence is not sensitive to possible delays in the system. Such a policy relies on a highly robust environment.

- Total Slack (**TSL**):

$$eff\_dl(T_k) = dl(T_k) + avl\_sl$$

  In this policy all the available slack time is added to the deadline of the activity which is going to be executed next. Such a distribution can lead to consuming of all the slack time by the currently executing activity without any consideration about needs of its successors.

  In **NOA** and **TSL** we do not use any additional information about workflow. If we collect some statistics and derive avg_exec characteristics for activities, we can use other, more sophisticated methods.

The following three strategies need some explanations. These strategies are applied recursively. For the workflow schema shown in Figure 5.2, it would

work as follow. At submission time of T_1 the available slack is calculated for T_1 considering the execution times for T_2, T_3, and T_4. At T_2's submission point, we calculate the portion of the available slack time for it and then repeat the slack time distribution for T_{21} and T_{22} using the same formula (T_2's portion of the available slack time is considered as the entire available slack time for T_{21} and T_{22}). Then we distribute slack portions for T_{211} and T_{212} again proceeding on the assumption that the available slack time for them is the slack portion of T_{21}. We assume that splitting activities (T_2 and T_{21}) carry information about execution times of their immediate subactivities.

- Proportional Execution (**PEX**): In WFMSs, fully automated activities can have deadlines in terms of seconds, and the other activities which require human invocation, can have deadlines in term of hours. Therefore, assigning a second slack to an activity with an hour deadline or vice versa does not seem logical. Moreover, it can lead to an abortion situation which could be avoided if another deadline assignment technique is used. Proportional execution strategy tries to smooth off this shortcoming by distributing the slack according to the average execution times of the activities. Assuming that the workflow schema consists of N activities we can calculate the effective deadline of an activity T_k by the following formula:

$$eff\_dl(T_k) = dl(T_k) + avl\_sl * \frac{avg\_exec(T_k)}{\sum_{j=k}^{N} avg\_exec(T_j)}$$

- Proportional Escalation (**PES**): If the escalation cost is a crucial point for the underlying business processes, it is desirable to distribute the available slack using the escalation cost associated with each activity.

$$eff\_dl(T_k) = dl(T_k) + avl\_sl * \frac{esc(T_k)}{\sum_{j=k}^{N} esc(T_j)}$$

- Proportional Load (**PLO**): When in a multiple node system, several agents or facilities are used, the length of the facility queues can be used as a distribution parameter. This technique can be useful in heavily loaded systems within ad hoc business processes. Assume that activity T_k is ready for execution and a facility A_i will execute it. There are several

facilities in the set S_{T_k} which can execute T_k.

$$eff\_dl(T_k) = dl(T_k) + avl\_sl * \frac{avg\_q(A_i)}{\sum_{A_j \in S_{T_k}} avg\_q(A_j)}$$

The following algorithm makes use of the parameters *num_ru* and *num_di*. In Section 5.2 we have provided the definition of *service oriented* systems. The execution process in this kind of systems is built on requests to system facilities. The facilities in turn take up system resources. Beyond traditional data resources, system resources can be represented by human resources, mail server resources, and resources[3] of another organization involved in the workflow. Via the *num_ru* parameter the resources that are not of data type are taken into account. Activities that use software applications (e.g., in our workflow 'Search and Registration Tools') and request read and/or write accesses from the system databases, are assigned the *num_di* parameter. The reason we make a distinction between these two parameters is in great time superiority of a single resource unit access over a single data item access. A method based on data access parameters was studied in [LHK97]. The authors employed this method for priority assignment in a real-time system. Inspired by the good performance of it presented by the authors, we propose the following algorithm for non-priority system simulation.

- Proportional Resource Usage (**PRU**): This policy assigns deadlines to activities based on the number of data items the activity accesses and the number of services the activity needs to request to accomplish its goal. The *avg_data* and *avg_serv* parameters carry information about the average time of a single data item access and the average service time consumed, respectively.

$$eff\_dl(T_k) = avg\_serv * num\_ru(T_k) + avg\_data * num\_di(T_k) + avl\_sl * \frac{num\_ru(T_k) + num\_di(T_k)}{num\_ru(W) + num\_di(W)}$$

  Where $num\_ru(W)$ and $num\_di(W)$ are the total number of resource units and data items accessed by the entire workflow W, respectively.

[3] in terms of person/time coefficient, for example.

5.2.2 Deadline Assignment Algorithms for Priority Systems

In dealing with a priority system where the execution sequence can be changed during the life cycle, we express a deadline as a time till which an activity should be finished. Consequently for priority systems, we use different deadline assignment techniques that are mostly adapted from [LHK97].

- Ultimate Deadline (**UDL**): Assume that we have a workflow W, then according to this policy the deadline of W is assigned to each activity T_k constituting the workflow schema.

$$eff\_dl(T_k) = dl(W)$$

- Effective Deadline (**EDL**): The previous policy does not take into account any information about the activities of a workflow schema W and therefore does not discriminate between long-lived and traditional activities. One possible method to overcome this shortcoming is the effective deadline policy. Assuming that N is the total number of activities in W,

$$eff\_dl(T_k) = dl(W) - \sum_{j=k+1}^{N} avg\_exec(Tj)$$

  The problem with **UDL** and **EDL** is that they allocate all the remaining slack time of the workflow to the currently executing activity.

- Equal Slack (**EQS**): This heuristic evenly distributes the slack among the remaining activities.

$$eff\_dl(T_k) = ar(T_k) + avg\_exec(T_k) + \frac{dl(W) - ar(T_k) - \sum_{j=k}^{N} avg\_exec(T_j)}{N - k + 1}$$

- Equal Flexibility (**EQF**): This policy uses the notion of *flexibility* which is the ratio of the slack of an activity to the execution time of that activity. It distributes the slack of activities proportional to their average execution times.

$$eff\_dl(T_k) = ar(T_k) + avg\_exec(T_k) + (dl(W) - ar(T_k) - \sum_{j=k}^{N} avg\_exec(T_j)) * \frac{avg\_exec(T_k)}{\sum_{j=k}^{N} avg\_exec(T_j)}$$

- Contention Ratio (**CTR**): In this heuristic activity deadlines are assigned based on a function which includes real-time properties of activities besides the number of data items accessed by activities and the number of services it requests.

$$eff\_dl(T_k) = ar(T_k) + avg\_exec(T_k) + (dl(W) - ar(T_k) - \sum_{j=k}^{N} avg\_exec(T_k)) * cont\_ratio(T_k)$$

  Where $cont\_ratio(T_k) = 1 - \frac{num\_ru(T_k)+num\_di(T_k)}{num\_ru(W)+num\_di(W)}$. The larger the number of data items and/or recourses an activity accesses, the smaller *cont_ratio* is, and therefore the earlier is T_k's effective deadline (i.e., it has higher priority). By raising the priority of an activity it is hoped that the activity can complete earlier and the degree of data contention in the system can be reduced. **CTR** thus considers both the deadline requirement of activities and resource contention.

Having introduced the algorithms for our study we would like to explicitly note some aspects, regarding parallel execution, that might be shadowed before.

The algorithms we proposed are suitable for sequential execution. One can find a lot of examples of workflow applications in the literature [Kim95]. These applications include:

- mail routing in office computing,
- loan processing,
- purchase order processing,
- service order processing in telecommunication,
- product life-cycle management,
- health care management.

In most of the application areas listed above, the workflow schemas are predefined and required to accomplish several tasks in a certain order. Additionally, it is clear that the basic functionality of WFMSs is not of a computational

nature. They are not devoted to complex mathematical computations but to managing a set of activities. Parallel execution has shown great performance improvements in large computational tasks for which it is deeply studied. In our schema parallelism might exist in two ways, but they are not related to deadline assignment. Firstly, an activity or a few of them could be computational tasks that may or may not be executed in parallel[4]. For the activities of this type we would assign deadlines using the same algorithms assuming that the average execution time for them can be estimated. Secondly, we might introduce parallelism by letting the system facilities have several servers and serve more than one activity at a time. But again, due to the predetermined execution order, the situation of having more than one activity requesting the same facility is not likely. Nevertheless we study this case using the **PLO** policy. From the above discussion we may conclude that the need of parallel execution in the workflow area is weak since intrinsically WFMSs do not perform any complex computations by themselves. Moreover, in many ways, a WFMS is not different from a sophisticated scheduler in which the scheduling is performed based on inter-activity dependencies, organizational structure, staff availability, and existing computing infrastructures.

[4]Depending on the time constraints and the hardware platform.

Chapter 6

Implementation

6.1 Implementation Notes and Assumptions

6.1.1 Program

Our model is implemented in Sim++ simulation package [Fis95]. Since we study two different workflow system settings, corresponding to priority and non-priority systems, two main programs were written. The reason for separating the code in two different programs is that in priority simulation we use a preemption mechanism. Each program consists of an input data file with the initial parameter settings for the activities; facilities and a facilities monitoring tool; activities' pool and a mechanism for their monitoring. After an activity has been scheduled for execution, it is placed in a linked list, which is called a *Future Event List* in Sim++. The order of this list is defined by the activity submission times (for non-prioritized simulation) or by the activity priorities (for prioritized simulation). All the facilities are implemented as single servers.

