다중 워크플로우 시스템 구조를 포용하는 인터넷 기반 워크플로우 시스템

(An Internet Based Workflow System Covering Multiple Workflow System Architectures)

한 동 수 † 심 재 용 ††

(Dongsoo Han) (Jaeyong Shim)

요 약 인터넷상의 워크플로우를 위한 최적의 워크플로우 시스템은 해당 워크플로우 시스템에 입력되는 워크플로우 종류에 의존적이다. 워크플로우 시스템이 처리해야 하는 워크플로우 종류가 다양하고 이들에 가장 적합한 워크플로우 시스템의 구조가 각각 서로 다르다고 볼 때, 하나의 워크플로우 시스템이 다양한 종류의 워크플로우 시스템 구조를 포용하고 입력되는 워크플로우 종류에 따라서 적절히 그것의 구조를 변경하는 것이 이상적이라고 할 수 있다. 본 논문에서는 다양한 워크플로우 시스템 구조를 포용하는 워크플로우 시스템 구조를 제안한다. 제안된 시스템에서는 태스크들을 위한 수행 객체를 객체 인스턴스 형태로 생성하고 이들을 시스템이 원하는 임의의 위치에 배치함으로써 그것의 구조를 변경할 수 있다. 초기의 시스템에 대해서 그 구조를 변경하면서 성능 평가를 수행하였으며 그 결과가 제시되었다. 제시된 결과에 의하면 인터넷상에 분산된 복수의 워크플로우 서버는 분산된 태스크에 대한 안정적인 운영 외에도 전체 시스템의 성능을 향상시키는 데에도 도움이 되는 것으로 확인되었다.

Abstract The best-fit workflow management system architecture for the workflows in the internet is dependent on the workflow types that should be processed in the workflow system. Since a workflow management system should accommodate various kinds of workflows requiring different workflow system architectures respectively as their best-fit workflow system architectures, it is ideal that a workflow system covers various workflow system architectures and changes its architecture according to the input workflow types. In this paper we propose a workflow system architecture that covers various workflow architectures within a single workflow system. The system changes its architecture by creating an execution object for a task in the form of an object instance and placing the created object instance to anywhere the system wants to. Performance test has been performed on the early versions of the system changing its architectures, and the results are illustrated. The results show that the distributed multiple workflow servers in the internet can contribute to not only reliable control of the distributed tasks but also enhancing total throughput of a workflow system.

1. Preface

Since the needs of workflow system had been invoked numerous workflow systems have been developed for either commercial or academic purposes [1, 6, 7, 8, 9, 10, 11]. Developed early workflow systems have the inclination to focus on equipping functions and facilities that the workflow system should be supposed to provide. As a result relatively less study has been performed on the influence of workflow system in architectural aspects. Centralized *monolithic workflow system has mainly been adopted* as workflow architecture in the community[1, 9].

Recently, the number of computers connected to the internet is increasing drastically and the

† 종신회원 : 한국정보통신대학원대학교 정보공학부 교수
dshan@icu.ac.kr
†† 비 회 원 : 한국정보통신대학원대학교 정보공학부
jaeyong7@icu.ac.kr
논문접수 : 1999년 12월 16일
심사완료 : 2000년 3월 23일

computing environment is changing rapidly. It became possible that any computers or users connected to the internet form a group to process certain workflow. However for the workflow systems to cover the various workflows on the internet, they should be much more scalable and flexible as well. Centralized workflow system is often reported that it has limitation to support workflow on the internet and distributed workflow systems have been suggested instead.

In fact there are many kinds of workflow and the best fit workflow architecture for each kind of workflow differs from one another. Thus it is hard to be insisted that one is better than the other. Rather a workflow system is better to cover various workflow architectures and change its architecture according to input workflow arbitrarily because a workflow system should accommodate workflows which require different workflow architectures as best fit workflow architectures respectively.

