December 2000

Workflow System Architectures and their Performance Model

Dongsoo Han
Information and Communications University

Jaeyong Shim
Information and Communications University

Myungjae Kwak
Information and Communications University

Workflow System Architectures and Their Performance Model

Dongsoo Han[1], Jaeyong Shim[2], Myungjae Kwak[3]
School of Engineering,
Information and Communications University
P.O. Box 77, Yusong P.O., Taejon, 305-600 Korea

Abstract
Various workflow system architectures are proposed to meet diverse requirements for workflow management system in the internet environment. Centralized, decentralized, distributed and fully distributed workflow system architectures are considered to be the generally accepted typical workflow system architectures. Each workflow system architecture is known to have its pros and cons in different situations but there are no clear criteria and performance evaluation models to compare them each other yet. In this paper we develop performance evaluation model for workflow system architecture, focusing on the proto-type workflow system developed on the CORBA environment. The proto-type workflow system can change its architecture arbitrarily. Thus the performance evaluation model can be applied to any workflow system architectures but it is more feasible to apply to workflow systems developed on the distributed object environment.

Keywords: Workflow system architectures, Distributed workflow system, Task managing instance, Workflow performance model

1. Preface

Since the needs of workflow system had been invoked, numerous workflow systems have been developed for either commercial or academic purposes (Alonso, 1995; Ellis, 1995; Han, 1996; Kim et al., 1997). Recently, various workflow system architectures are proposed to meet diverse requirements for workflow management systems. Centralized, decentralized, distributed and fully distributed workflow system architectures are considered to be the generally accepted typical workflow system architectures.

Centralized workflow system is criticized of its limitation to support workflow systems on the internet because of its shortage of scalability and flexibility. Decentralized or distributed workflow systems often have been suggested to overcome the limitation in centralized workflow system. In decentralized workflow system, multiple workflow servers work together independently dividing the input jobs into a mass and there is no interaction among workflow servers to complete a process instance. In distributed workflow system, multiple workflow servers work together like decentralized workflow system, but there could be interactions among workflow servers during the execution of a process instance. Even fully distributed workflow system can be proposed to give the extreme flexibility and scalability. Each workflow system architecture has its pros and cons according to the situation that the workflow system has to manage. However there are no clear criteria and performance evaluation models to compare them each other yet. As a result, people tend to advocate a

[1] Dr. Dongsoo Han is an assistant professor at the School of Engineering of Information and Communications University and can be approached via email: dshan@icu.ac.kr.
[2] Jaeyong Shim is in doctorial course at the School of Engineering of Information and Communications University and can be approached via email: jaeyong7@icu.ac.kr.
[3] Myungjae Kwak is in M.S. course in the School of Engineering of Information and Communications University and can be approached via email: mjkwak@icu.ac.kr.

certain type of workflow system architecture with rather unclear conjecture.

To break through the situation, more concrete and clear criteria to compare workflow system architectures are required. Performance model of workflow system architectures could be a good candidate for such purposes. People can discuss or choose better workflow system architectures in terms of system performance in the given conditions once the performance model is provided.

In this paper we derive a performance model for workflow system architectures. The performance model is developed referring to a sample workflow system developed on CORBA environment. For a performance model to be applicable to large domains, it should reflect general workflow system architectures. Since the sample workflow system can change its architecture arbitrarily if necessary, the performance model can be applied to any workflow system architectures. We introduce how the sample workflow system can change its architectures in detail. Even though the performance model can be applied to any workflow system architectures, it is more feasible when it is applied to workflow systems developed on distributed object environment, such as CORBA or DCOM.

Performance evaluation for the workflow system architectures has been performed using the model varying the conditions. The results of the test reveal the characteristics of each workflow system architectures as expected and fall into the scope what we can expect from the test. It implies that the performance model can be used to estimate the performance of workflow systems if some conditions are given.

The paper is organized as follows. In section 2, we classify workflow types and workflow architectures. In section 3, system architecture and the components of sample workflow system are described. In section 4, we derive performance models of each architecture and show the result of the performance evaluation. Related work are described in section 5 and we draw conclusion in section 6.

2. Workflow Types and Workflow Architectures

2.1 Workflow Types

Workflow processes can be classified into several different workflow types according to their inherent properties. For example, while there are workflow processes requiring hard real time process controls and monitoring services, other workflow processes do not require such a strong process controls and monitoring services. Rather they often prefer process controls in which task managers can react automatically and independently. The latter can support workflow processes more effectively in mobile environment? We call the former as *control/monitoring oriented* workflow processes and the latter as *autonomy oriented* workflow processes.

In some other workflow processes, the vicinity of the application server and workflow enactment services is critical. The workflow processes often require the workflow enactment services or the task manager to be located close to the application server. We call the workflow processes as *application server oriented* workflow processes.