6.1.2 Simulation

In our simulation we try to be as close to the application area as possible. Since we simulate an administrative process, the value of one simulation time unit is equal to one minute. For simplicity we will keep the notion of time unit in our further discussions. Our simulation cycle is equal to 480 time units. This value is prompted by the duration of one business day (8 hours). This way, in our experiments we study the following workload values: $0.0021, 0.0042, 0.0084, 0.0168, 0.0336$ and 0.0672, which are respectively equal to $1, 2, 4, 8, 16$ and 32 submitted applications per one business day. The interarrival time is uniformly distributed. The higher workload values are not considered since, as it will be shown later, in our system the maximum number of processed application in a single business day varies from 5 to 7 depending on the deadline assignment technique employed.

Activity	Transaction Model
T_1	Saga
T_2	Flat
T_3	Flex
T_{34}	Saga
T_4	AND-split, VOTE-split, Flex or OR-split
T_5	Saga
T_6	Nested, Open Nested, Nested Saga
T_{62}	Nested
T_7	VOTE-split
$T_{71} - T_{74}$	Sagas
T_8	Saga
T_9	Saga
T_{10}	Flat
T_{11}	Saga

Table 6.1: Transaction Models

6.2 Transaction Models

Table 6.1 shows the transaction model settings for our workflow schema. The subactivities not included in the table are implemented using the flat model.

Activity	$dl(T_k)$	$esc(T_k)$	$q(T_k)$	$num\_ru(T_k)$	$num\_di(T_k)$	$avg\_exec(T_k)$
T_1	8	5	0.1	1	2	6.037
T_2	3	2.2	0.2	0	8	1.146
T_3	40	5	0.2	1	4	10.24
T_{31}	0.15	0.01	0.4	0	1	0.102
T_{32}	0.15	0.01	0.4	0	1	0.99
T_{33}	0.15	0.01	0.4	0	1	0.103
T_{34}	30	3	0.2	1	1	19.54
T_4	170	7	0.7	13	26	95.46
T_5	35	7	0.3	1	9	30.04
T_6	140	10.5	0.2	4	13	81.62
T_{61}	0.075	0.2	0.5	0	1	0.032
T_{62}	0.23	0.6	0.6	0	3	0.128
T_{63}	0.075	0.2	0.3	0	1	0.049
T_{64}	0.075	0.2	0.3	0	1	0.044
T_{65}	77	5	0.2	2	5	45.98
T_{66}	44	5	0.2	2	5	27.59
T_7	70	10	0.4	8	4	40.83
T_{71}	25	3	0.3	2	1	12.77
T_{72}	25	3	0.2	2	1	13.01
T_{73}	25	3	0.4	2	1	13.18
T_{74}	25	3	0.1	2	1	14.66
T_8	15	15	0.1	1	2	6.642
T_{81}	5	2	0.2	0	1	3.381
T_9	10	15	0.1	1	1	6.427
T_{10}	3	4	0.1	0	8	4.821
T_{11}	8	5	0.1	1	2	7.21

Table 6.2: Initial settings for non-priority system simulation

6.3 Simulation Results and Performance Analysis

6.3.1 Non-Priority System Simulation

Table 6.2 provides the initial settings for non-priority system simulation. The values of $dl(T_k)$ and $q(T_k)$ parameters are prompted by common sense and a practice in the application area. Settings for $esc(T_k)$ parameters are derived from timing cost of abortion of the corresponding activities. $num\_ru(T_k)$ and

$num\_di(T_k)$ parameters are taken from the execution requirements. The values of $avg\_exec(T_k)$ are obtained through the set of 50 runs of our workflow. In this experiment the *avg_data* and *avg_serv* parameters are defined for the **PRU** algorithm. They are set to 0.081 time units and 17.36 time units, respectively.

The evaluation of the results presented in Figures 6.1-6.12 enables us to provide suggestions in terms of:

- optimal system throughput,
- recommended capacity of facilities,
- possible transaction models and control structures for the better system performance.

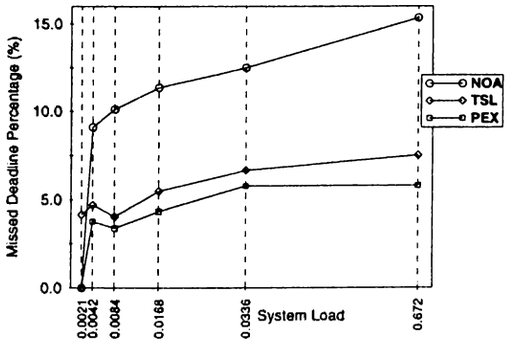

Figure 6.1: Missed Deadline Percentage

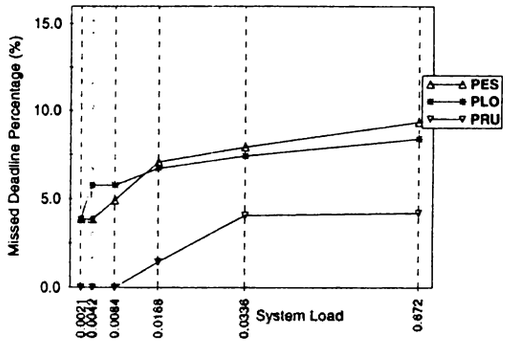

Figure 6.2: Missed Deadline Percentage

From Figures 6.1-6.2 we can judge about the worst and the best performing strategies. Under all load values (except for the first one) the **NOA** strategy exhibits the worst performance in terms of missed deadline percentage. On the other extreme there is the **PRU** strategy which provides the best performance results for our workflow schema not only in the terms of missed deadline percentage but also in terms of system throughput. The other strategies can be partitioned into two groups according to the their slack distribution approaches. **TSL** comprises the first group whereas, **PEX**, **PES**, and **PLO** belong to the second. Under a light system load **TSL** performs worse than the strategies of the second group. Due to the slack distribution approach used by this policy, the whole available slack time can be consumed at the first stages

of the executing process. Nevertheless, in presence of concurrency, i.e., under a higher system load, it outperforms **PES** and **PLO**. In a heavily loaded system the process failure is often caused by failures of the first activities which is in turn caused by a long waiting time at the early stages of the executing process. Since **TSL** assigns the whole available slack time to any activity (including first activities), the policy can overcome this situation. Among the strategies comprising the second group, the **PES** strategy exhibits the worst performance under high loads. The value of the escalation cost parameter increases for the last activities therefore, the **PES** algorithm assigns a larger portion of the available slack to the latter activities. As it can be seen form the initial settings (Table 6.2), in our schema T_4 is the longest activity and it needs to be assigned an adequate deadline whereas, under **PES** policy the latter activities (e.g., T_8 and T_9) are assigned larger deadlines. The strategy would perform fair if the execution times of activities were close to each other. In our schema this is not the case; the last activities are mostly shorter than the first ones. Under the **PES** policy activities T_8 and T_9 for example, are assigned slack portions of 16.22 and 6.41 time units, respectively, and activity T_4 is assigned a slack portion of 6.13 time units; however the execution time of T_4 (95.46 time units) is an order of magnitude higher than that for T_8 (6.642 time units) and T_9 (6.427 time units).

The results given in Figures 6.3-6.6 show the system throughput. The maximum number of processed applications for our schema is 7, which is achieved under the **PRU** policy. Again, the worst performing strategies among those in the second group are **PES** and **PLO**. These policies distribute the available slack based on escalation costs of activities and the lengths of facility queues respectively, not taking into account the execution time. **TSL**, which assigns all the available slack to the currently executed activity, and **PEX**, which works based on the *avg_exec* parameter, shows better performance results than **PES** and **PLO**. Under the highest system load **NOA** as well outperforms **PES** and **PLO** since it uses the deadlines estimated based on an experience in the application area.

