CORBA-Based Run-Time Architectures for Workflow Management Systems

J.A. Miller, A.P. Sheth, K.J. Kochut and X. Wang
Large Scale Distributed Information Systems Lab
Department of Computer Science
The University of Georgia
Athens, GA 30602-7404
email: <jam, amit, kochut>@cs.uga.edu
URL: *http://www.cs.uga.edu/LSDIS*

ABSTRACT

This paper presents five run-time architectures for implementing a Workflow Management System (WFMS). The architectures range from highly centralized to fully distributed. Two of the architectures have been implemented at the Large Scale Distributed Information Systems (LSDIS) Lab at The University of Georgia. All the WFMS architectures are designed on top of a Common Object Request Broker Architecture (CORBA) implementation. The paper also discusses the advantages and disadvantages of the architectures and the suitability of CORBA as a communication infrastructure. A minor extension to CORBA's Interface Definition Language (IDL) is proposed to provide an alternative means of specifying workflows. Simplified examples from the healthcare domain are given to illustrate our technology. [1]

1 INTRODUCTION

Competition and economic pressures force modern business corporations to look for new information technologies to support their business process management. Since workflow technology provides a model for business processes, and "a foundation on which to build solutions supporting the execution and management of business processes" [HK95], it has been receiving much attention in the past few years. A workflow can be simply defined as a set of tasks (also called activities or steps) that cooperate to implement a business process. A good workflow technology can also provide a way to to make good use of past investments by allowing integration of legacy systems, and the flexibility to support significant organizational changes and technology evolutions associated with today's dynamic enterprises.

A workflow model can be used to design automated or semi-automated solutions for certain business processes within an enterprise and across multiple enterprises. Workflow models tend to

[1]This research was partially done under a cooperative agreement between the National Institute of Standards and Technology Advanced Technology Program (under the HIIT contract, number 70NANB5H1011) and the Healthcare Open Systems and Trials, Inc. consortium. See URL: *http://www.scra.org/hiit.html*.

be more computer-oriented than traditional business process models. Consequently, they better facilitate automatic generation of substantial portions of actual solutions (i.e., executable workflows).

The history of workflow technology dates back to office automation and batch processing in the late 1970's, with the first use of the term in early 1980's [Smi93]. In recent years, workflow technology has gained in popularity due to the trend of business process reengineering and many emerging related technologies such as middleware and object-oriented technology, which make the development of a realistic workflow management system possible. A technical overview on workflow management appears in [GHS95] and tutorial materials available include [Rei94, She95]. After several years of development, many workflow products and prototypes are now available [WF94, GHS95], and early empirical studies in applying the workflow technology or using the products are also being reported [JAD$^+$94, J$^+$95].

A workflow is composed of multiple *tasks* (also called steps or activities). There are two types of tasks – simple tasks which represent individual indivisible activities, and compound tasks which represent some activities which can be divided into sub-activities (simple tasks or even other compound tasks). An entire workflow can be regarded as a large compound task. A simple task may be a program which can run on *processing entities*, which include application systems, servers supported by client-server systems or Transaction Processing Monitors (TP-Monitors), Database Management Systems (DBMSs), etc. Tasks are operations or a sequence of operations that are submitted for execution at the processing entities using their interfaces.

A *WorkFlow Management System* (WFMS) provides "the ability to specify, execute, report on, and dynamically control workflows involving multiple humans and HAD (Heterogeneous, Autonomous, and Distributed) systems" [KS95]. For workflow execution, a *workflow scheduler* is necessary to enforce inter-task dependencies, and therefore, to coordinate the execution of tasks in the workflow. Also, *task managers* are designed to start tasks and to perform a supervisory role in forward recovery.

Most workflow related efforts done in the past few years can be categorized into workflow specification and design, inter-task dependency and scheduling studies, and workflow management system design.

Many papers have been published on workflow modeling, workflow specification, and workflow design [HK95, SR93, KS95]. Krishnakumar and Sheth [KS95] described the modeling and specification of workflows. Forst et al. [FKB95] proposed a language called C&CO which is an extension of C to specify workflows. Duitshof performed an empirical study with emphasis on workflow design methods in three different organizations [Dui94]. Using the term *transactional workflows*, use of transaction concepts in workflow management was introduced in [SR93] and subsequently discussed in several papers, including [BDSS93, GH94, RS95b, GHS95, M$^+$95, TV95]. However, more work needs to be done in this area.

The specification and enforcement of intertask dependencies started with the early efforts to try to specify transaction structure and behavior [Kle91, CR90, DHL91]. Klein proposed two primitives to describe conditional existence dependency [Kle91]. The task state transition diagram introduced is still a popular way to describe task structures. A variety of database operation related dependencies (e.g., *commit dependency* and *abort dependency*) have also been defined in [CR92, ANRS92, GH94].

Attie et al. [ASSR93] formalized inter-task dependencies by using the Computation Tree Logic (CTL) language [Eme90], introduced a way to synthesize an automaton that captures the computations satisfying the given dependency, and designed a centralized scheduler. Tang and Veijalainen [TV95] extended the work in [ASSR93] and proposed a protocol to enforce inter-task dependencies. Several scheduling approaches have been reviewed in [RS95b].

Since workflow management system design depends on workflow specification, research on this aspect started a little later than that of workflow specification. Breitbart et al. [BDSS93] proposed an integrated architecture to support the properties of transactions and their workflow models. The centralized architecture of FlowMark is discussed in [LA94]. Alonso et al. extended the centralized FlowMark architecture to a distributed architecture of a workflow management system [AAA+95]. ObjectFlow's architecture is discussed in [HK95]. Kappel et al. presented a WFMS architecture which integrates the rules into an object-oriented model [KRR95]. Barbara, Mehrotra and Rusinkiewicz proposed an implementation of a new WFMS model based on the INformation CArrier (INCA) [BMR94]. INCAs carry data as well as scheduling/routing information and travel from processing station to processing station. This approach facilitates highly flexible, fully distributed and dynamic workflows, possibly at the sacrifice of efficiency. Issues of low-level and infrastructure support for building workflow management systems have been discussed in [GHS95, RS95a]. So far, we have not seen empirical studies of different workflow system architectures in published literature. An early effort is reported in the thesis by Wang [Wan95]. Workflow reliability and recovery is still a new area where few people have ventured. Discussions on recovery and use of compensation appear in [RS95b, KS95]. Failure handling in large scale workflow management systems has been discussed in [AKA+95].

Following the advocacy for distributed object management based infrastructure for workflow management systems in [GHS95], this paper discusses the use of specific services provided by Post-Modern Computing's ORBeline, a CORBA-based product (we also gained additional experience in using CORBA by using IONA's ORBIX and by participating in a beta test of SUN's DOE). As a part of our comprehensive investigation into alternative workflow management system architectures, we have designed five run-time architectures. In this paper, we introduce these architectures and specifically comment on the use of ORB services to support these architectures. Task structures play an important role in our workflow model to support heterogeneous non-transactional and transactional tasks. With the aim of easily developing run-time executable code from a high level workflow specifications, we propose the Workflow Interface Definition Language (WIDL) that involves a simple extension to CORBA's IDL.

In section 2 of this paper, we discuss the CORBA communication infrastructure and its suitability for workflow. Section 3 briefly describes how workflows are modeled/specified. In section 4, we present the five WFMS run-time architectures. Task structures and task models are presented in section 5. Finally, section 6 summaries the advantages of CORBA and highlights useful capabilities found in CORBA 2.0.