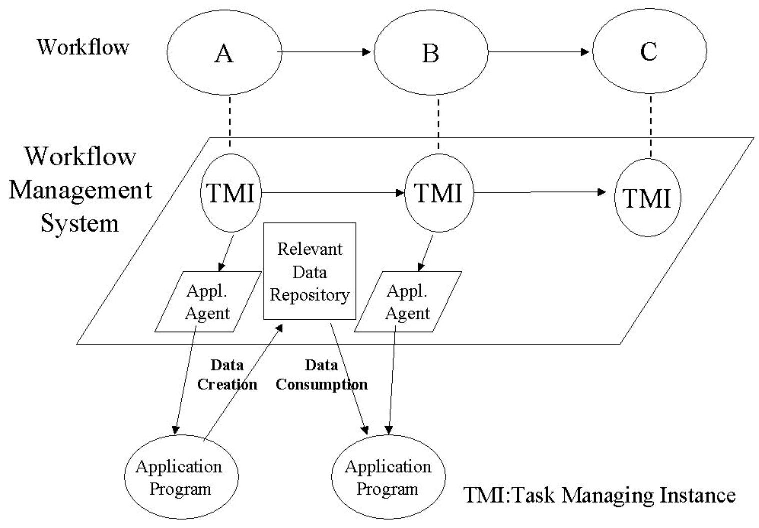


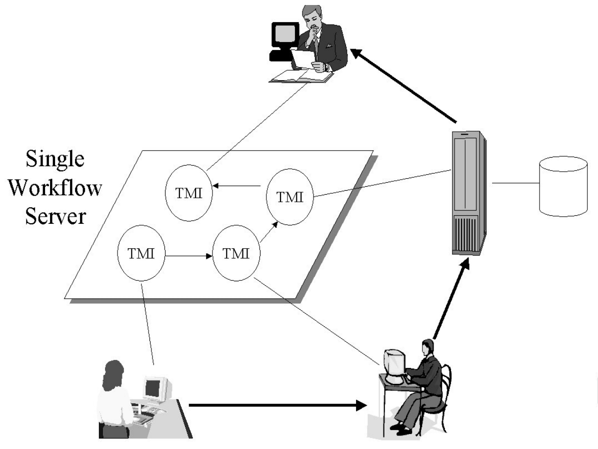
Figure 1. Run Time Architecture of Workflow Enactment Service

2.2 Run Time Architecture of Workflow Enactment Services

Workflow enactment services can be implemented in various ways. But the run time architecture of each enactment services can be derived as Figure 1. The execution object in Figure 1 manages from the invocation to the end of the corresponding task. When the task is over, it signals the next execution object to begin its task. The workflow enactment services can be considered as the collection of the execution objects. Thus the workflow enactment services can be implemented in various ways according to the method we implement the execution objects. They can be implemented in a monolithic module or several modules which are able to corporate one another to provide services. Also it is possible that each execution object is implemented as a transient object instance. In that case, workflow sever should be equipped with the generator and disposer of execution object instance.

2.3 Classification of Workflow Architectures

Workflow systems can be classified into several different workflow architectures according to the displacement of the execution object instances which we call TMI(Task Managing Instance) in this paper. In centralized workflow system, all the TMIs are created and operated



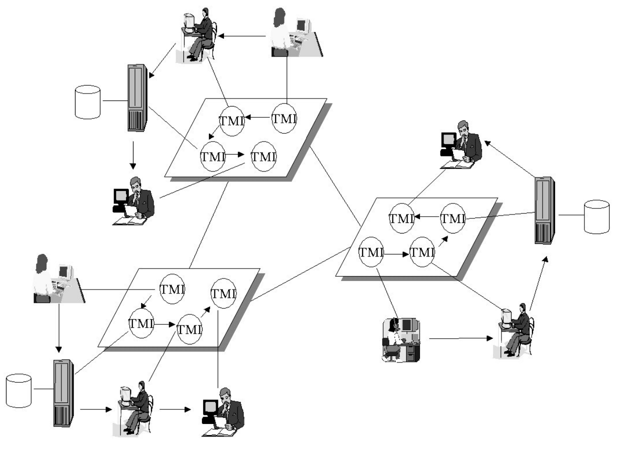


Figure 2. Centralized Workflow System

Figure 3. Decentralized Workflow System

in one central workflow server. Since the TMI creation is relatively simple compared with other kind of workflow systems, the system is very simple. Figure 2 shows the centralized workflow system. In decentralized workflow system, created TMIs are placed in several workflow servers keeping all the TMIs processing activities in the same workflow process created in the same server. Decentralized workflow system architecture can be considered as a kind of multi-workflow server implementation. Figure 3 shows the run time structure of the decentralized workflow system.

Distributed workflow system is the same with decentralized workflow system in that it places the created TMIs in physically distributed multiple workflow servers. But there is no restriction that TMIs in the same workflow process should be created in the same workflow server in distributed workflow system. Hence more flexible and situation compliant TMI placement is possible in distributed workflow system. The run time structure of distributed workflow system is in Figure 4.

In fully distributed workflow system, TMIs are created in the client sites where the workflow activity processing actually happens. Limited functions, such as first TMI creation and starting, remain in central workflow server. Once the first TMI, however, start to manage its task, the control of the workflow process is handed over to the TMIs in the clients. The TMIs in the clients can act and react upon its tasks from the server autonomously and independently and there is no control from the server after the first TMI started. Thus fully distributed workflow system architecture can cope with mobile environment, because autonomous and independent control for mobile clients are often required in the mobile environment. Figure 5 shows the workflow system architecture of fully distributed workflow system.