Distributed workflow systems [2, 3, 12] can be considered to cover centralized workflow system in a sense because it can be operated as centralized workflow system. But most distributed workflow systems are not always able to change its architecture at will according to input workflow. Moreover the range of distributed workflow system is so various that even one kind of distributed workflow system often cannot cover other kind of distributed workflow system. Since a workflow system should accommodate all kinds of workflow and each workflow type matches well to certain workflow architecture, it is ideal that a workflow system covers various workflow architectures and changes its architecture to the type of input workflows.

In this paper we propose and implement a workflow system which covers various workflow architectures within a single workflow system. The system changes its architecture by creating an execution object for a task in the form of object instance and placing the created objects instance to anywhere the system wants. Simple object instance generator needs to be prepared in the site a prior to the creation. The generator can be installed fairly easily with the help of Web browser. As input workflow types, control/monitoring oriented, application server oriented and mobile oriented workflow types have been proposed according to their characteristics and four different workflow architectures are introduced in the point of placement of the execution objects for tasks.

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 the proposed system are described. In section 4, we show the performance result of each architecture on the early version of the system. Related work is 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, there are workflow processes requiring hard real time process controls and monitoring services. Workflow processes in manufacturing factory often fall into this area. On the other hand some workflow processes do not require such a strong process controls and monitoring services. Rather they often prefer the process control in which each task manager react autonomously and independently. Inter-company workflow processes or

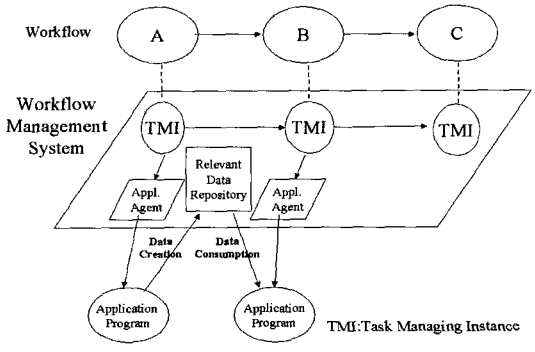

Fig. 1 Run Time Architecture of Workflow Enactment Service

mobile salesman involved workflow processes are the typical example of this type of workflow. Workflow processes in mobile environment can be supported more effectively by autonomous and independent task managers. We call the former *control/monitoring oriented workflow processes* and the latter *autonomy* oriented workflow processes.

In some other workflow processes, the relation between 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 near to the application server. We call the workflow processes *application server* oriented workflow processes. Workflow processes involving legacy system's applications can be considered one of the typical *application server* oriented workflow processes.

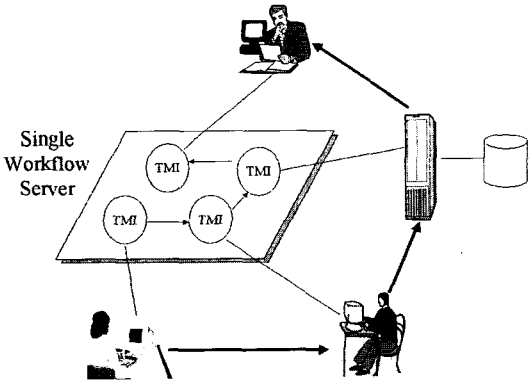

Fig. 2 Centralized Workflow System

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 modeled in Figure 1. The execution object in the Figure 1 manages an activity step from the invocation of the corresponding task to the end of the task. When the task is over it signals to 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 by how we implement the execution objects. They can be implemented in a monolithic module as a whole or several modules can corporate one another to provide the services. Even it is possible that each execution object is implemented as a transient object instance. In that case, workflow sever is 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 way of placement 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 in one central workflow server. Thus the system is very simple since the TMI creation is relatively simple when compared with other kind of workflow systems. 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 being created in the same workflow server. Decentralized workflow system architecture can be considered as one kind of multi-workflow server implementation. Figure 3 shows the run time structure of the decentralized workflow system.