In the HIIT project, as a follow-on effort to developing the workflow automation technology (one aspect of which is described in this paper), we are applying and evaluating the technology through application partnership with the Connecticut Healthcare Research and Education Foundation in the area of healthcare delivery in a managed care context (see *http://www.cs.uga.edu/LSDIS/workflow* for details). Our goal is to develop and demonstrate the use of appropriate workflow technology that (a) enables additional functionality in healthcare delivery systems through better coordination of tasks within and across enterprises, and (b) supports healthcare delivery by utilizing the resources (e.g., doctors, nurses, technicians, operating rooms, hospital rooms, lab tests, etc.) more efficiently, thus leading to higher quality patient care in less time and at a lower cost to the patient and the healthcare system.

2 CORBA COMMUNICATION INFRASTRUCTURE

At present, most systems in an enterprise are connected together with networks. However, to build a workflow management system that supports the integration and interoperability between Heterogeneous, Autonomous, and Distributed systems, a communication mechanism operating at a higher-level then Sockets or Remote Procedure Calls (RPC) would be beneficial. Distributed Object Management (DOM) supports this kind of integration and interoperability. OMG (Object Management Group)'s CORBA (Common Object Request Broker Architecture) [OMG93] is a rapidly maturing standard for DOM. The CORBA specification defines the architecture of an Object Request Broker (ORB), whose job is to enable and regulate interoperability between objects and applications. CORBA is seen as the only viable technology truly headed in a cross-platform, non-proprietary direction. CORBA matches the rigorous demands of workflow systems in today's widely distributed, rapidly evolving and unpredictably fluctuating enterprises. Since CORBA is maturing rapidly both as a technology and as a standard, it was chosen as the communication infrastructure for our workflow project.

The CORBA 1.0 specification was released in October 1991. It was followed by CORBA 1.1 in March 1992 and CORBA 1.2 in December 1993. CORBA 2.0 was announced in November 1994

[OMG93, Bet95] and its specification was published in July 1995 [OMG95a, OMG95b]. There are already almost a dozen commercial ORBs or CORBA-like products available in the market (many implementing CORBA 1.2 and a few implementing CORBA 2.0). Table 2 lists some of them.

Product Name	Company	Mapping Languages
DOE (NEO)	Sun Microsystems	C++, C
ORBeline	PostModern Computing Technologies, Inc.	C++
Orbix	IONA Technologies	C++
ObjectBroker	Digital Equipment Corporation	C++
SOM (DSOM)	IBM	C
HyperDesk	HyperDesk Corporation	C++
ORBplus	Hewlett Packard	SmallTalk
XShell	Expertsoft Corporation	C++

Table 1: Commercial ORBs

CORBA provides a clean, high-level approach to writing distributed applications. Client programs communicate with servers (object implementations). This is usually done by specifying an interface between the client and the server. For greater flexibility, the interface is specified in a relatively simple definition language called the Interface Definition Language (IDL). This allows an object implementation to be changed without affecting any clients. Bindings between IDL and commonly used programming languages (e.g., C, C++, SmallTalk) allow mixing and matching (programmer's choice) of languages [OMG93].

Client programs bind to object implementations in order to make requests to be serviced. The binding returns a reference to the object implementation. Using the object reference, clients make requests in the form of remote operations (or method calls). Similar to ordinary method calls in C++, clients may pass parameters to the method and receive values back from the object implementation. Typically, method calls are synchronous (clients wait for methods to return); however, if the keyword `oneway` is used and values are not to be returned, the call can be asynchronous.

Remote method calls are implemented in the following manner. A client calls a client stub which marshals parameters into a message that is sent to the appropriate object implementation (host machine and process) by the ORB. At the remote machine, the message is converted into a method call on the IDL skeleton which then calls the actual user written code implementing the method. After the remote method finishes, control and output values are returned to the client. Only the client and the actual method implementation need to be coded, since the rest of the code is generated automatically by the IDL compiler.

```
return_value = object_reference->operation (param1, param2, ...);
```

The signatures for the operations (or methods) are given in an interface definition (see Figure 8). These definitions are much like a C++ class definition in that several methods are defined as a package and the inheritance mechanism can be used. Unlike C++ class definitions, private data representing the state of an object is not defined in the interface, but rather in the object implementation. This increases the independence between clients and servers. It is, however, possible to specify attributes in interfaces. This does not violate independence, since the attributes are implemented as functions returning/setting values within an object's state.

3 WORKFLOW MODELING

Workflows may be modeled or specified using a graphical model or using a workflow specification language. Because of the multitude of factors involved in designing efficient workflows for healthcare delivery and managed care, high level modeling becomes very useful. Using our Graphical Workflow Designer [Mur95], a workflow can be readily designed by an application domain expert who may not be a technology expert.

Let us consider the following simplified example in which a patient comes to the Emergency Room (ER) of a hospital with a specific complaint (see *http://www.cs.uga.edu/LSDIS/workflow* for more detailed examples). The patient first registers with the receptionist, who then assigns the patient to an examining room in the emergency wing. Once an ER doctor becomes available, he/she examines the patient and comes up with an initial diagnosis. The initial diagnosis may call for lab tests (x-Rays or a biopsy) which will then be analyzed, or it may simply lead directly to a final diagnosis. The doctor at this time determines if the patient should be treated on an in-patient or outpatient basis. After in-patient treatment, the patient may continue treatment as an outpatient or terminate treatment. If the out-patient's progress is not satisfactory, then the patient may return to the hospital for another round of diagnosis; otherwise, they may terminate treatment. Specific treatment plans (both in-patient and outpatient) would in practice expand into their own subworkflows. Figures 1 and 2 are workflow models produced with our Graphical Workflow Designer depicting this example (the figures were adapted from [HK95]).

Our Workflow Management System (WFMS) utilizes a simple model repository from which workflows may be selected for execution (we plan to make our repository more sophisticated in future implementations). Workflows or components of workflows may be added to the model repository by specifying them in a workflow language (e.g., WFSL/TSL [KS95] or WIDL (see section 5)) or by designing them with our Graphical Workflow Designer [Mur95]. The WorkFlow Specification Language (WFSL) [KS95] is a declarative rule-based language to describe the conceptual workflow specification, while the Task Specification Language (TSL) [KS95] is a language to specify simple