Facility utilizations are displayed in Figures 6.7-6.12. The bottleneck facilities are 'Advisor/ChairMan' and 'Staff Members'. Because, these facilities

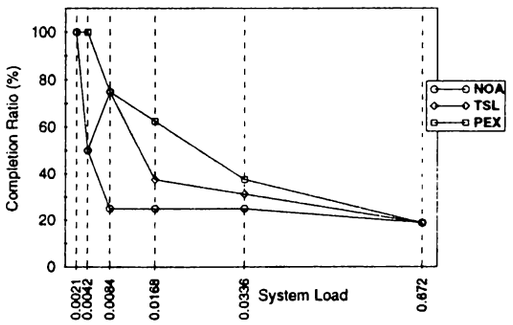

Figure 6.3: Completion Ratio

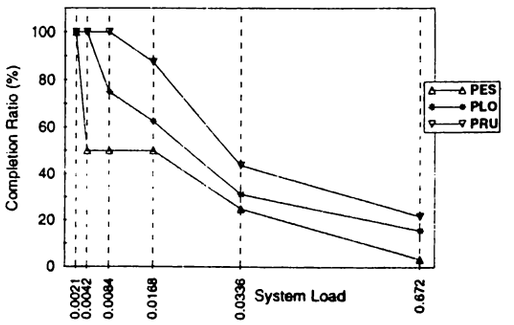

Figure 6.4: Completion Ratio

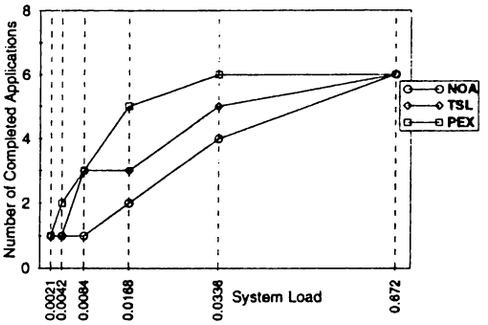

Figure 6.5: Number of Completed Applications

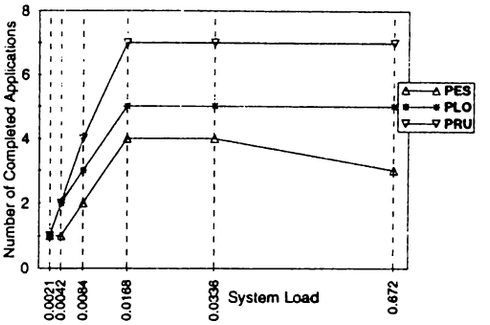

Figure 6.6: Number of Completed Applications

quickly become fully used under increasing system load. Remember that in our settings all the facilities are realized as single servers. The results of the simulation suggest us to extend the capacities of these facilities. With the **PRU** strategy, for the better performance we pay by higher facility loads. Under this strategy, as is seen from Figure 6.12, we should extend the capacity of the 'Mail Server' facility as well.

Having studied the behavior of the six deadline assignment techniques, namely **NOA**, **TSL**, **PEX**, **PES**, **PLO** and **PRU**, we clarify the following peculiarities inherent in them. **NOA** does not give any flexibility in using the unutilized time for improving the performance of the system. We have used this strategy to provide a basis for evaluating the performance benefits of the other strategies. Due to the dynamic nature of our workflow schema (OR-split, VOTE-split activities) **TSL** outperformed the **PEX**, **PES** and **PLO** strategies. **PEX**, **PES** and **PLO** distribute the available slack among *all* the future activities based on a corresponding activity parameter. Therefore, these strategies take into account the further activities that may not be chosen for

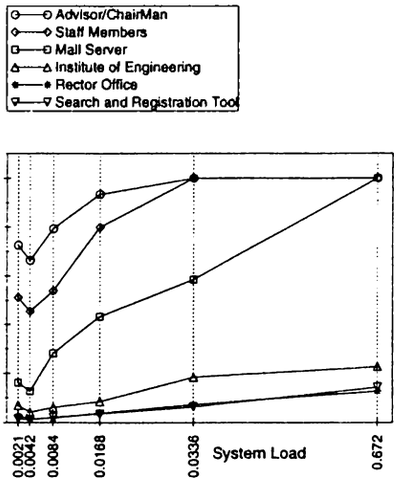

Figure 6.7: NOA, Facilities' Utilization

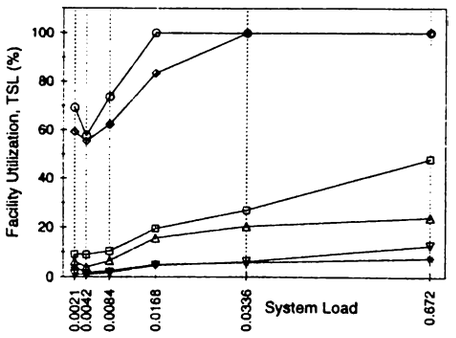

Figure 6.8: TSL, Facilities' Utilization

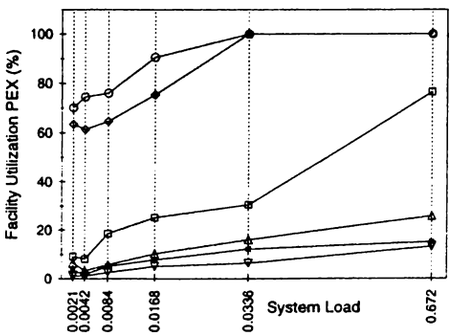

Figure 6.9: PEX, Facilities' Utilization

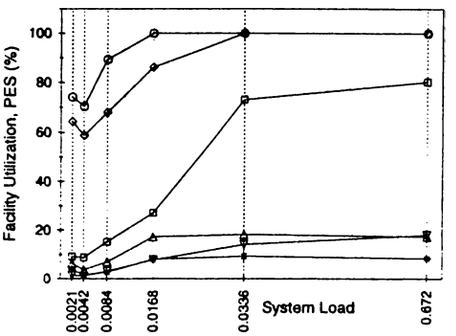

Figure 6.10: PES, Facilities' Utilization

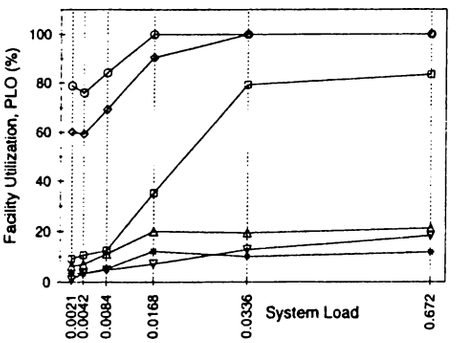

Figure 6.11: PLO, Facilities' Utilization

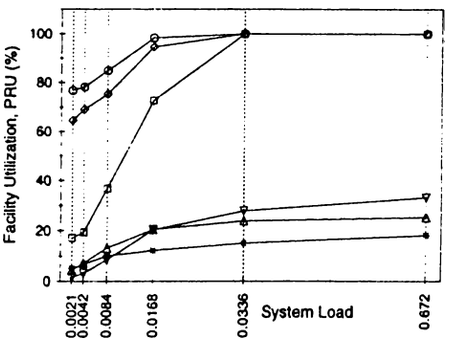

Figure 6.12: PRU, Facilities' Utilization

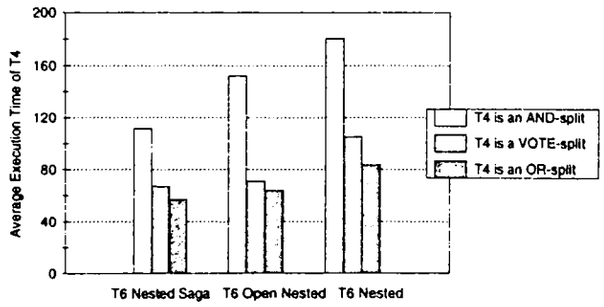

Figure 6.13: Transaction Models Performance

execution, causing an artificial reduction of the slack portion assigned to the currently executing activity. On the other hand, **TSL** assigns the whole slack to the currently executing activity allowing it to meet its deadline. It is important to assign larger slack portions to long-lived activities which are processed at the beginning in our workflow schema. Among the **PEX**, **PES** and **PLO** strategies, **PEX** is more suitable for our schema since it uses the *avg_exec* parameter in distribution of the *avl_sl* value. **PES** and **PLO** are incognizant to the difference in execution times and therefore, to the beneficial slack magnitude assigned to the activities. Moreover, these policies may overestimate or come short of the value of the distributed slack portions since they use the parameters of different measuring scale (i.e., execution time versus escalation cost and the length of facility queues). For example, under the **PLO** strategy activity T_{61} with the execution time of 0.042 time units is assigned a slack of 0.58 time units. The **PRU** strategy is more fair in assigning deadlines to activities. It distributes the available slack time according to the 'weights' of the activities. It allows the applications to be processed in a more natural way by letting them reside in the system for a longer time if the next required facility is busy with another application. With this policy, the submitted applications are allowed to wait for about 50% longer time until they are processed or aborted. As it was mentioned before, for the better performance under **PRU**, we pay by about 30% heavier facilities' loads.

Regardless of the deadline assignment strategy employed, we also examined the performance of transaction models. According to the list of models in Table 6.1, we varied the model settings for the activities T_4 and T_6. The results

are depicted in Figure 6.13. The best performance, i.e., the lowest average execution time, is achieved when T_6 is represented by the Nested Saga model and T_4 is an OR-split activity. The selection of the transaction model can also be based on the rules at a particular graduate school admission office. For example, it may be an obligatory requirement for T_4 to be realized as an AND-split transaction.