2.4 Comparison of Workflow System Architectures

Each workflow system architecture introduced in the previous subsection has its pros and cons to certain situation. Centralized workflow system is very effective to monitoring and controlling the workflow process in real time. Robust workflow system can be implemented relatively easily because the system structure is simple and most of workflow operations executed in a server. Control/ monitoring oriented workflow types are well suited to centralized workflow system. However the system inevitably incurs bottleneck when the

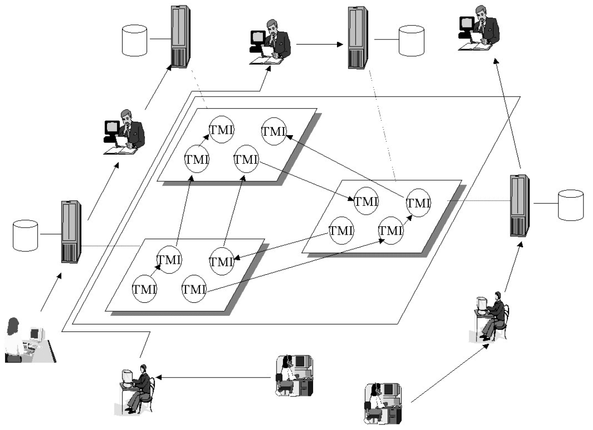


Figure 4. Distributed Workflow System

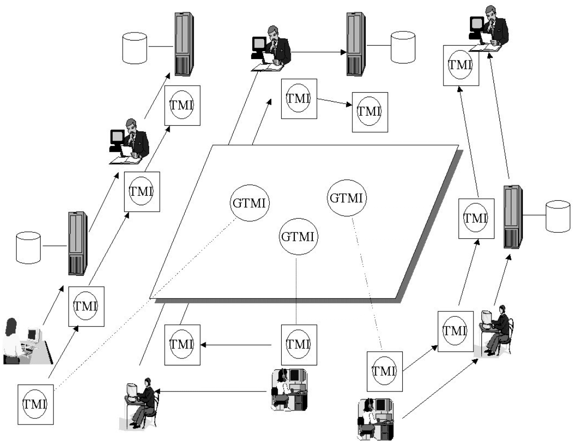


Figure 5. Fully Distributed Workflow System

number of process instances grows over the threshold value the system can accommodate. Coping with the situation in centralized workflow system is non-trivial.

Decetralized workflow system can handle the situation by adding additional servers and dividing and allocating the jobs to the servers. Since all the TMIs for one workflow process instance are generated in a server, the servers need not interact each other to hand over controls between TMIs. They only have to interact for gathering monitoring or historic information of process instances. Thus workflow servers in decentralized workflow system operate in fairly loosely coupled mode. But the restriction that all the TMIs for one process instance should be generated only in a server could be a barrier to serving various workflow types effectively. For instance decentralized workflow system cannot cope with the situation some distributed legacy application servers involved in one process instance prefer TMIs to be placed near to them.

Distributed workflow system is more flexible than decentralized workflow system because it can place TMIs on any workflow servers. Thus above situation can be handled easily in distributed workflow system. But the distribution of TMIs for one process instance on multiple workflow servers could incur interactions between workflow servers and the overhead of the interactions depend on the way of TMI placement. Thus desirable TMI placement strategy should be devised and deployed in distributed workflow systems. However distributed workflow system has also limitation in mobile environment where the network disconnection is frequent.

In fully distributed workflow system, since TMIs are pushed to the client sites, users can work on the work items in disconnected state of the network. The result and the control of the TMI are handed over after the disconnected state is recovered. Thus the load of centralized workflow servers is drastically reduced. In theory, tremendous number of workflow instances can be accommodated in fully distributed workflow system but the cost of monitoring and system consistency maintenance could be very high. Figure 6 shows the comparison matrix of each architecture.

	Centralized Workflow System	Decentralize Workflow System	Distributed Workflow System	F-Distributed Workflow System
Complexity	Low	Medium	Medium	High
Scalability	Bad	Medium	Good	Very Good
Reliability	Yes or No	Good	Good	Yes or No
Monitoring	Good	Good	Good	Bad
Administration	Good	Good	Good	Bad
Mobility	Bad	Bad	Medium	Good
Flexibility	Bad	Medium	Good	Very Good

Figure 6. Comparison Matrix of each Workflow Architecture

3. Distributed Object Based Workflow System for Modeling

In this section we describe software architecture of ICU/COWS which is a research workflow system developed in the Software Systems Lab. of Information and Communications University in Korea since 1998. The primary aim of the system is to support multiple workflow system architectures within single workflow system architectures. The system is developed using distributed objects on the CORBA environment and the performance model of this paper is developed based on the system. Thus distributed workflow systems which are based on distributed objects like CORBA or DCOM and their system structures are similar to ICU/COWS can use the performance model introduced in this paper to analyze their system performances.