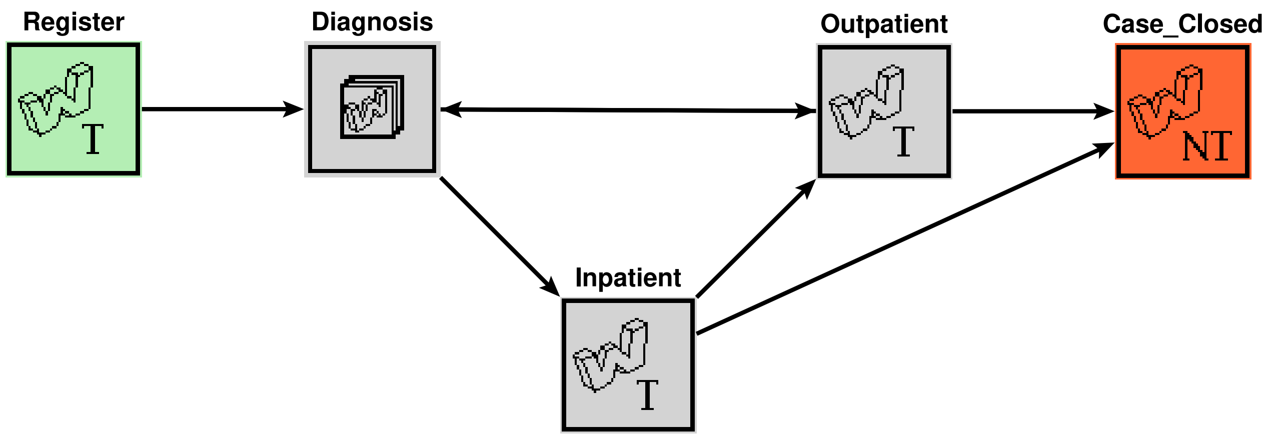

Figure 1: Patient Workflow Model

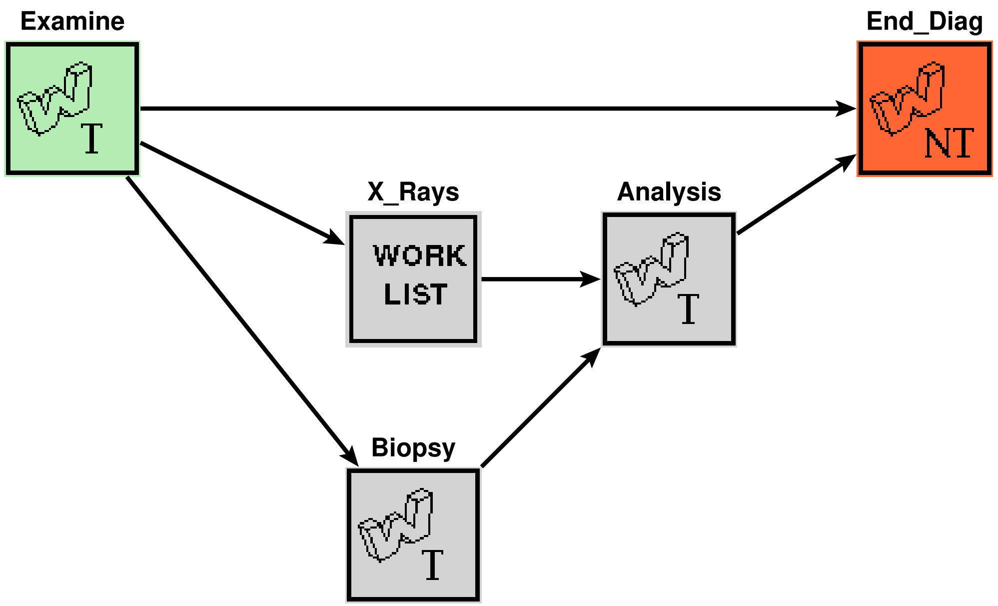

Figure 2: Diagnosis Subworkflow Model

tasks that run in a HAD information systems environment. Once a workflow is selected from the repository, several translation steps are carried out that instantiate a workflow instance that is able to run within the HAD execution environment (some manual coding/recoding may be necessary).

4 RUN-TIME ARCHITECTURES FOR A WFMS

The main components in the execution environment (or run-time system) are the Workflow Scheduler, Task Managers (TMs) and Tasks. Tasks are the run-time instances of an enterprise's applications. Today they typically run independently or are tied together in ad hoc ways. WFMSs tie these tasks together in a loosely coupled fashion. This is achieved by making minor modifications to existing application code or enforcing standards for new application development. The modifications provide hooks into the task that allow the transitions between major steps to be observed and in some cases controlled by the task manager for the task. To establish global control as well as facilitate recovery and monitoring, the task managers communicate with a scheduler. It is possible for the scheduler to be either centralized or distributed, or even some hybrid between the two [Wan95]. We discuss the five architectures in the subsections below. A brief comparison of the architectures is given in the conclusions.

4.1 Highly Centralized Architecture

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

4.2 Synchronous Centralized Architecture

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

In this architecture, as shown in Figure 4, CORBA IDL interfaces are used at two distinct levels: (1) the scheduler process contains threads which communicate with task managers using CORBA IDL interfaces; and (2) task managers communicate with tasks using CORBA IDL interfaces.

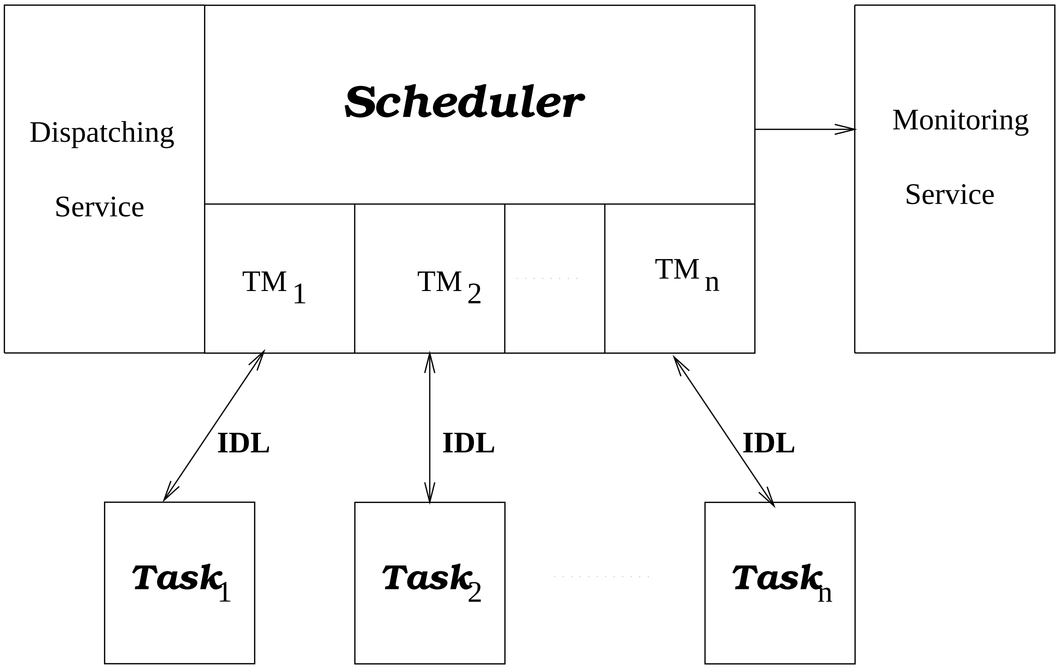

Figure 3: Highly Centralized Scheduler

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

4.3 Asynchronous Centralized Architecture

This architecture does not use thread agents or threads for task managers. As shown in Figure 5, the scheduler communicates with task managers using asynchronous IDL interfaces. Task managers send asynchronous messages back to the scheduler if necessary. Task managers communicate with tasks using synchronous IDL interfaces, although this interface could be made asynchronous as well.

The dispatching service sends an asynchronous message to initiate the corresponding task manager for a task using an asynchronous method call through the ORB. After a task completes a state, the task manager sends an asynchronous message to the message collecting service which is designed as a server incorporated into the scheduler process. The scheduler can obtain messages from the message collecting service. In order to process the received messages better, the message collecting service could run as a thread in this architecture. In this scheme, a task id number can be used to identify the task manager from which the message has been sent.