In summary, analysis of the graphs in Figures 6.1-6.13 lets us derive the following suggestions for our workflow schema:

- Optimal system throughput is $5 - 7$ applications per business day.
- The capacity of 'Advisor/Chaiman' and 'Staff Members' facilities should be extended. Figures 6.14-6.17 show the performance of the system under the **PRU** policy where the capacities of these facilities are doubled. With the increased facility capacities, the number of completed applications raises till 11 per business day. The overload of the facilities is also prevented. The value of facility utilization between 40% and 60% is recognized in [DKOS97] to be sufficient and not crucial. In comparison to the previous performance results of **PRU** (Figures 6.2, 6.4, 6.6), the better performance is obtained with the extension of the capacities of the facilities.
- A better performance is achieved if the activity T_4 is implemented as an OR-split activity and the activity T_6 is represented by the Nested Saga model.

6.3.2 Priority System Simulation

In the priority system simulation we assume that the execution order of the activities is determined by the deadlines assigned to them. This means that the activities may be executed in an order different than that in the non-priority system. As we mentioned in Section 5.2, for the priority system simulation we do not assign initial deadlines for the activities, instead they are determined

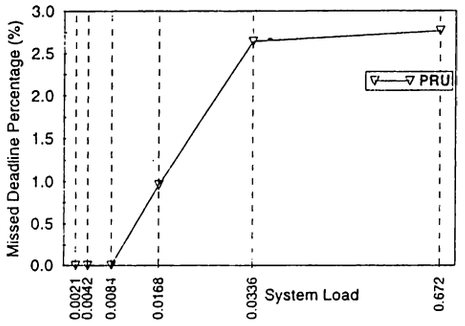

Figure 6.14: Missed Deadline Percentage

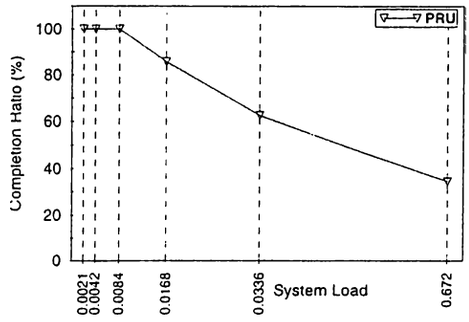

Figure 6.15: Completion Ratio

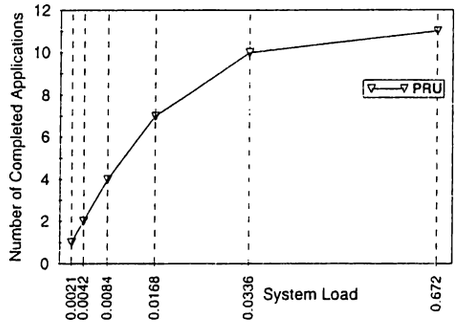

Figure 6.16: Number of Completed Applications

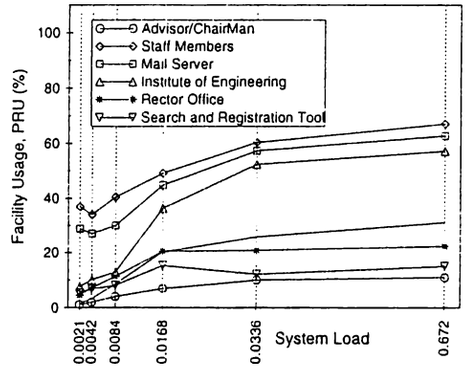

Figure 6.17: PRU, Facilities' Utilization

Activity	$q(T_k)$	$num\_ru(T_k)$	$num\_di(T_k)$	$avg\_exec(T_k)$
T_1	0.1	1	2	8.04
T_2	0.2	0	8	2.37
T_3	0.2	1	4	16.33
T_{31}	0.4	0	1	0.102
T_{32}	0.4	0	1	0.99
T_{33}	0.4	0	1	0.103
T_{34}	0.2	1	1	19.54
T_4	0.7	13	26	157.41
T_5	0.3	1	9	32.11
T_6	0.2	4	13	93.07
T_{61}	0.5	0	1	0.034
T_{62}	0.6	0	3	0.228
T_{63}	0.3	0	1	0.047
T_{64}	0.3	0	1	0.049
T_{65}	0.2	2	5	49.31
T_{66}	0.2	2	5	31.65
T_7	0.4	8	4	69.71
T_{71}	0.3	2	1	12.77
T_{72}	0.2	2	1	13.01
T_{73}	0.4	2	1	13.18
T_{74}	0.1	2	1	14.66
T_8	0.1	1	2	7.85
T_{81}	0.2	0	1	5.88
T_9	0.1	1	1	7.42
T_{10}	0.1	0	8	4.821
T_{11}	0.1	1	2	7.21

Table 6.3: Initial settings for priority system simulation

dynamically during the execution. Similar to the first set of experiments, we determined the *avg_exec* characteristics through 50 system runs. With different deadline assignment methods for the priority system simulation, we obtained slightly different values for the *avg_exec* parameter of each activity. Since the fluctuation of the *avg_exec* value did not exceed 7% for any activity, we calculated the average value of the *avg_exec* parameter for each activity and we assumed this parameter to be the same with all deadline assignment methods. The same assumption was made while investigating the transaction models' performance (Figure 6.28).

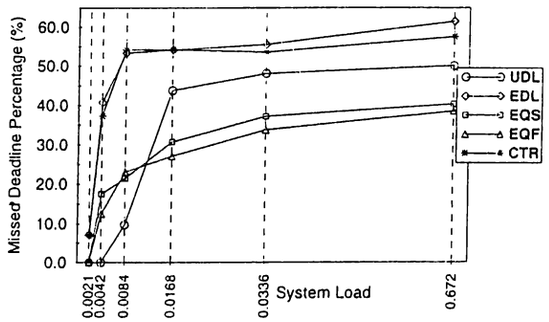

Figure 6.18: Missed Deadline Percentage

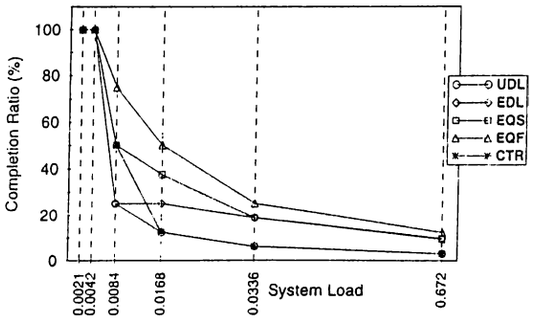

Figure 6.19: Completion Ratio

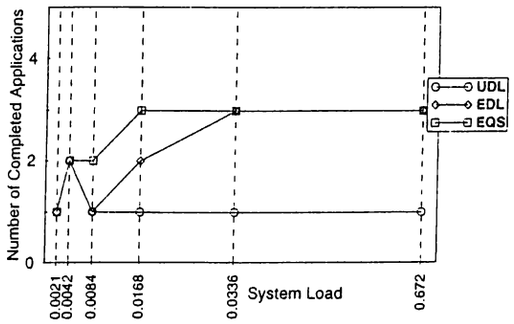

Figure 6.20: Number of Completed Applications

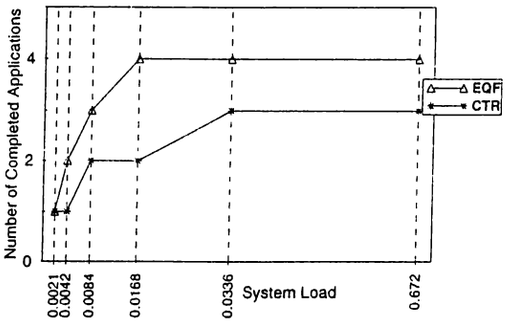

Figure 6.21: Number of Completed Applications

In general, the priority system simulation exhibited worse performance than that obtained for the non-priority system. We think that the performance decline is due to both the preemption mechanism used in this set of experiments and the non strict execution order of the activities.

Compared to the non-priority system performance results, missed deadline percentage values (Figure 6.18) increase about 300% under the worst performing policy and about 100% in the best case. The number of completed applications is reduced down by 42.9% which makes 4 completed applications per one business day[1] (Figure 6.21). The assumption that a failure of an activity does not lead to the failure of the entire process, but to reassigning of the deadline and reexecution of that activity, lets the number of completed applications not be reduced dramatically. In the absence of concurrency (i.e., when the system load is 0.0021 or 0.0042), all the strategies perform equally. The completion ratio is 100% for these system loads. Under higher loads, the difference between performance results of strategies starts to appear. The algorithms that make use of the arrival times of the activities assign/reassign the deadlines in a more efficient way than that with the other methods. When an activity misses its deadline, if the entire process deadline is not missed and there is still enough time for reexecution of the failed activity, the arrival time value of this activity is replaced by the current time value and then its deadline is recalculated. As a confirmation of the discussion above, we can see from Figure 6.18 that the missed deadline percentage values for **EQS** and **EQF**[2] are lower than that for the other algorithms. Algorithms **UDL** and **EDL** are incompetent of an efficient deadline adjustment. Under these policies we observe a chaining effect of missing deadlines. When an activity misses its deadline it indirectly leads to missing of the entire workflow's deadline. Under **UDL** and **EDL**, the system repeatedly reassigns the same deadline to the failing activity and reexecutes it but the slack portion allowed for this activity is not sufficient to meet its deadline. Furthermore, continuos reexecution causes consumption of nearly all the recourses of the involved facility as well as delaying of the arriving activities.