3.1 Software Architecture

This section describes the system architecture and the components of ICU/COWS. ICU/COWS is developed on the CORBA environment using WTS(Workflow Transaction Services) which is specially devised for ICU/COWS. The WTS can be considered as a kind of extension of CORBA OTS(Object Transaction Service) to enable convenient workflow system development. Details of the WTS specifications will be treated in other papers in the near future. The components of ICU/COWS consist of TMIF(Task Managing Instance Factory), GTMIG(Global Task Managing Instance Generator), Simulator, Process Builder, Admin/Monitoring Service, and Worklist Handler. Every component is built as CORBA objects and the details of each module will be described in the following subsections.

3.1.1 TMI and GTMI

TMI(Task Managing Instance) is created for each activity to manage the task of the activity. It either sends a work item to a worklist handler or invokes an application through the application agents. Application agents access the workflow relevant data via TMI. The TMI also monitors the status of the invoked tasks through the communication with worklist handlers or application agents. When a task is completed the TMI sends the start event to the next TMI and the TMI which receives the event starts the task. In this way, control is transmitted as defined at process build time.

GTMI controls the process of the global process instance. It either receives status reports from the TMIs or suspends TMIs transiently to handle requests from the administrator, such as dynamic reconfiguration. Although a TMI has to report its status to its GTMI, TMI can continue its execution even if the GTMI crashes because it does not check whether the GTMI has received its report or not. This approach is effective to achieve availability in a distributed environment where network partitioning is frequent.

3.1.2 GTMIG and TMIFs

Each server is equipped with one GTMIG and one or more TMIFs respectively. GTMIG asks TMIF to generate a GTMI for a process instance and the GTMI asks TMIF to generate all the TMIs for the process instance. Since multiple TMIs are generated and the TMIs may be created on different servers, when GTMI asks TMIF to generate TMIs, it asks the TMIF that resides in the same site as the generated TMIs. To the GTMI, local TMIFs and remote TMIFs are viewed equivalently and they are invoked in the same way. So the workflow system operates in a fully distributed fashion. The TMI generation site is determined by the user directives or by considering the system configuration as described in the subsection 4.1.4. When one server is down, the GTMI searches and uses an alternative server on behalf of the crashed server. In this way, the whole system can maintain high system availability irrespective of system failures.

3.1.3 A Workflow Instance Life Cycle

In this subsection, we explain what is happening in ICU/COWS when a process instance creation is requested by a user. The following is the normal sequence of the steps from a process instance creation to the end of its execution:

a. A user asks to create a certain process instance.
b. GTMIG creates a GTMI for the process instance through TMIF.
c. GTMI creates all TMIs of the process instance through TMIF.
d. GTMI sends a start signal to the first TMI.
e. TMI starts work and sends a start signal to the next TMI once the work is finished successfully.
f. Iterate step e until the last TMI is reached.
g. The last TMI sends an end signal to the GTMI when it finishes.
h. GTMI destroys all the generated TMIs and itself.

Both GTMIG and TMIF reside in all the servers. Therefore if one server crashes the other server can take over the GTMI and TMI creation job instead. When GTMI creates TMIs, it can place the TMIs in several different ways based on the user directives. A user can denote which TMI should be placed on which server explicitly when defining a process template. If there are no user directives at all, TMIs are created in a distributed fashion by GTMI considering the location of application servers and load balancing. GTMI creates a TMI

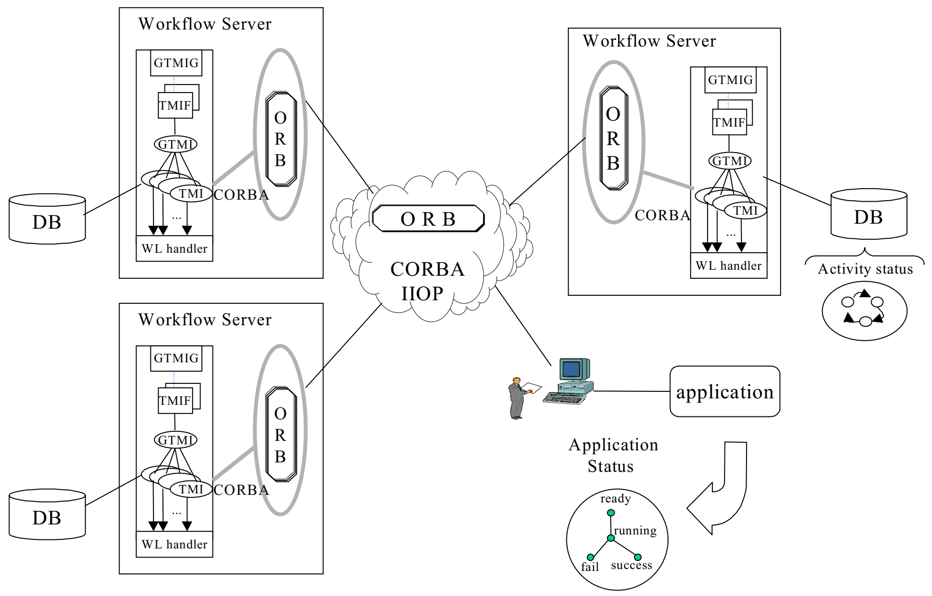


Figure 7. Run Time Architecture of ICU/COWS

through TMIF which is installed on each workflow server. When the designated server of a TMI crashes, the other server is selected instead for the TMI creation. We found that this creation method is very flexible and effective in achieving high availability and scalability of the workflow system. Figure 7 shows the run time architecture of ICU/COWS.