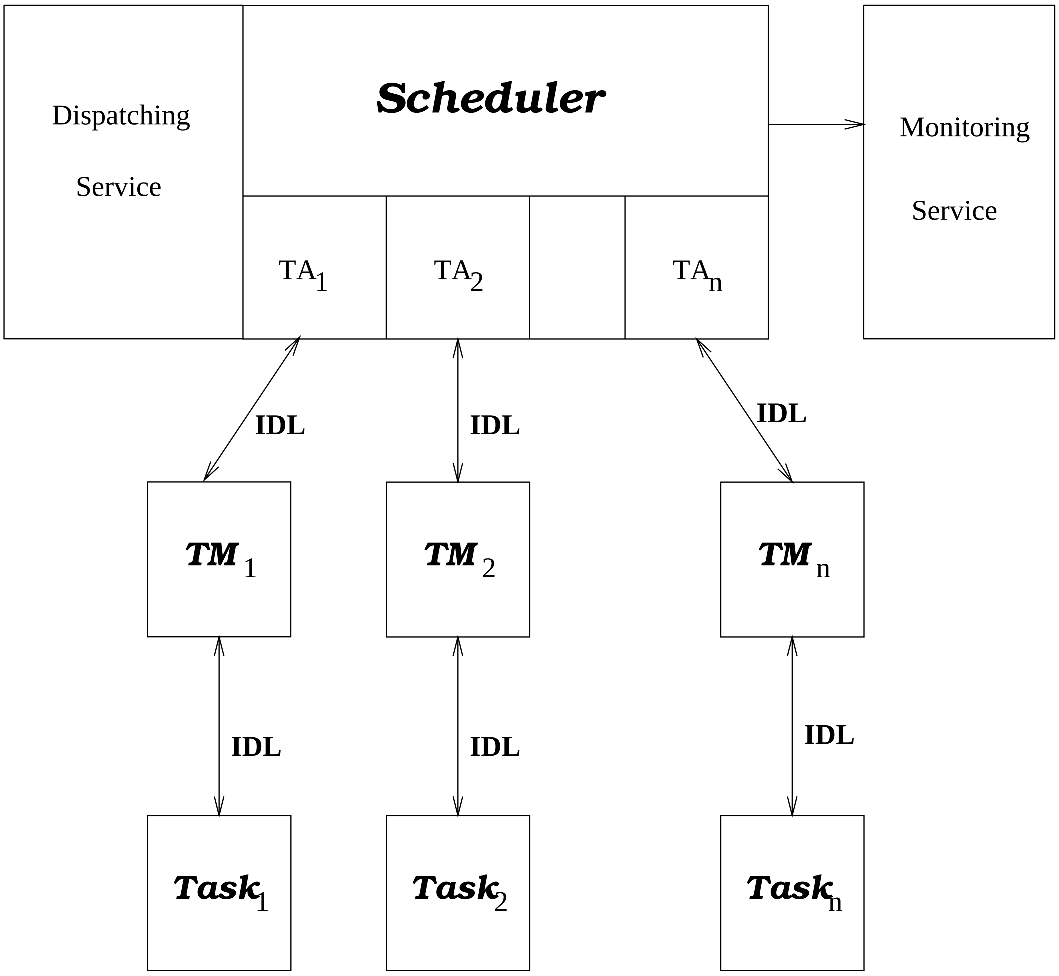

Figure 4: Synchronous Centralized Scheduler

4.4 Semi-Distributed Architecture

In this architecture, as shown in Figure 6, the scheduler communicates with the task managers through asynchronous IDL interfaces. Task managers are also allowed to communicate with other task managers through asynchronous IDL interfaces. The major difference between this architecture and the previous one is that task managers are allowed to talk to each other. Control signals are still sent to the scheduler, while data is sent directly to the appropriate task manager.

The purpose of designing this architecture is to reduce the central scheduler's work burden and to distribute more work to task managers, while still keeping the centralized scheduling algorithm and the advantages of the centralized architecture.

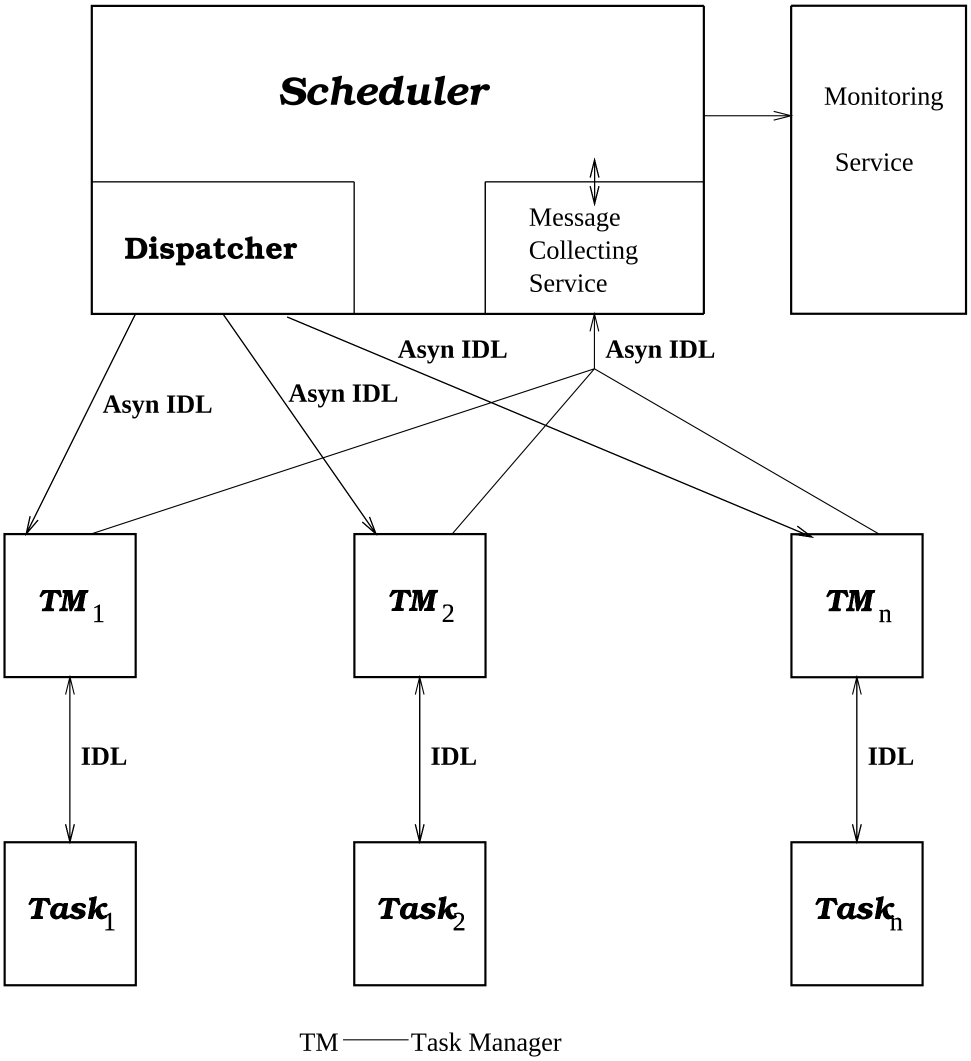

Figure 5: Asynchronous Centralized Architecture

4.5 Fully Distributed Architecture

In the fully distributed architecture, as shown in Figure 7, there is no centralized scheduler. In the figure, each task manager (marked as TM) is equipped with a fragment of code which determines if and when a given task is supposed to start execution. Since a task may depend on a number of other tasks (either AND-ed or OR-ed), an AND-OR tree is used (normalized to a disjunction of conjunctions) and compiled into a fragment of C++ code, essentially acting as part of the scheduler of the overall workflow.

As seen in Figure 7, the layout of the workflow closely resembles the workflow as it was designed in the graphical designer (see Figure 2). Individual task managers have "knowledge" of only the successor tasks. They communicate with other task managers through asynchronous IDL interfaces.