Analyzing the facility utilization graphs presented in Figures 6.22-6.26, we

[1] under the **EQF** policy.

[2] both use the arrival time parameter in slack distribution.

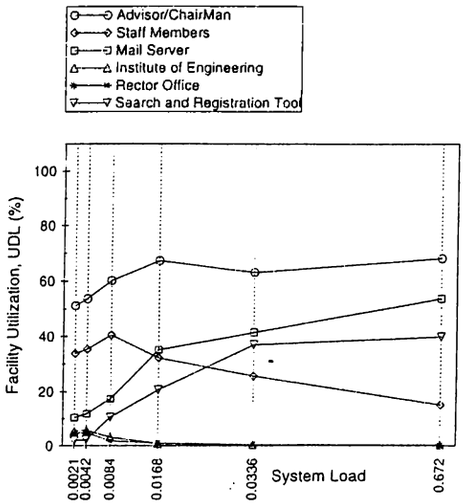

Figure 6.22: UDL, Facilities' Utilization

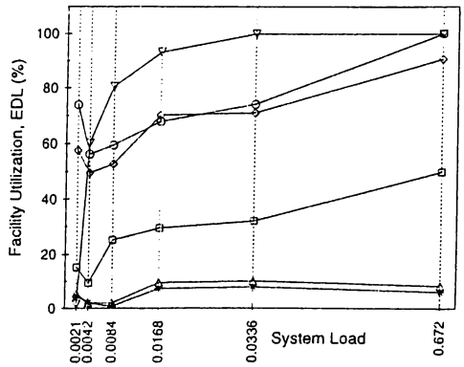

Figure 6.23: EDL, Facilities' Utilization

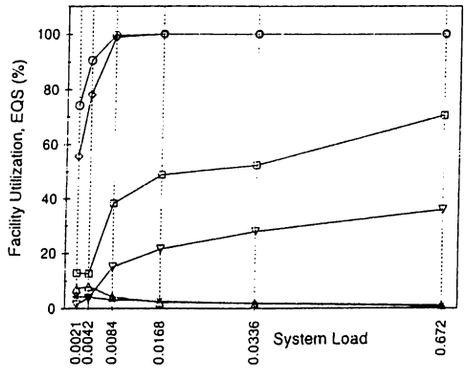

Figure 6.24: EQS, Facilities' Utilization

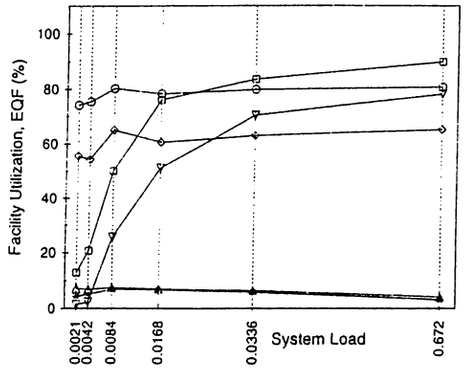

Figure 6.25: EQF, Facilities' Utilization

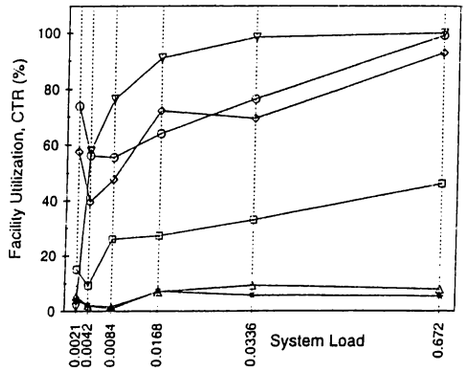

Figure 6.26: CTR, Facilities' Utilization

can differentiate the impact of the deadline assignment strategies more precisely. The **UDL** policy causes cascading aborts in the presence of concurrency. With this policy all the activities in a workflow are assigned the same deadline which is the deadline of the entire workflow. Therefore, none of the ensuing activities[3] can start execution until the current workflow is finished. This leads to the fact that the next workflow will most probably miss its deadline. The same situation occurs for all the subsequent applications submitted to the system. This behavior results in very low number of completed applications. Considering the facilities' utilization we note that the facilities that are involved at the early stages of the execution process (e.g, 'Mail Server', 'Search and Registration Tool') are uselessly heavily loaded; the utilization of those which serve latter activities approaches to 0. The **EDL** and **CTR** strategies turned out to be equal for our system settings. The corresponding formulas (Section 5.2.2) become equal since the contention ratio parameter (*cont_ratio*) used by the **CTR** strategy is neighbouring 1. This means that, $ar(T_k)$ and $avg\_exec(T_k)$ parameters in the formula of **CTR** are cancelled and the formula becomes the same as that of **EDL**. The performance of these algorithms in terms of the number of completed applications is better than that of **UDL**, but as we mentioned before, **EDL** and therefore **CTR** are not effective in deadline adjustment. Utilization of the facilities that serve longest and shortest activities ('Advisor/ChairMan', 'Staff Members' and 'Search and Registration Tool', respectively) raises greatly. To explain the reason for this situation we provide an example depicted in Figure 6.27.

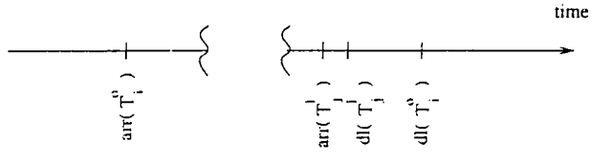

Figure 6.27: A Deadline Assignment Example

T_i^0 is a long-lived activity belonging to a currently executed workflow. T_j^1 is a short (traditional) activity belonging to another workflow which is processing an application submitted later. Since T_j^1 has an earlier deadline, it preempts T_i^0 leading to the reexecution of it later. Consequently, activity T_i^0, which

[3]comprising the next workflow.

is executed twice or more, consumes more recourses, and also since it is a long-lived activity, it delays execution of the activities requesting the same facility for a considerably long period of time. In the case of short activities' competition, which are assigned short deadlines and therefore have short slacks in reserve, it is not likely that once been interrupted these activities will meet their deadlines.

EQS and **EQF** both make use of the arrival time of the activities, thereby providing a better way for deadlines' adjustment. Nevertheless, **EQS**, assigning an equal portion of the currently available slack to an executing activity, does not differ long-lived and traditional activities. This policy leads to an unnecessary increase in the deadlines of short activities and also the slack portions assigned to long-lived activities are not sufficient. Tight deadlines result in a higher missed deadline percentage which in turn causes a more intensive utilization of the facilities serving long-lived activities. It can be seen from the figures that, the **EQF** algorithm demonstrates better performance results than the others. This is because, in this strategy, in conjunction with the arrival time of an activity, the average execution time characteristic is also used in the slack distribution.

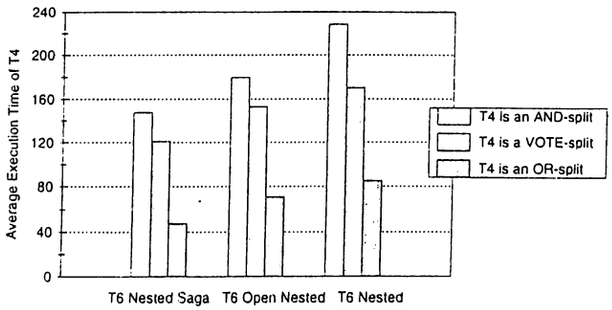

Figure 6.28: Transaction Models Performance

Considering the transaction models' performance, presented in Figure 6.28, we observe the same tendency of the graphs as that in Figure 6.13 for the non-priority system simulation. Again, the Nested Saga model provides better performance than others in terms of the elapsed time. As it was mentioned before, for the priority simulation the elapsed times are larger then those for the non-priority simulation. The interesting point in this figure is that the time

gap between the AND-split and OR-split representation of T_4 is significantly increased and the gap between AND-split and VOTE-split representation of T_4 is reduced. Thus, the benefit gained by switching from the AND-split to the VOTE-split model, which was substantial for the non-priority system, is less considerable for the priority system. The rationalization can be found in our assumption regarding a non-strict execution order and a possibility of preemption of the activities. The execution order is determined by the deadlines of the activities, therefore it is not guaranteed that the subactivities comprising T_4 and T_6 are executed one by one without interruptions.