3.1.4 Application Servers and TMI Placement.

It is advantageous that workflow servers reside in the same or close site to the application servers processing the workflow instance. However, in large enterprises, a long running workflow instances may need services from several application servers which are located on physically different sites. Thus, when multiple servers operate in a distributed fashion they need to be deployed considering the location of application servers and the TMI generation should be conducted in the same manner. That is, the TMI, which invokes an application program that requires services from an application server, have to be created in the workflow server which is the same or close to the application server.

3.1.5 Worklist handler and Client

The worklist handler plays a role in bridging TMIs and clients. Only one worklist handler can exist in a workflow server. However, multiple worklist handlers can exist in the overall system. TMI sends work items to worklist handlers in a push mode and the worklist handler sends the work items to the corresponding clients in the same manner. Although a worklist handler can be preferred by a TMI or a client, there is no need for a worklist handler to be dedicated to a certain TMI or client. That is, a TMI can be connected to any worklist handler to send work items to clients and a client can be connected to any worklist handler to receive work items. Several worklist handlers can retain different worklists for a client but the worklists for a client are usually maintained by the primary worklist handler. Figure 8 shows the conceptual view of these connections.

Since a new worklist handler to the system can be added by changing slightly database and the crash of a worklist handler does not imply disconnection of the TMI from the client, the

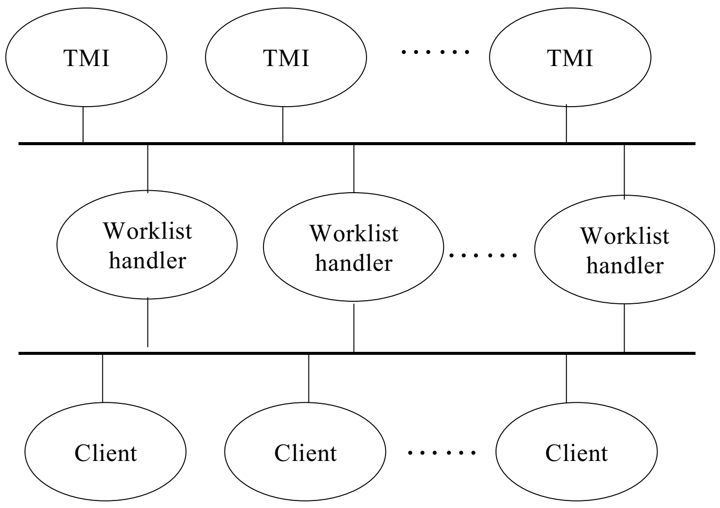


Figure 8. Conceptual view of connections among TMIs, Worklist handlers, and Clients

system is very scalable and resilient.

3.2 Support of Multiple Workflow Architectures

In this subsection, we explain how ICU/COWS supports multiple workflow system architectures. Since most of the basic ideas have already been explained in the previous sections, we only comment how the user requests are processed in ICU/COWS. When a creation of control/monitoring oriented workflow instance is requested, GTMIG selects a workflow server in which all the TMIs are generated using the TMIF. The selection of a workflow server is under studying currently.

If requested workflow type is application server oriented, GTMIG searches feasible workflow servers and creates TMIs in the selected workflow servers in a distributed fashion. The decision of which TMI should be created in which workflow server can be guided by process designers at design time. But if there is no guide input from the process designer, GTMIG decides the workflow server based on the physical closeness to the application server.

If the requested workflow type is mobile client oriented, GTMIG creates TMIs on client sites. In case when the client is not logged in yet, GTMIG asks the logging module to create the TMI when the client is logged in on behalf of it. Note that the TMIs for mobile environment should be designed to manage the situations like the TMI is disconnected with the workflow server. More deliberate TMI design is required. We do not describe the details of the design which is out of scope of this paper. In that way ICU/COWS changes its architecture according to the input workflow types.

4. Performance Model of Workflow System Architectures

The performance model of this paper is based on the workflow system introduced in the previous section. To understand the performance of each workflow system architecture, we need to define the performance metrics of each architecture. The expected execution time of a process instance is the major barometer for the performance evaluation. A process instance is created before it is processed. Then the created process instance is processed according to the defined path until it reaches the end point. Thus the total execution time of a process instance can be defined as the equation (1)

$$TPIET = PICT + PIET \tag{1}$$

where *TPIET* is total process instance execution time and *PICT* is process instance creation time and *PIET* is process instance execution time.

4.1 Process Instance Creation Time

Basically the *PICT* is inverse proportion to the system load which is dependent on the inter process instance creation request time. Thus we can represent *PICT* for centralized workflow system architecture as the equation (2-1)

$$PICT_c = n * local_t / \lambda \tag{2-1}$$

where $local_t$ is the time to create a local TMI, n is the number of TMIs in the process instance and λ is the inter process instance creation request time.

In the decentralized workflow system, all the TMIs of a process instance are created in a server and the creation time of TMIs is divided by the number of multiple workflow servers. Thus PICT for decentralized workflow system is modeled as the equation (2-2)

$$PICT_m = (n * local_t) / (m * \lambda) \tag{2-2}$$

where m is the total number of workflow servers in the decentralized workflow system.