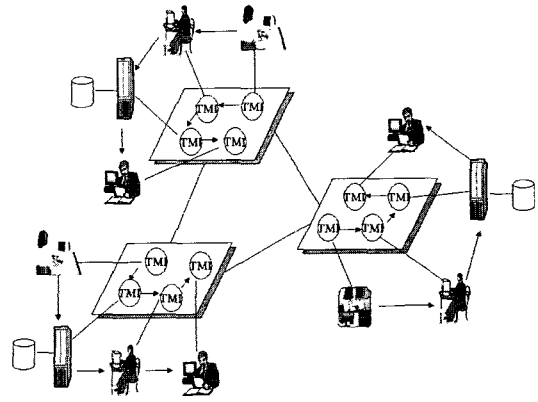

Fig. 3 Decentralized Workflow System

Distributed workflow system is the same with decentralized workflow system in that it places the created TMIs to 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.

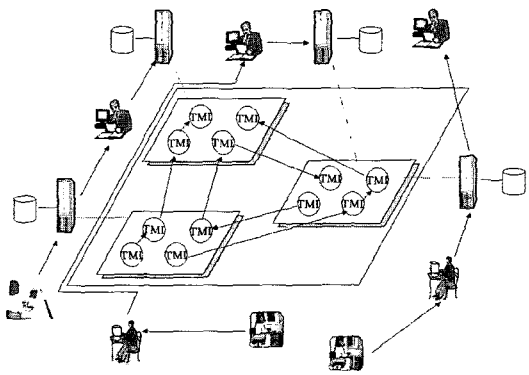

Fig. 4 Distributed Workflow System

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. However once the first TMI start to manage its task, the control of the workflow process is handed over to the TMIs in the clients. The created TMIs in the clients can act and react to its tasks autonomously and independently from the server and there is no control from workflow server after the first TMI start to work. Thus fully distributed workflow system architecture can cope with mobile environment because in the mobile environment, autonomous and independent control for mobile clients are often required. Figure 5 shows the workflow system architecture of fully distributed workflow system.

2.4 Comparison of Workflow 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 one server. Control/monitoring oriented workflow types are well suited to centralized workflow system. However the system inevitably incurs bottleneck when the number of process instances grows over the threshold value where 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 the jobs into the servers. Since all the TMIs for one workflow process instance are generated in one server, the servers need not to interact each other to hand over controls between TMIs. They only have to interact for gathering monitoring or historic information on 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 one server could be a barrier to service various workflow types effectively. For instance decentralized workflow system cannot cope with the situation when some distributed legacy application servers involved in one process instance prefer TMIs to be placed near to them.

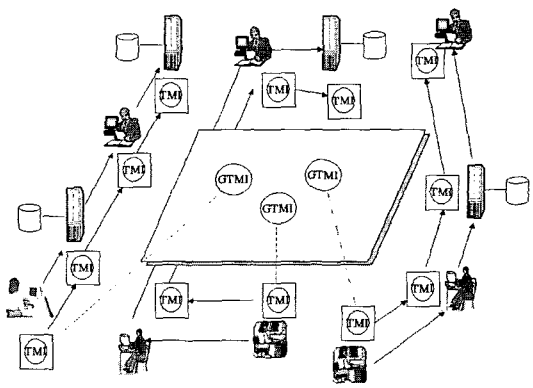

Fig. 5 Fully Distributed Workflow System

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

management 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 are dependent on the way of TMI placement. Thus desirable TMI placement strategy should be devised and deployed in distributed workflow. However distributed workflow system has also limitation on supporting mobile environment where the network disconnection is frequent.

	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

Fig. 6 Comparison Matrix of each Workflow Architecture

In fully distributed workflow system TMIs are pushed to the client sites so users can still 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 to connected state. Thus the load of centralized workflow servers is drastically reduced. Theoritically tremendous number of workflow instances can be accommodated in fully distributed workflow system. But practically the cost of monitoring and system consistency maintenance could be very high in fully distributed workflow system. Figure 6 is summing up the comparison of these workflow types.