The monitoring service is used to watch over the workflow execution. Individual task managers

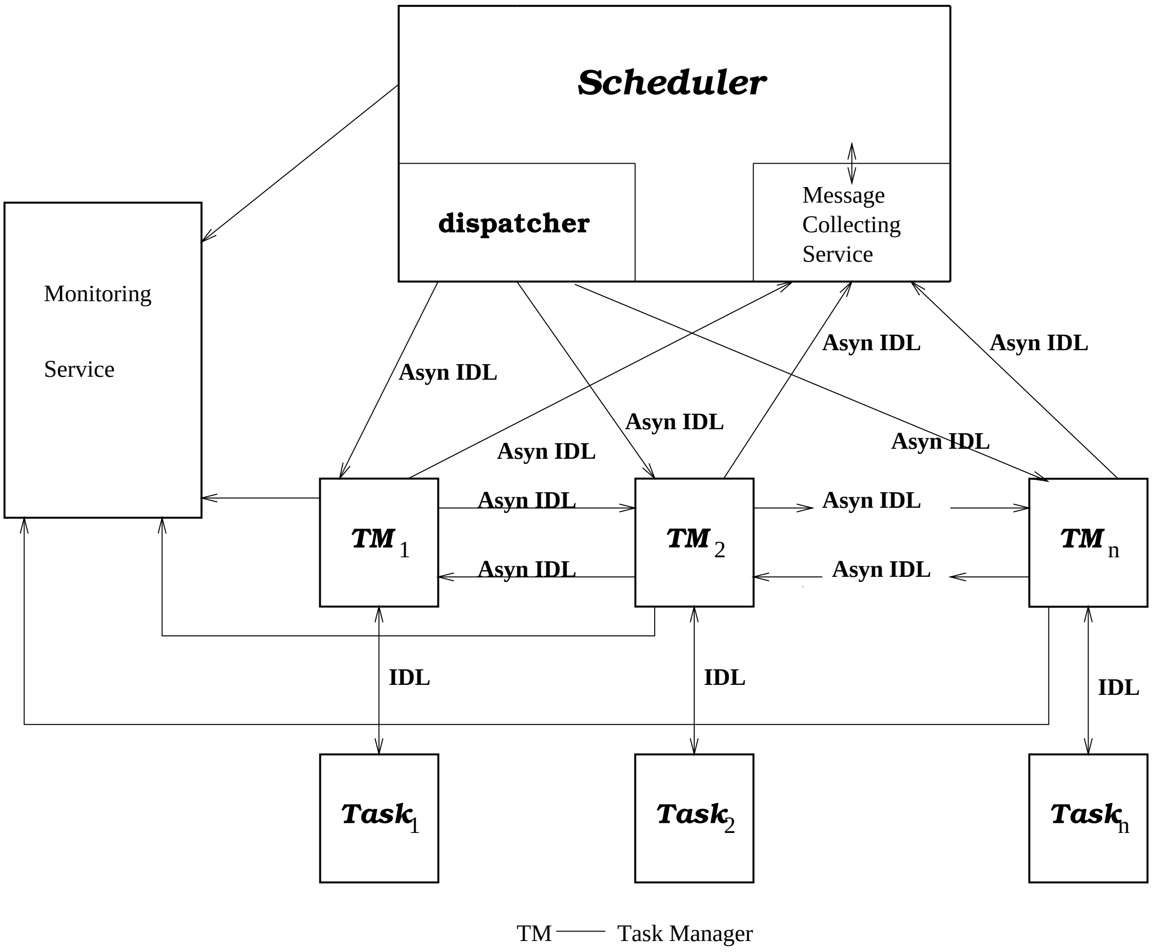

Figure 6: Semi-Distributed Architecture

communicate to this service their internal observable states, as well as data object references (for possible recovery). Task managers communicate with the monitor by asynchronous messages sent through the ORB.

The distributed architecture matches the inherent distributional character of workflow very well. Also, it eliminates the bottleneck of task managers having to communicate with a remote centralized scheduler during the execution of the workflow. Another advantage of this architecture is its resiliency to failures. If one node crashes, only a part of the workflow is affected.

5 TASK STRUCTURES AND TASK MODELS

A task structure indicates the generic form of a task. A structure simply identifies a set of states and the permissible transitions between those states. A full task specification will fix the type of the task. Beyond the task structure, this also requires a specification of allowable operations. Following the object-oriented paradigm, each state will correspond to an operation (or method).

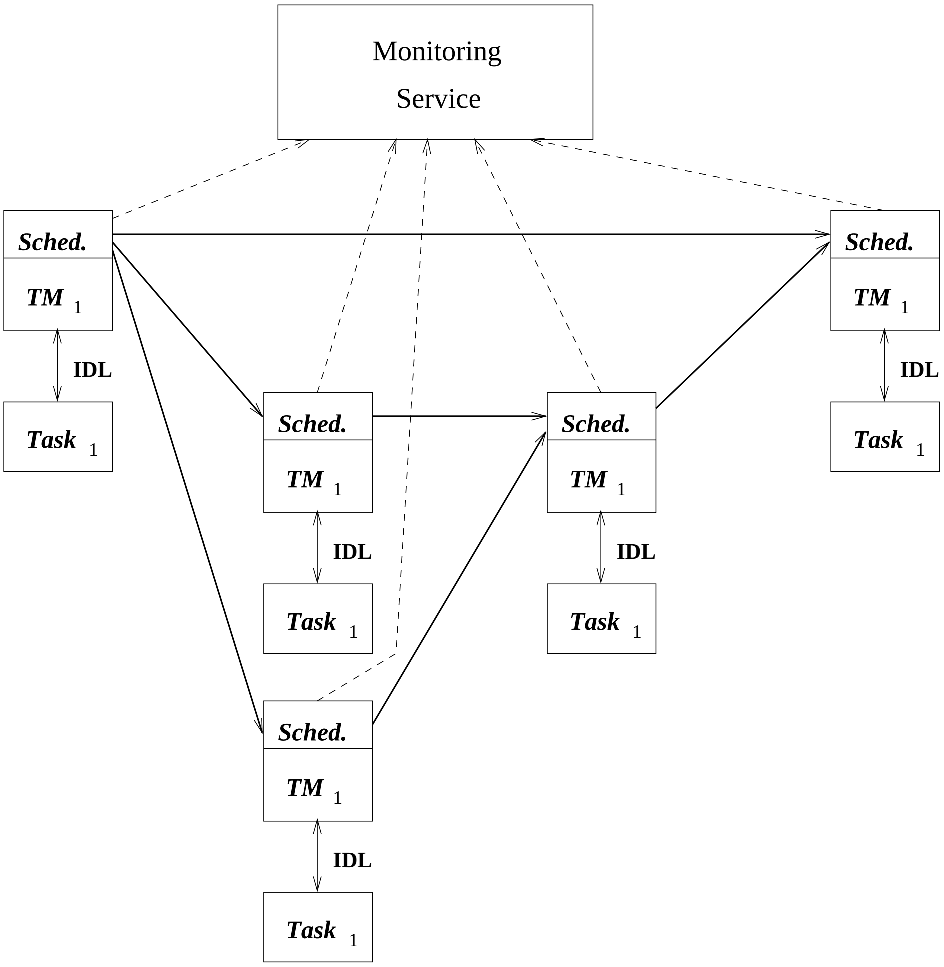

Figure 7: Fully Distributed Architecture

Several different task structures have been developed [ASSR93, KS95, Wan95].

In our system, diagrammatic workflow models [KS95, Mur95] are inherently hierarchical. At the top level is a model for the overall workflow (see Figure 1), below this are subworkflows (or compound tasks) (see Figure 2) which themselves may be made up of subworkflows or simple tasks. Each task is further modeled via a state transition diagram (see Figure 9). Optionally, the gates (see section 5.1) guarding states may be displayed. These graphical workflow and task models created using the Graphical Workflow Designer, along with supplementary specifications (e.g., conditions, inputs, outputs, etc.), are the basis for partially automated code generation for actual workflows.