As a concluding remark for the two studied system settings, we should note the following. For our workflow schema, when some activities are represented by dynamic structures such as AND-, VOTE- and/or OR-split transaction models, and the execution times of the activities vary from tenths of the time unit to the tens and hundreds of the time units, the deadlines are better to be estimated a priori as well as the execution order should be defined by the workflow schema designer. In other words, the non-priority assumptions and techniques are more appropriate for the dynamic class of workflows which our schema belongs to.

6.3.3 Simulation with Different Initial Settings

As we have mentioned in our earlier discussions, the weak performance of some of the deadline assignment strategies was caused by the significant differences in the lengths of the activities. In both sets of our simulation experiments, this fact prevented the methods which do not make use of execution times in assigning deadlines, from providing a better performance. In this section, we present the performance results obtained with different parameter settings. The values of the *avg_exec* parameter of activities are chosen in a way to reduce the differences in the execution times of long and short activities. Although these values are less realistic for our particular application than those used for the previous experiments, we run this experiment to confirm our earlier conclusions and predictions. With the new settings, in the non-priority system simulation we expect an improvement in the performance of **PEX** and **PES**

strategies, and in the priority system simulation an improvement for **EDL**, **CTR**, and **EQS** strategies.

Activity	$dl(T_k)$	$esc(T_k)$	$q(T_k)$	$num\_ru(T_k)$	$num\_di(T_k)$	$avg\_exec(T_k)$
T_1	8	5	0.1	1	2	6.037
T_2	3	2.2	0.2	0	8	1.146
T_3	9	5	0.2	1	4	6.51
T_{31}	1.5	0.1	0.4	0	1	1.47
T_{32}	1.5	0.1	0.4	0	1	1.38
T_{33}	1.5	0.1	0.4	0	1	0.97
T_{34}	7	3	0.2	1	1	6.67
T_4	39	7	0.7	13	26	24.81
T_5	11	7	0.3	1	9	9.84
T_6	21	10.5	0.2	4	13	16.24
T_{61}	0.75	0.2	0.5	0	1	0.52
T_{62}	2.5	0.6	0.6	0	3	1.98
T_{63}	0.75	0.2	0.3	0	1	0.47
T_{64}	0.75	0.2	0.3	0	1	0.62
T_{65}	13	5	0.2	2	5	11.69
T_{66}	4	5	0.2	2	5	3.8
T_7	11	10	0.4	8	4	7.91
T_{71}	6	3	0.3	2	1	5.21
T_{72}	6	3	0.2	2	1	4.12
T_{73}	6	3	0.4	2	1	3.23
T_{74}	6	3	0.1	2	1	4.01
T_8	15	15	0.1	1	2	6.642
T_{81}	5	2	0.2	0	1	3.381
T_9	10	15	0.1	1	1	6.427
T_{10}	3	4	0.1	0	8	4.821
T_{11}	8	5	0.1	1	2	7.21

Table 6.4: Initial settings for non-priority system simulation

Non-priority System Simulation

In this experiment, the values of the *avg_exec* parameter (Table 6.4) are of the same order of magnitude except for only a few activities.

As displayed in Figures 6.29-6.30, we observe considerable changes in the performance results for the **PEX** and **PES** strategies. While the results for

the **PES** strategy agree with our hypothesis, **PEX** shows the opposite tendency. Under this policy with the new settings, long-lived activities are assigned smaller portions of the available slack time. The reason we think is the fact that the extracted portions of the *avl_sl* parameter corresponding to the remaining activities, increase since the execution time values of these activities

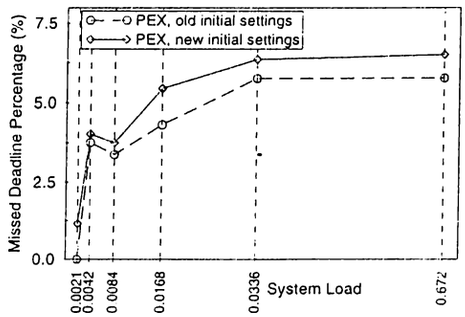

Figure 6.29: Missed Deadline Percentage

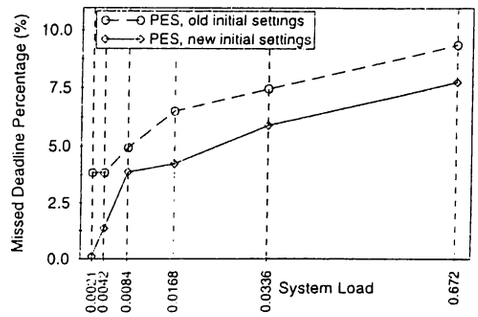

Figure 6.30: Missed Deadline Percentage

are increased relative to the *avg_exec* values of the long-lived activities. To explain the performance of the **PES** method, we first note that the set of the *avg_exec* values become more homogeneous and 'closer' to the values of the escalation cost (*esc*) parameter. By 'closer' mean that the corresponding values of *avg_exec* and *esc* parameters have a correlation. As a consequence, with the new settings, **PES** behaves similar to **PEX**. It is seen from the figures that for some values of the system load **PES** performs even better than **PEX**. This is because **PES** still[4] preserves a more clear distinction between long-lived and short activities since we did not change the values of the *esc* parameter. The other deadline assignment methods are insensitive to the changes in the values of the execution time parameter and we did not observe sizeable changes in their performance results.

Priority System Simulation

Our new parameter settings for the priority system simulation are provided in Table 6.5. The observation we made regarding the *avg_exec* values in the previous section is also valid for this set of *avg_exec* parameter values.

[4]with the new settings.

Activity	$q(T_k)$	$num\_ru(T_k)$	$num\_di(T_k)$	$avg\_exec(T_k)$
T_1	0.1	1	2	8.04
T_2	0.2	0	8	2.37
T_3	0.2	1	4	10.23
T_{31}	0.4	0	1	1.08
T_{32}	0.4	0	1	1.31
T_{33}	0.4	0	1	1.27
T_{34}	0.2	1	1	11.41
T_4	0.7	13	26	39.15
T_5	0.3	1	9	13.25
T_6	0.2	4	13	24.08
T_{61}	0.5	0	1	0.81
T_{62}	0.6	0	3	2.17
T_{63}	0.3	0	1	0.53
T_{64}	0.3	0	1	0.71
T_{65}	0.2	2	5	15.84
T_{66}	0.2	2	5	7.21
T_7	0.4	8	4	13.43
T_{71}	0.3	2	1	5.21
T_{72}	0.2	2	1	4.58
T_{73}	0.4	2	1	6.21
T_{74}	0.1	2	1	6.45
T_8	0.1	1	2	7.85
T_{81}	0.2	0	1	5.88
T_9	0.1	1	1	7.42
T_{10}	0.1	0	8	6.15
T_{11}	0.1	1	2	9.03

Table 6.5: Initial settings for priority system simulation

Our prediction about the behavior of **EDL** and **CTR** with the new parameter settings was not justified. We expected to observe an increased performance of these methods since we made an adjustment in the average execution time values to reduce the difference among them. The results obtained with the new settings are presented in Figure 6.31. As it is seen from the figure, when the system is not heavily loaded the performance results of **EDL** and **CTR** are slightly improved. Nevertheless, the increase in the system load returns the values of missed deadline percentage up to those obtained in the previous set of experiments. Therefore, the inefficiency of **EDL** and **CTR** cannot be prevented by the change of the *avg_exec* values. These methods still preserve the

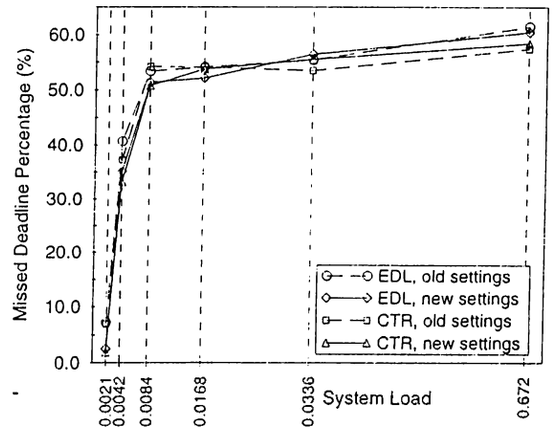

Figure 6.31: Missed Deadline Percentage

drawbacks discussed earlier in Section 6.3.2. The **UDL** policy which does not use any information about the activities (including *avg_exec* parameter values) did not react to the changes in the parameter settings in terms of system performance. Algorithms **EQS** and **EQF** exhibited very close performance results (Figure 6.32). The new values obtained for these methods are in between the results of **EQF** and **EQS** in the previous experiments. This means that the performance of **EQF** decreased and the performance of **EQS** increased. For the **EQF** method the degradation in performance is due to the same reason as that for the **PEX** method discussed in the previous section. **EQS**, by dis-

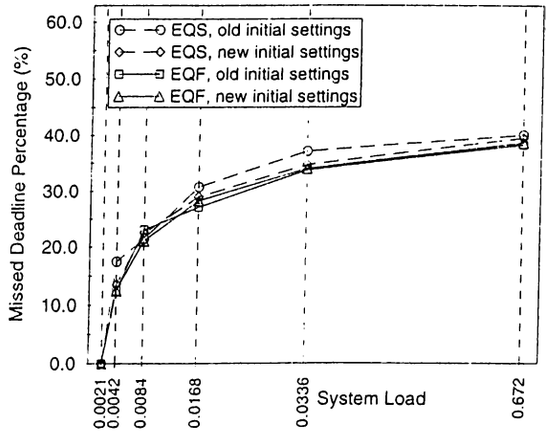

Figure 6.32: Missed Deadline Percentage

tributing the slack time evenly among all the activities, is expected to provide better results with more or less similar execution time values. We observe

confirmation of this prediction in Figure 6.32. Having chosen the parameter settings listed in Table 6.5, we 'tuned' the **EQS** method to provide better performance results for our schema. Thus, the observed higher performance of it is an unstable effect. Whereas, for the **EQF** method, for which the distribution of the available slack time based on *avg_exec* is envisaged in its formula, the performance results are less dependent on the parameter settings and this method is expected to provide steadily better performance than **EQS**.