In the distributed workflow system, TMIs of a process instance can be created both in local server and remote servers arbitrarily. Thus *PICT* for distributed workflow system is modeled by the equation (2-3)

$$PICT_d = [\ n * (p * local_t + (1-p) * remote_t\ )\ ] / (m * \lambda) \tag{2-3}$$

where p is the probability that TMIs of a process instance is created in the local workflow server and $remote_t$ is the time to create a remote TMI.

In fully distributed workflow system, all the TMIs are created in the client site. Thus if we assume that at most one TMI of a process instance is created in a client site, we can represent *PICT* for fully distributed workflow system as

$$PICT_f = n * remote_t / (n * \lambda) + n * \alpha = remote_t / \lambda + n * \alpha \tag{2-4}$$

where α is the coordination overhead to create TMIs in client sites.

4.2 Process Instance Execution Time

After a process instance is created, the process instance is processed getting services from workflow system. The process instance execution time covers the span of the time from the start to the end of a process instance. If there is no parallel execution during the processing we can define the process instance execution time as

$$PIET = \Sigma_{AI(k) \in PI}\, AI_kET \tag{3}$$

where *PIET* is process instance execution time, $AI(k)$ is the k-th activity instance in the process of the process instance *PI*, and AI_kET is the execution time of k-th activity instance.

When we contemplate the AI_kET, we can denote it by the equation (4) because the total execution time of AI_kET is the summation of the time spent by workflow system and user or workflow system and application respectively according to the type of tasks.

$$\begin{aligned} AI_kET &= ST + UT \text{ (Manual Task)} \quad \text{or} \\ &\ ST + AT \text{ (Automatic Task)} \end{aligned} \tag{4}$$

where *ST* is system time, *UT* is user time, and *AT* is application time. User time is the time spent by workflow participants to complete the work item delivered to him/her and application time is the time spent by the application program invoked by the k-th activity. Since the user time and application time are independent from the workflow system, we only take into account system time in this paper. System time is the time spent by the workflow system to complete the activity. The system time includes the time spent in sending work item from workflow engine to clients and receiving the complete signal from clients to workflow engine etc. For a work item to be delivered from workflow engine to clients, it has to be relayed by TMI and worklist handler. Thus the relative location of TMIs, worklist handlers, clients and application servers should be reflected in the system time. Taking the fact into account, we represent the system time as the equation (5)

$$ST = (S_t + TW + WC + CW + WT + TT) / \lambda \text{ (Manual Task) or}$$
$$(S_t + TW + WA + AW + WT + TT) / \lambda \text{ (Automatic Task)} \qquad (5)$$

where S_t is the time spent by workflow system to service the execution of the activity except the time spent in sending signals among TMIs, worklist handlers, clients, and applications. Repository service time for worklist handler and history information maintenance is included in the S_t. Thus S_t could be a fixed value independent of workflow system architectures. *TW* is the time to signal from TMI to worklist handler and *WA* is the time to signal worklist handler to application. The other notation can be interpreted in the same way. If we do not distinguish the direction of the signaling, we can simplify the equation of automatic task as the equation (6)

$$ST = (S_t + 2TW + 2WA + TT) / \lambda \qquad (6)$$

Again, the time of signaling between the objects in the same machine and the objects in the different machines is not the same. Thus we can denote the system time of centralized workflow system as the equation (6-1) because all the objects except application servers are in a machine in centralized workflow system.

$$ST_c = [S_t + 2TW_{local} + 2[\, p_0 * WA_{local} + (1-p_0) * WA_{remote}] + TT_{local}] / \lambda \qquad (6\text{-}1)$$

where TW_{local} and TT_{local} is the time to signal between objects in the same machine. WAremote is the time to signal between objects in the different machines. In decentralized workflow system, multiple servers work together where each server works in centralized way. Thus the equation (6-1) can be changed to equation (6-2) in decentralized workflow system.

$$ST_m = [S_t + 2TW_{local} + 2[\, p_0 * WA_{local} + (1-p_0) * WA_{remote}] + TT_{local}] / (m * \lambda) \qquad (6\text{-}2)$$

where m is the number of servers in the decentralized workflow system.

In distributed workflow system, TMIs and worklist handlers can be located in remote machines, the system time equation is denoted as the equation (6-3)

$$ST_d = [S_t + 2[\, p_0 * TW_{local} + (1-p_0) * TW_{remote}]$$
$$+ 2[\, p_1 * WA_{local} + (1-p_1) * WA_{remote}]$$
$$+ 2[\, p_2 * TT_{local} + (1-p_2) * TT_{remote}]] / (m * \lambda) \qquad (6\text{-}3)$$

In fully distributed system, TMIs and worklist handlers are located in the same site and the application server is located in remote site from client site. Thus the system time equation is expressed by the equation (6-4)

$$ST_f = [S_t + 2TW_{local} + 2WA_{remote} + TT_{local}] / (n * \lambda) \qquad (6\text{-}4)$$

where n is the number of activities in the process instance.