Our workflow system also allows workflows to be specified textually (although graphical specification is preferred). WFSL/TSL may be used to specify workflows [KS95]. We are currently in the process of designing a language that adds WFSL/TSL-based features to CORBA's Interface Definition Language (IDL). For brevity, we discuss some elements of this language below using a simple example. Consider an application in which money is to be withdrawn from an account in one bank and deposited into an account in another bank. This is an example of transactions on multiple databases. The specification for this workflow that coordinates the execution of two transactions is given in Figure 8 which displays the Workflow Interface Definition Language (WIDL). A full task specification requires that the parameters and return type of each operation be specified. In essence, this is analogous to a class specification in C++. However, to facilitate interoperability and distribution, the specification is in the form of a CORBA IDL interface specification. In addition to operations (methods), attributes and exceptions may be specified.

Clients may call this interface using any language for which a binding is provided (e.g., C, C++, SmallTalk). Servers (i.e., implementations) have this same flexibility.

For our WFMS system, the base interface TTask (Transactional Task) is implemented as an abstract class (all member functions (methods) are pure virtual). Thus, the base class serves as an application framework [Str91].

For simplicity, only CORBA `in` and `out` parameters are allowed. All `in` parameters (the inputs) are grouped together into a single structure derived from WFL_Data (similarly for the `out` parameters (outputs)). These structures may contain embedded object references so that actual objects are not sent over the network, rather references to CORBA objects are passed.

5.1 Intertask Dependencies and Enable Arcs

Coordination between tasks is accomplished by specifying intertask dependencies using `enabled by` clauses. An `enabled by` clause may have any number of predicates. Each predicate is either a task-state vector or a Boolean expression.

```
enabled by ([<task_1>, <state_1>] && [<task_2>, <state_2>] && <condition>)
    at <time> ;
```

The entire clause will be represented in Disjunctive Normal Form (DNF), i.e., the `or` of `and` clauses. Just as in most programming languages, the `and` operator is given higher precedence than `or`. Therefore the clause will be in DNF if no parentheses are used. If parentheses are used, then the clause is converted to DNF.

This is similar to an Event-Condition-Action (ECA) rule [C+89]. When a specified event occurs (e.g., <task_1> leaves <state_2>) the corresponding task-state vector is set to true. Then if the entire expression including <condition> evaluates to true, the action is enabled. The <condition> may be any Boolean expression on the `in` parameters of the corresponding method calls. For example, suppose <task_2> is Deposit_Task and <state_2> is Execute, then a possible condition might be of the following form.

```
payment.money >= 1400.00 && payment.retries <= 3
```

```
struct Money_Transfer : WFL_Data {
    long   account_no;
    double money;
    long   retries;
}; // Money_Transfer

interface Bank_Workflow : Compound_TTask {
    long Execute ();
    long Abort ()
        enabled by ([Withdrawal_Task, Abort] || [Deposit_Task, Abort]);
    long Commit ()
        enabled by ([Deposit_Task, Commit]);
}; // Bank_Workflow

interface Withdrawal_Task : TTask {
    long Execute (in Money_Transfer request)
        enabled by ([Bank_Workflow, Execute]);
    long Abort ();
    long Commit (out Money_Transfer payment);
}; // Deposit_Task

interface Deposit_Task : TTask {
    long Execute (in Money_Transfer payment)
        enabled by ([Withdrawal_Task, Commit]);
    long Abort ();
    long Commit ();
}; // Deposit_Task
```

Figure 8: Bank_Workflow.widl

Only if this condition evaluates to true will the remote method call

```
return_value = Deposit_Task->Execute (payment);
```

be scheduled for execution.

A successful enable will cause one leaf node in the gate guarding (<task_2> , <state_2>) to become true. If the gate's AND-OR tree now evaluates to true, the gate will open (it is fully enabled) and the method implementing (<task_2> , <state_2>) will be scheduled for execution. If <state_2> is the root of the directed graph representing <task_2>'s task structure, then enabling causes task initiation. The execution may be scheduled to occur immediately (this is the default) or at a given time in the future. For scheduling using real time, one could potentially address many complex issues. Our WFMS is at present not intended for use in domains where hard real time constraints are present; we simply incorporate deadlines into the calculation of dynamic priorities. The dynamic priority will affect placement on WFMS dispatch queues, and optionally the actual execution priority for operating systems that allow such influence.

The specifications of Task Interfaces, Enable Arcs and Gates are all embedded in the .widl file for a workflow. This file is preprocessed to provide the following: (1) a task manager for each interface; (2) a .idl file for each interface; and (3) a load module for the scheduler (load_wf_structure.cc). The load module is an automatically generated function which loads the workflow structure. The workflow structure is a data structure within the scheduler that represents each task's task structure, the enable arcs between tasks and the gates guarding certain states, as well as the current state, priority, etc. of each task.

5.2 Varieties of Task Structures

In this subsection, we discuss several useful task structures. One fundamental question is whether to provide a fixed set of task structures or allow users (albeit sophisticated ones) to create their own task structures. If the latter is chosen, then surely design constraints should be enforced. A proliferation of task structures would likely lead to incomprehensibility. We have chosen to give users the ability to design task structures, and are still working on formulating a simple set of design constraints. In the rest of this subsection, we present several fundamental task structures that we believe should be built into any WFMS. The task structures are shown as directed graphs (typically acyclic). The nodes in the graph correspond to the externally visible states, while the arcs correspond to permissible internal transitions. An internal transition is said to be controllable if it can be affected by another task via an enable arc; otherwise, the internal transition is said to be uncontrollable. Note that the figures depicting task structures also show sample enable arcs, even though these are not part of the task structure per se.

A non-transactional task is used when an ordinary application that does not enforce atomicity or isolation is to be included in a workflow. This model does not permit micromanagement of the flow of control within a task. Such a task can be initiated or forced to fail (terminate early), but that is it. The externally visible states of a simple non-transactional task are *initial*, *executing*, *failed* and *done*. The task structure is shown in Figure 9. Optional enable arcs may come into the *failed* state to force the task to fail. The transition *i-e* would be uncontrollable if no enable arc comes into the *executing* state, and controllable, otherwise. Transitions *e-f* and *e-d* would be controllable if any enable arc comes into the *failed* state since the transition *e-d* would be rejected if an enable arc forces the task to fail. They are uncontrollable if no enable arc comes into the *failed* state.