6.3.4 Summary

In this chapter we studied several deadline assignment strategies for non-prioritized and prioritized perception of our system. The two sets of experiments showed considerably different performance results in terms of missed deadline percentage, completion ratio, and number of completed applications with different system load values. As our experiments show, the performance results obtained with the non-prioritized system are much better than those with the prioritized system. We conducted additional experiments for both of the approaches with different initial parameter settings to see how the results are affected. The results we obtained with the new settings did not show substantial changes in the performance. Thus, in the final analysis, we confirm the conclusions we made earlier in this chapter. For our workflow schema which comprises long-lived and short activities and contains dynamic structures such as AND-, OR-, and VOTE-split points, the non-priority approach provides more appropriate functionality of the system. We do not generalize these results over all workflows since we studied only a single application and moreover we think that there cannot be a common conclusion for all possible workflow applications. Summarizing this discussion, we claim that for each workflow application the decision regarding the most suitable implementation in terms of the prioritized/non-prioritized realization should be taken after studying both these approaches in conjunction with different deadline assignment strategies.

Chapter 7

CONCLUSION

In this thesis we consider several aspects of workflow management. Our study contributes to the theory of the workflow management as well as to the practical issues of workflow management systems (WFMSs) implementation.

We started our work with an observation of functionality of a WFMS suggested by the workflow management coalition (WFMC). In our opinion, a deficiency of WFMSs in the scope of the WFMC definitions is the absence of simulation tools which should provide a preliminary picture of future workflow performance. We understand that for currently existing commercial WFMSs which were produced a few years ago, there is a little ground for conducting simulation experiments. These WFMSs (excluding a few) do not support transactional representation of workflows. Therefore, a workflow in the frame of such WFMS is a static representation of a business process which allows a workflow designer to define only inter-activity dependencies. However, to improve an execution process and give a workflow more stable chances to be still adequate after a few years by considering scalability, security, and concurrency issues, workflow management should borrow and adopt database concepts to WFMSs implementation. In our work we select a set of transaction models which can function as a projection of a workflow schema onto the database domain. As a further extension of the simulation ground we suggest to probe workflows with prioritized and non-prioritized perceptions.

Also in this thesis, we discussed the aspects of human invocation and utilization of legacy tools during a workflow life cycle. We contributed with a combined workflow task structure which allows to control the execution of a legacy application and monitor human invocation based on the database technology. Also, we complemented the notion of the deadline provided by the WFMC with another representation which turned out to be more appropriate for our workflow application.

A practical contribution of our study is the workflow schema we developed and analyzed through simulation experiments. Our application which models an admission procedure at a graduate school, allowed us to apply and examine the performance of different transaction models and system settings. Adopting different deadline assignment and adjustment techniques, we proposed a new deadline assignment method based on the number of data items accessed and recourse units requested by an activity. This method provided good performance results for our workflow schema. Referring back to the theoretical contributions, we should add that while designing our schema we felt an insufficiency in the set of dynamic structures allowed for a workflow schema definition. At this stage we contributed with an additional definition of split points which we named VOTE-split point and corresponding VOTE-split transaction model.

Based on our conclusions derived from the simulation results a designer of a similar application could draw suggestions in terms of optimal system throughput, recommended capacity of facilities, and possible transaction model and control structures for the better system performance.

As a future prospect, to make the simulation system transparent and user-friendly, we present a sketch of a graphical user interface (GUI) for it which would hide the mathematical complexity from workflow designers (Figures 7.1-7.3). As we stated earlier, such a tool could be incorporated in a WFMS and involved after the design stage of workflow schemas. The schema which is seen in Figure 7.1 thus should be specified at an earlier stage. Then this schema should be processed and simulated using chosen settings. The results of the simulation might be represented in table and graphical forms similar to those provided in Figure 7.3.