From the equation (5), we can simplify the equation of manual task by the equation (7) like the automatic task

$$ST = (S_t + 2TW + 2WC + TT) / \lambda \tag{7}$$

The system time of each architecture is defined as

$$ST_c = (S_t + 2TW_{local} + 2WC_{remote} + TT_{local}) / \lambda \tag{7-1}$$
$$ST_m = [S_t + 2TW_{local} + 2WC_{remote} + TT_{local}] / (m * \lambda) \tag{7-2}$$
$$ST_d = [S_t + 2[p_0 * TW_{local} + (1-p_0) * TW_{remote}] + 2WC_{remote} + 2[p_2 * TT_{local} + (1-p_2) * TT_{remote}]] / (m * \lambda) \tag{7-3}$$
$$ST_f = [S_t + 2TW_{local} + 2WC_{local} + TT_{local}] / (n * \lambda) \tag{7-4}$$

respectively.

4.3 Performance Analysis

In this section, we analyze the performance of workflow system architectures based on the analytic model defined in the previous subsection. For the analysis, we assume that the following conditions are given. First, process instance creation request is invoked every second and the process instance contains 100 sequential activities. That is, the value of λ is 1 and the value of n is 100. Second, 4 servers are used to construct decentralized and distributed workflow system. That is, the value of m is 4. Third, the relative time ratio k of $remote_t/local_t$ is dependent on the environment. The measured value is about 2 to 4 in the SUN ULTRA 10 servers connected by 10 Mbps Ethernet network environment. Fourth, we assume that the relative time ratio $\alpha/local_t$ is 0.2. Fifth, the relative time ratio of TW_{remote}/TW_{local} is about 2.5 which is also measured value in the same environment.

Figure 9 shows the change of *PICT* as the ratio of $local_t/remote_t$ increases. For the distributed workflow system, three cases with the values of p are 0.25, 0.50, 0.75 are evaluated in *PICT* expectation. While the load of system does not influence to the time of $remote_t$ so much, the time of $local_t$ increases drastically as the system is overloaded. Thus the heavier the system load is, the bigger the value of $local_t/remote_t$. *PICT* of decentralized workflow system always

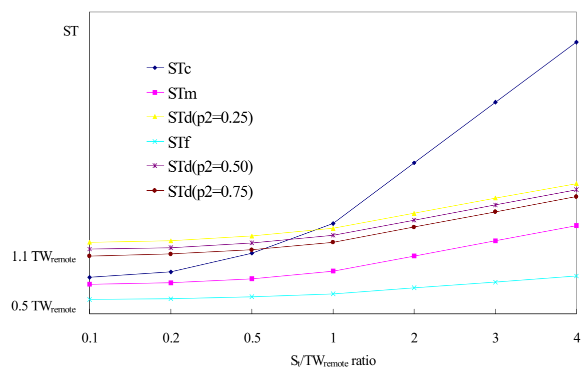


Figure 9. Process Instance Creation Time

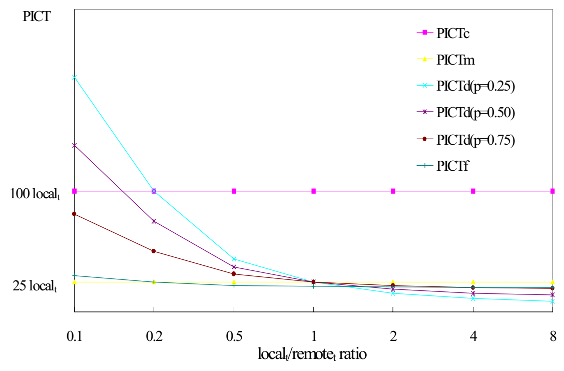


Figure 10. System Time of Process Instance

outperforms that of centralized workflow system because the work is divided in decentralized system. *PICT* of distributed workflow system depends on the degree of distribution of the generated TMIs. When the system load is light, the multiple server effect of distributed workflow system does not appear, but when the system load is heavy, *PICT* of distributed workflow system shows better performance than that of the centralized workflow system. In the fully distributed workflow system, the influence of the system load is very weak as expected.

Figure 10 shows the change of system time(*ST*) in *PIET* as the ratio of $S_t/remote_t$ increases. As system load does not influence on the time of $remote_t$ so much and the time of S_t increases drastically when the system is overloaded, we may assume that the heavier the system load is, the bigger the value of $S_t/remote_t$. For the decentralized workflow system, the value of p_0 is set to 0.25, that is, 75% of the application invocation requires remote accesses. For the distributed workflow system, the values of p_0 and p_1 are set to 0.25 as the decentralized workflow system and three cases with the values of p_2 are 0.25, 0.50, 0.75 are evaluated for the system time of *PIET* expectation. *ST* of decentralized workflow system always outperforms that of the centralized workflow system because of the same reason that *PICT* of decentralized workflow system outperforms that of the centralized system. *ST* of distributed workflow system is dependent on the degree of distribution of the generated TMIs and system loads. When the system load is light, the multiple server effect of distributed workflow system does not appear as the case of *PICT*, but when the system is overloaded *PICT* of distributed workflow system shows better performance than that of the centralized workflow system. In the fully distributed workflow system, the influence of the system load is very weak as the case of *PICT*.