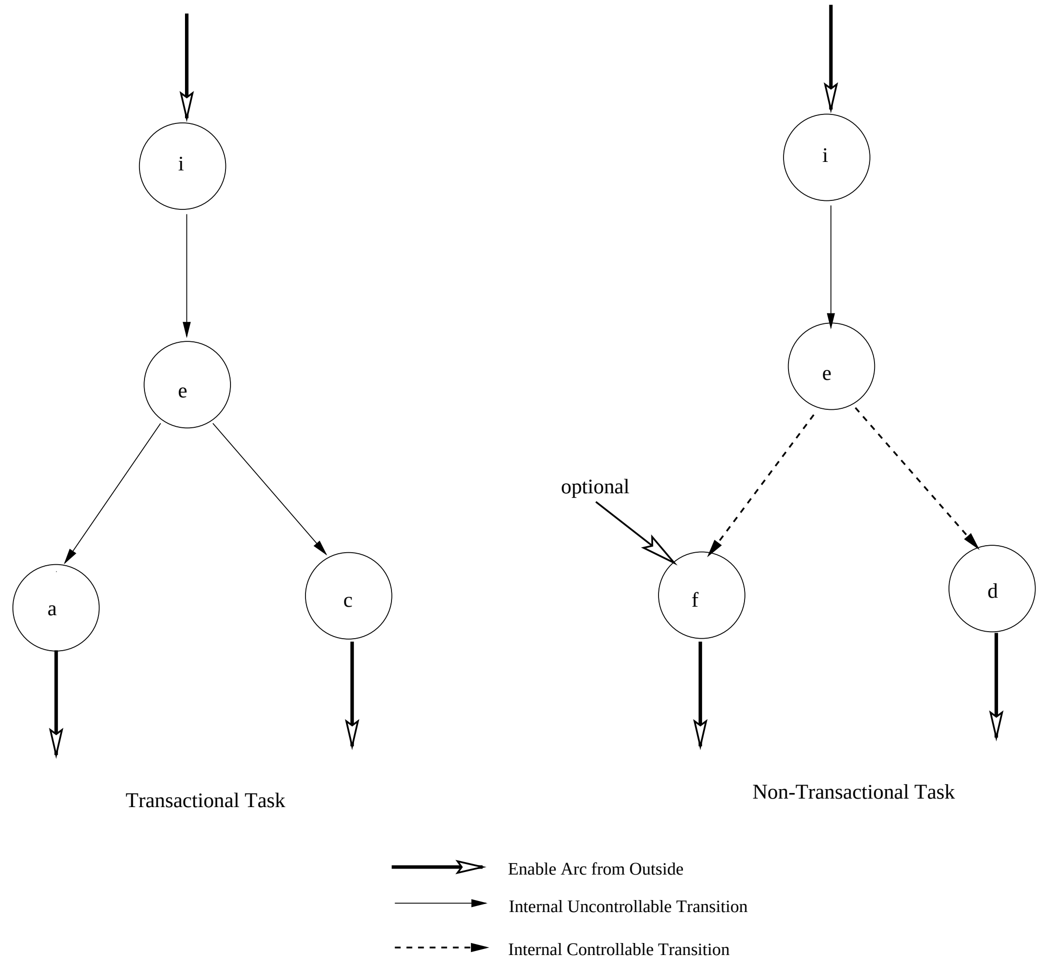

Figure 9: Task Structures

Figure 9 also shows the transactional task structure. A transactional task minimally supports the atomicity property and maximally supports all ACID (Atomicity, Consistency, Isolation, and Durability) properties [GR93]. The externally observable states of a transactional task are *initial*, *executing*, *aborted* and *committed*. The transition *i-e* is uncontrollable. Optional enable arcs may come into the *aborted* state from outside the task to force the task to abort. Transitions *e-a* and *e-c* are controllable if there are any optional enable arcs since the transition *e-c* would be rejected. They are uncontrollable, otherwise.

A compound task organizes a collection of related tasks into a group. These subtasks (children tasks) can only communicate with each other and the compound task (parent task). Consequently, compound tasks provide a mechanism for hierarchically organizing a workflow. There are two types of compound tasks, transactional compound tasks and non-transactional compound tasks. The task structures are the same as those for simple tasks (see Figure 9). The basic techniques for supporting transactional compound tasks are (i) two-phase commitment, (ii) nested transactions [Mos82], and (iii) sagas [GMS87]. More complex strategies are possible, but we ignore them for brevity.

Two-Phase Commit (2PC) is a well-known protocol allowing a set of participants to eventually all commit or all abort. Typically, a workflow may involve some transactions supported by DBMSs which need the 2PC feature [KS95]. Therefore, it is necessary to provide the 2PC task to describe this feature. We have designed and implemented a Two-Phase Commit (2PC) coordinator task structure and a coordinator that can enforce two-phase commit involving the members of a compound task. For details on both task structures see [Wan95].

6 CONCLUSIONS

In this paper, we presented five run-time architectures for a workflow management system. There are advantages and disadvantages to each. The centralized architectures are somewhat easier to implement and make simulation, animation and monitoring easier [MSK+95]. The more distributed architectures should enhance reliability and eliminate potential bottlenecks at the machine running the centralized scheduler. However, if an organization uses a powerful multiprocessor as a centralized server, these bottlenecks may not be significant. We have implemented, evaluated and compared the Highly Centralized and Synchronous Centralized architectures [MSK+95]. This limited study suggests that the Highly Centralized architecture is superior when tasks have heavier CPU requirements, while the Synchronous Centralized architecture is superior when CPU requirements for task managers are substantially greater than the CPU requirements for tasks. We are now in the process of implementing two more of the architectures and plan to carry out a comprehensive study comparing the performance and reliability of these architectures.

Another near term goal is to support transactional workflows. Our current prototypes allow for transactional tasks in the case that the processing entity (e.g., a DBMS) supports this capability or a task has no side-effects. The WFMS at present provides no direct support to enhance transactional capabilities or provide global transactional capabilities. (Note, one exception is that we do provide a two-phase commit coordinator to facilitate transactional tasks committing as a group.) In our next prototype, we plan to utilize the transactional capabilities of CORBA 2.0 Object Services to achieve this goal more fully.

In addition to considering the advantages and disadvantages of the five architectures, we have also considered the suitability of building a WFMS on top of CORBA (specifically ORBeline, a CORBA 1.2 implementation). The main advantages that CORBA provides over other possible communication infrastructures are the following: The object-oriented nature of CORBA enables clients to communicate with servers simply by executing a remote method call. Invoking a remote service is easy; simply find the object (using `_bind`) and then call the remote method passing the relevant parameters. Objects may be found based upon their interface and optionally, a host machine name or object name. Once found, a reference to the object may be maintained. The use of a well-defined interface (in IDL) facilitates distribution, interoperability and heterogeneity (allowing clients and servers to use different languages, different machines, different types of machines as well as run under different operating systems). Finally, the usual advantages of object-oriented programming brought about by encapsulation and inheritance, such as protection, information hiding, modularity, and code reuse, were found to be indeed valuable. We believe that COBRA 1.2 provides a solid foundation for building a Workflow Management System.

In this paper, we also proposed a minor extension to CORBA's IDL to provide an alternative means for specifying workflows. This language called Workflow Interface Definition Language (WIDL) simply adds an `enabled by` clause to IDL. We believe that WIDL would be useful to those already familiar with IDL. It also shows that much of the specification given in a language like WFSL can be captured in just IDL.

The newest CORBA standard, 2.0, is even more suitable as an underlying technology for Workflow Management Systems, particularly ones supporting transactional workflows. CORBA 2.0 provides several powerful object services [OMG95b] which would be useful in building a workflow management system.

- **Naming Service.** This service allows names to be assigned to objects. Names must be unique within a naming context (analogous to filenames within a directory). This provides a flexible way for objects to be looked up.

- **Event Service.** Events can be thought of as messages sent between clients and servers. They differ from ordinary method calls in that method calls require both client and server to be running (or if the server is not running, it will be activated), while events do not.

- **Persistent Object Service.** This service allows the private data within an object to be made persistent, so that the object maintains its state even if the process containing it terminates.

- **Life Cycle Service.** This service facilitates the creation, destruction, movement and duplication of objects. To create multiple objects for a certain interface, the user should write a factory interface for it.

- **Concurrency Control Service.** A general locking facility is provided to allow multiple clients to access a shared server object without corrupting its state. Locking may be used by both transactional and non-transactional clients.

- **Externalization Service.** This service permits a snapshot of an object's state to be saved in a data stream (e.g., an ASCII file). It facilitates saving, displaying and copying objects (even to different ORBs).

- **Relationship Service.** Objects (or entities) can form relationships with other entity objects. Relationships are stored in relationship objects. The relationship structures and constraints are rich, including much of what is found in ER and OMT models.

- **Transaction Service.** This service combined with the concurrency control service greatly reduces the amount of work required to implement a WFMS supporting transactional workflows. Both flat and nested transactional models are supported.