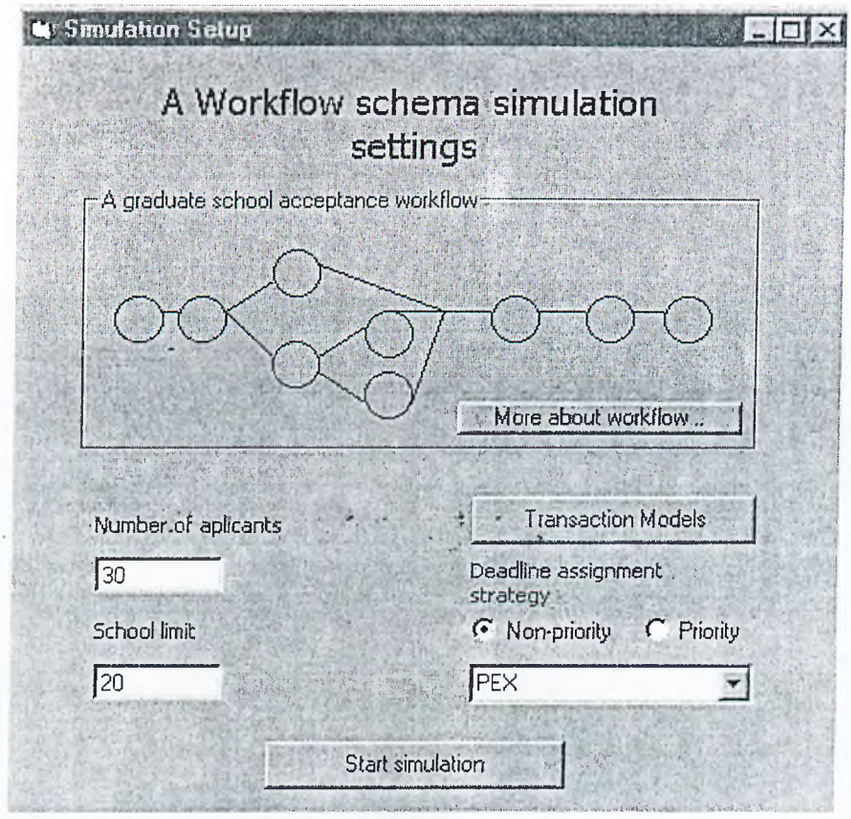

Figure 7.1: GUI, Simulation Setup

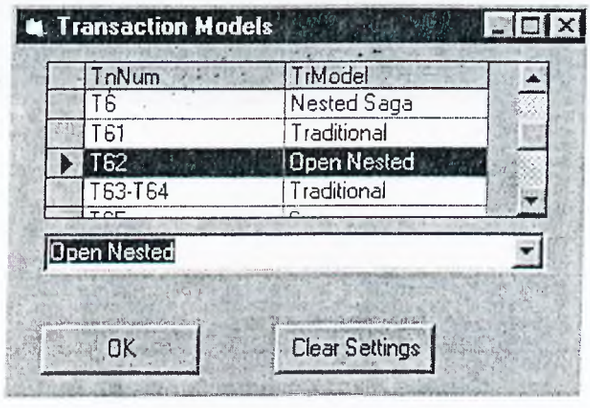

Figure 7.2: GUI, Transaction Models Selection

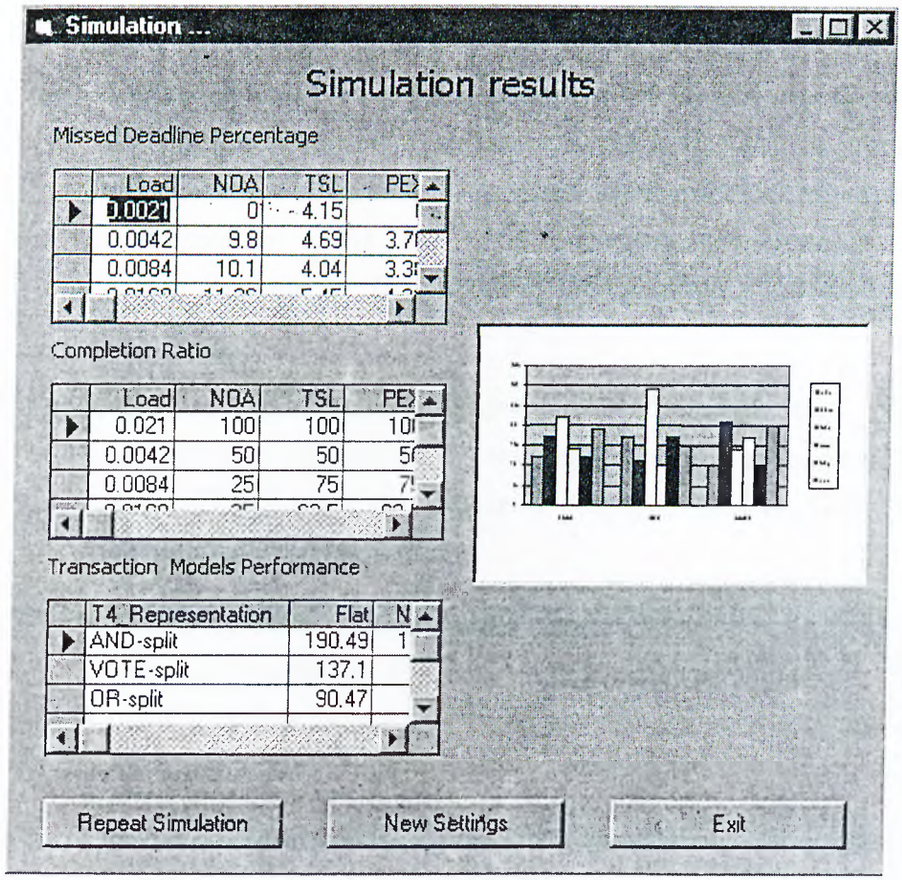

Figure 7.3: GUI, Simulation Results

Bibliography

[AAA+95] Gustavo Alonso, Dharma Agrawal, Abd-El Abbadi, C. Mohan, R. Gonthur, and M. Kamath. Exotica/FMQM: A persistent message-based architecture for distributed workflow management. In *Proceedings of the IFIP WG8.1 Working Conference on Information Systems for Decentralized Organizations*, Trondheim, August 1995.

[AAAM98] Gustavo Alonso, Dharma Agrawal, Abd-El Abbadi, and C. Mohan. Functionality and limitations of current workflow management systems. IEEE-Expert (to appear in a special issue on Cooperative Information Systems), 1998.

[Alo98] Gustavo Alonso. The future of workflow management systems. In *Proceedings of the 1st International Symposium on Advanced Database Support for Workflow Management*, Enschede, the Netherlands, May 1998.

[Amb96] Michael Amberg. Modeling adaptive workflows in distributed environments. In *Proceedings of the First International Conference on Practical Aspects of Knowledge Management*, Basel, Switzerland, October 1996.

[AS96] Gustavo Alonso and Hans-Jörg Schek. Database technology in workflow managenment. *Journal of the Swiss Computer Society*, (1), 1996.

[BMM96] Perakath Benjamin, Charles Marshall, and Richard Mayer. A workflow analysis and design environment - WADE. Technical report, Knowledge Based Systems, Inc., 1996.

[Bob96] Angelo Bobak. *Distributed and Multi-database Systems.* Artech House, Boston, 1996.

[CA95] Balaji Chakravarthy and El Anwar. Exploating active database paradigm for supportiing flexible transaction model. Technical Report UF-CIS-TR-95-026, University of Florida, CIS Department, April 1995.

[CCPP95] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Conceptual modeling of workflows. In *Proceedings of the 14th Object-Oriented and Entity-Relationship Approach International Conference*, Gold Coast, Australia, 1995.

[CHRW98] Andrzej Chichoki, Abdelsalam Helal, Marek Ruzinkiewicz, and Darell Woelk. *Workflow and Process Automation. Concepts and Technology.* Kluwer Academic Publishers, Boston, Dordrecht, London, 1998.

[CR94] Panos K. Chrysantis and Krithi Ramamrithan. Synthesis of extended transaction models using ACTA. *ACM Transactions on Database Systems*, (3):450–491, 1994.

[DKOS97] Asuman Doğaç, Leonid Kalinichenko, M. Tamer Özsu, and Amit Sheth. *Advances in Workflow Management Systems and Interoperability.* NATO Advanced Study Institute, Turkey, 1997.

[ELLR90] Ahmed Elmagarmid, Y. Leu, W. Litwin, and Marek Rusinkiewicz. A multidatabase transaction model for Interbase. In *Proceedings of the Conference on Very Large Data Bases*, Brisbane, Australia, August 1990.

[Elm92] Ahmed Elmagarmid. *Database transaction models for advanced applications.* M. Kaufmann Publishers, San Mateo, California, 1992.

[Fis95] Paul Fishwick. *Sim++ Reference manual.* University of Florida, CSE Department, 1995.

[GHKM94] Dimitros Georgakopoulos, Mark Hornik, P. Krychniak, and F. Manola. Specification and management of extended transactions in a programable transactional environment. In *Proceedings*

of the 10th International Conference on Data Engineering, pages 462–473, Huston, Texas, February 1994.

[GHS95] Dimitros Georgapoulos, Mark Hornick, and Amit Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, 1995.

[GMS87] Hector Garcia-Molina and Kim Salem. Sagas. In *Proceedings of the Conference on Database Systems in Office, Technique and Science*, pages 249–259, May 1987.

[Int] International symposium on advanced database support for workflow management http://wwwis.cs.utwente.nl:8080/ wide/sympo/.

[JK97] Sushil Jajodia and Larry Kerschberg. *Advanced Transaction Models and Architectures.* Kluwer Academic Publishers, 1997.

[Joo96] Stef Joosten. Workflow management research area overview. Second American Conference on Information Systems, Tutorial, 1996.

[KGM93] Benjamin Kao and Hector Garsia-Molina. Deadline assignment in a distributed soft real-time system. In *Proceedings of the 13th IEEE International Conference on Distributed Computing Systems*, 1993.

[KGM94] Benjamin Kao and Hector Garsia-Molina. Subtask deadline assignment for complex distributed soft real-time tasks. In *Proceedings of the 14th IEEE International Conference on Distributed Computing Systems*, 1994.

[Kim95] Won Kim. *Modern database systems: the object model, interoperability, and beyond.* Addison-Wesley, New York, 1995.

[KP92] Gail E. Kaiser and Calton Pu. *Dynamic Restructuring of Transactions*, chapter in [Elm92]. M. Kaufmann Publishers, 1992.

[KR95] Mohan Kamath and Krithi Ramamrithan. Modeling, correctness and system issues in supporting advanced database applications using workflow management systems. Technical Report CS-TR-95-50, University of Massachusetts, CS Department, 1995.

[KS95] Narayan Krishnakumar and Amit Sheth. Managing heterogeneous multi-system tasks to support enterprise-wide operations. *Distributed and Parallel Databases*, 3(2):155–186, 1995.

[LHK97] K. Lam, L. S. Hung, and Benjamin Kao. Impact of priority assignment on optimistic concurrency control in distributed real-time databases. *IEE Proceedings - Computers and Digital Techniques*, 144(5), September 1997.

[Moh97] C. Mohan. *Recent Trends in Workflow management Products Standards and Research*, chapter in [DKOS97]. NATO Advanced Study Institute, 1997.

[PKH88] C. Pu, E. Kaiser, and N. Hutchinson. Split transactions for open-ended activities. In *Proceedings of 14th International Conference on Very Large Data Bases*, Los Angeles, California, USA, August 1988.

[PR97] Euthimios Panagos and Michael Rabinovich. *Reducing Escalation-Related Cost in WFMSs*, chapter in [DKOS97]. NATO Advanced Study Institute, 1997.

[PR98a] Euthimios Panagos and Michael Rabinovich. Managing activities deadlines in WFMSs. ATT Labs - Research, 1998.

[PR98b] Euthimios Panagos and Michael Rabinovich. Reducing escalation-related cost in WFMSs. ATT Labs - Research, 1998.

[SK97] Amit Sheth and Krys Kochut. *Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems*, chapter in [DKOS97]. NATO Advanced Study Institute, 1997.

[SST94] Rajendran Sivasankaran, John Stankovic, and Don Towsley. Priority assignment in real-time active databases. In *Proceedings of the Parallel and Distributed Information Systems*, 1994.

[WAN97] Jürden Wäsch, Karl Aberer, and Erich J. Neuhold. *Transactional Support for Cooperative Applications*, chapter in [DKOS97]. NATO Advanced Study Institute, 1997.

[Wid] WIDE project home page. http://dis.sema.es/projects/WIDE/.

[Wor96] Workflow Management Coalition. *The Workflow Management Coalition Specification. Workflow Management Coalition Terminology and Glossary, WFMC-TC-1011*, June 1996.