Although the results of the evaluation reveal the overall characteristics of each workflow system architectures, it still has some limitations because the model does not take into account the network conditions sufficiently. The network conditions are only reflected on the value of the $local_t/remote_t$, $S_t/remote_t$, and TW_{remote}/TW_{local} in the model and we fixed the value for the evaluation. However the value should be changed in different network conditions. We need to compare the result in this paper with the real data obtained from the real workflow system to decide whether more deliberate model is required or not and which will be our future work.

5. Related Work

Considerable work has been done on the research of the workflow (Alonso 1995, Ellis 1995, Han 1996, Kim 1997 et al.). But most workflow systems fall into the centralized monolithic workflow system. Recently various workflow system architectures are proposed to accommodate various kinds of workflows. However few works have been done on the performance model of workflow system architectures. There are several papers evaluating the performance of distributed objects. But they are still in infant stage to apply the result to estimate the workflow system performance based on their result.

The performance model developed in this paper can be applied to several other well known distributed workflow systems because the sample workflow system introduced in this paper has very general structure. ORBWork(Das 1997, Wang 1995) is one of the well known distributed workflow system on CORBA environment for METEOR2 workflow model. They organize the system with fixed task managers which manage the tasks that is assigned to the task manager specified by the IDL interface(Wang 1995, Miller 1996). The task managers and tasks of ORBWork correspond to TMI and tasks respectively of our system.

The result of the work of Nortel(Nortel & University of Newcastle upon Tyne 1998, Parrington 1998) is also very close to our system. Their main design goals are to achieve interoperability, scalability, flexible task composition, dependability, and dynamic reconfiguration. They have tasks and task controllers which are implemented as the CORBA transactional objects. The task controller corresponds to our TMI but they do not have a central controller that manages the whole workflow like our GTMI. They may be able to change the workflow system architectures like our system but it seems that they are not noticed of this aspect.

6. Conclusion

Various kinds of workflow types exist in the real business world. Control/monitoring oriented, application server oriented and mobile oriented workflow types have been proposed according to their characteristics and four different workflow architectures are also explained in terms of the placement of execution objects for tasks. Since various workflow architectures exist, a workflow system architecture has to be selected from them to fit certain environment. Performance model of workflow system could be a good criterion for the selection.

In this paper we proposed a performance model for workflow systems. Performance tests for workflow system architectures have been performed with the performance model. From the results, we can confirm that the performance of distributed workflow system is very condition dependent while the performance of decentralized workflow system almost always shows good performance. Of course this does not imply decentralized workflow system is better than distributed workflow system. Rather, this means that when you choose distributed workflow system architecture, you should carefully examine the conditions and situation.

In the future, we plan to examine the test result of the performance model with the simulation result of the queuing model of workflow system or the real data obtained from real workflow systems.

References

Alonso, G., Günthör, R., Kamath, M., Agrawal, D., Abbadi, A. E., and Mohan, C. "Exotica/FMDC: Handling Disconnected Clients in a Workflow Management System," *In Third International Conference on Cooperative Information Systems (CoopIS-95),* May 1995.

Das, S. "ORB Work: A Distributed CORBA-based Engine for the METEOR2 Workflow Management System," *Master's thesis*, University of Georgia, Athens, GA, March 1997.

Das, S., Kochut, K., Miller, J., Seth, A., and Worah, D. "ORBWork: A reliable distributed CORBA-based workflow enactment system for METEOR2," *Technical Report,* Dept. of Computer Science, University of Georgia, UGA-CS-TR 97-001, 1997.

Ellis, C. A., Keddara, K., and Rozenberg, G., "Dynamic Change within Workflow Systems," *Proceedings of the ACM SIGOIS Conference on Organizational Computing Systems*, 1995, CA.: Milpitas.

Han, D. S., Shim, J. Y., and Yu, C. S. "ICU/COWS: A Distributed Transactional Workflow System Supporting Multiple Workflow Types," *IEICE Transactions of Information and Systems* (accepted).

Kim, K. H., and Ellis, C. A. "A Framework for Workflow Architectures," *Technical Reports*, Dept of Computer Science, University of Colorado, CU-CS-847-97, December 1997.

Miller, J. A., Sheth, A. P., Kochut, K. J., and Wang, X. *CORBA-based Run-Time Architectures for Workflow Management Systems,* [Dog96], (7:1), Winter 1996, pp. 16-27.

Nortel & University of Newcastle upon Tyne, *Workflow Management Facility Specification*, Revised Submission, OMG Document Number: bom/98-03-01, 1998.

Parrington, G. D., Shrivastava, S. K., Wheater, S. M., and Little, M. C. "The design and implemented of Arjuna," *USENIX Computing Systems journal* (8:3), 1998, pp. 255-308.

Paul, S., Park, E., and Chaar, J. "RainMan: a Workflow System for the Internet," *Proc. Of USENIX Symp. On Internet Technologies and Systems*, 1997.

Silver, B. R. "The BIS Guide to Workflow Software: A Visual Comparison of Today's Leading Products," *Technical report,* BIS Strategic Decisions, MA: Norwell, MA, September 1995.

Wang, X. "Implementation and Performance Evaluation of CORBABased Centralized Workflow Schedulers," *Master's thesis*, University of Georgia, August 1995.

Workflow Management Coalition Specification Document, *The Workflow Reference Model*, Version 1.1, November 1994.