References

[AAA+95] G. Alonso, D. Agrawal, A. Abbadi, C. Mohan, M. Kamath, and R. Guenthoer. Exotica/FMQM: A persistent message-based architecture for distributed workflow management. In *Proc. of the IFIP Working Conference on Information Systems Development for Decentralized Organizations*, pages 1–18, Trondheim, Norway, August 1995.

[AKA+95] G. Alonso, M. Kamath, D. Agrawal, A. Abbadi, C. Mohan, and R. Guenthoer. Exotica/FMDC: Handling disconnected clients in a workflow management system. In S. Laufmann, S. Spaccapietra, and T. Yokoi, editors, *Proc. of the 3rd Intl. Conference on Cooperative Information Systems*, pages 99–110, Vienna, Austria, May 1995.

[ANRS92] M. Ansari, L. Ness, M. Rusinkiewicz, and A. Sheth. Using flexible transactions to support multi-system telecommunication applications. In *Proc. of the 18th Intl. Conference on Very Large Data Bases*, pages 65–76, August 1992.

[ASSR93] P. Attie, M. Singh, A. Sheth, and M. Rusinkiewicz. Specifying and enforcing intertask dependencies. In *Proc. of the 19th Intl. Conference on Very Large Data Bases*, pages 134–145, Dublin, Ireland, 1993.

[BDSS93] Y. Breitbart, A. Deacon, H. Schek, and A. Sheth. Merging application-centric and data-centric approaches to support transaction-oriented multi-system workflows. *SIGMOD Record*, 22(3):23–30, September 1993.

[Bet95] M. Betz. OMG's CORBA. *Dr. Dobb's Journal*, 9(16):8–13, March 1995.

[BMR94] D. Barbara, S. Mehrotra, and M. Rusinkiewicz. INCAs: A computation model for dynamic workflows in autonomous distributed environments. Technical report, University of Houston, May 1994.

[C+89] U. Chakravarthy et al. HiPAC: A research project in active time-constrained database management. Technical Report XAIT-89-02, Xerox Advanced Information Technology, Cambridge, MA, 1989.

[CR90] P. Chrysanthis and K. Ramamritham. ACTA: A framework for specifying and reasoning about transaction structure and behavior. In *Proc. of ACM SIGMOD Conference on Management of Data*, pages 194–203, Atlantic City, NJ, 1990.

[CR92] P. Chrysanthis and K. Ramamritham. *ACTA: The SAGA Continues*, chapter 10. Morgan Kaufman, San Mateo, CA, 1992.

[DHL91] U. Dayal, M. Hsu, and R. Ladin. A transactional model for long-running activities. In *Proc. of the 17th Intl. Conference on Very Large Data Bases*, pages 113–122, Barcelona, Spain, September 1991.

[Dui94] M. Duitshof. Workflow automation in three administrative organizations. Master's thesis, University of Twente, Netherlands, July 1994.

[Eme90] E.A. Emerson. Temporal and model logic. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B. Elselvier, Amsterdam, Netherlands, 1990.

[FKB95] A. Forst, E. Kuhn, and O. Bukhres. General purpose work flow languages. *Distributed and Parallel Databases*, 3(2):187–218, April 1995.

[GH94] D. Georgakopoulos and M.F. Hornick. A framework for enforceable specification of extended transaction models and transactional workflows. *International Journal of Intelligent and Cooperative Information Systems*, 3(3):599–617, 1994.

[GHS95] D. Georgakopoulos, M. Hornick, and A. Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2):119–154, April 1995.

[GMS87] H. Garcia-Molina and K. Salem. Sagas. In *Proc. of ACM SIGMOD Conference on Management of Data*, pages 249–259, San Francisco, CA, May 1987.

[GR93] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[HK95] M. Hsu and C. Kleissner. ObjectFlow: Towards a process management infrastructure. Technical report, Digital Equipment Corporation, 1995.

[J+95] J. Juopperi et al. Usability of some workflow products in an inter-organizational setting. In *Proc. of the IFIP Working Conference on Information Systems Development for Decentralized Organizations*, Trondheim, Norway, August 1995.

[JAD+94] S. Joosten, G. Aussems, M. Duitshof, R. Huffmeijer, and E. Mulder. *WA-12: An Empirical Study about the Practice of Workflow Management.* University of Twente, enschede, The Netherlands, July 1994. Research Monograph.

[Kle91] J. Klein. Advanced rule driven transaction management. In *Proc. of the IEEE COMPCON*, pages 562–567, San Francisco, CA, 1991. IEEE Computer Society.

[KRR95] G. Kappel, S. Rausch Schott, and W. Retschitzegger. TriGSflow active object-oriented workflow management. Technical report, Department of Computer Science, University of Linz, Austria, 1995.

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

[LA94] F. Leymann and W. Altenhuber. Managing business processes as an information resource. *IBM Systems Journal*, 33(2):326–348, 1994.

[M+95] C. Mohan et al. Exotica: A project on advanced transaction management and workflow systems. *ACM SIGOIS Bulletin*, 16(1):45–50, August 1995.

[Mos82] J. Moss. Nested transactions and reliable distributed computing. In *Proc. of the Second Symposium on Reliability in Distributed Software and Database Systems*, pages 33–39, Pittsburgh, PA, July 1982.

[MSK+95] J.A. Miller, A.P. Sheth, K.J. Kochut, X. Wang, and A. Murugan. Simulation modeling within workflow technology. In *Proc. of the 1995 Winter Simulation Conference*, Arlington, VA, December 1995.

[Mur95] A. Murugan. Graphical workflow designer. Master's thesis, University of Georgia, 1995. (in preparation).

[OMG93] OMG. The common object request broker: Architecture and specification. Technical report, Object Management Group, December 1993.

[OMG95a] OMG. The common object request broker: Architecture and specification, revision 2.0. Technical report, Object Management Group, July 1995.

[OMG95b] OMG. CORBAservices: Common object services specification. Technical report, Object Management Group, March 1995.

[Rei94] B. Reinwald. Tutorial notes on workflow-management. Technical report, IBM, Almaden, August 1994. presented at the 13th IFIP World Computer Congress.

[RS95a] A. Reuter and F. Schwenkreis. Contracts - a low-level mechanism for building general-purpose workflow management systems. *IEEE Data Engineering Bulletin*, 18(1), 1995.

[RS95b] M. Rusinkiewicz and A. Sheth. Specification and execution of transactional workflows. In W. Kim, editor, *Modern Database Systems: The Object Model, Interoperability and Beyond*, pages 592–620. ACM Press, New York, NY, 1995.

[She95] A. Sheth. Tutorial notes on workflow automation: Application, technology and research. Technical report, University of Georgia, May 1995. presented at ACM SIGMOD, San Jose, CA, http://www.cs.uga.edu/LSDIS.

[Smi93] T. Smith. The future of workflow software. *INFORM*, pages 50–51, April 1993.

[SR93] A. Sheth and M. Rusinkiewicz. On transacational workflows. *IEEE Data Engineering Bulletin*, 16(2):1–4, June 1993.

[Str91] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, Reading, MA, second edition, 1991.

[TV95] J. Tang and J. Veijalainen. Enforcing inter-task dependencies in transactional workflows. Technical Report J-2/95, VTT Information Technology, Espoo, Finland, January 1995.

[Wan95] X. Wang. Implementation and performance evaluation of CORBA-based centralized workflow schedulers. Master's thesis, University of Georgia, August 1995.

[WF94] T. White and L. Fischer. *The Workflow Paradigm - The Impact of Information Technology on Business Process Reengineering*. Future Strategies, Inc., Alameda, CA, 1994.