3. Software Architecture of ICU/COWS

In this section we describe the design goals and software architecture of ICU/COWS(Connector Oriented Workflow System)[15] and then we explain how the system can support multiple workflow architectures.

3.1 Design Goals

As mentioned in the previous sections, the primary aim of the target system is to support multiple workflow system architectures within a single workflow system structure. In the system, process designer marks an appropriate workflow type when designing a process template. Which can be considered as the selection of workflow architecture in process template level. Users can also select workflow architecture in process instance level before he starts a process instance. In addition the target system is designed to satisfy the following design goals such as availability, scalability, reliability and dynamic workflow reconfiguration.

3.2 Software Architecture

This section describes the system architecture and the components of ICU/COWS which is designed to support multiple workflow system architectures within the system. 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 on it. Details about the WTS specifications will be treated in other papers in the near future. The components of ICU/COWS include TMIF(Task Managing Instance Factory), GTMIG(Global Task Managing Instance Generator), Simulator, Process

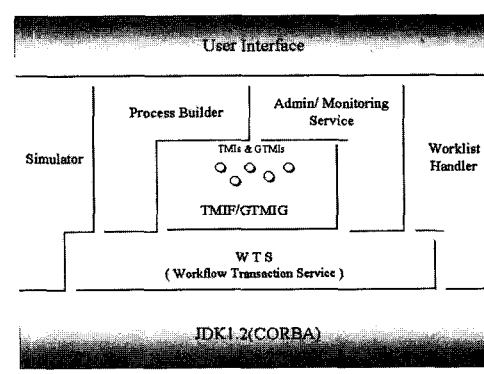

Fig. 7 Software Architecture of ICU/COWS

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. Figure 7 shows the software architecture of ICU/COWS.

3.2.1 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 by 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 processing 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 the network disconnection is frequent.

3.2.2 GTMIG and TMIF

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 is resident on 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 considering the system configuration as described in the subsection 4.1.4. When one server is down, the GTMI searches an alternative server and uses the selected server on behalf of the crashed server. In this way, the whole system can maintain high system availability irrespective of system failures.

3.2.3 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:

1. A user asks to create a certain process instance.
2. GTMIG creates a GTMI for the process instance through TMIF.
3. GTMI creates all TMIs of the process instance through TMIF.
4. GTMI sends a start signal to the first TMI to start work.
5. TMI starts work and sends a start signal to the next TMI once the work is finished successfully.
6. Iterate step 5 until the last TMI is reached.
7. The last TMI sends an end signal to the GTMI when it finishes.
8. GTMI destroys all the TMIs generated and itself.

Both GTMIG and TMIF are resident 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 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 8 shows the run time architecture of ICU/COWS.

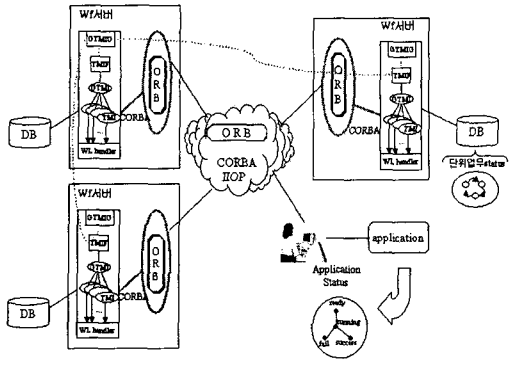

Fig. 8 Run Time Architecture of ICU/COWS

3.2.4 Application Servers and TMI Placement

It is advantageous when workflow servers are resident 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, a TMI which invokes an application program that requires services from an application server, have to be created in the workflow server that is the same or close site to the application servers.

3.2.5 Worklist handler and Client

The worklist handler does a role bridging between 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 9 shows the conceptual view of these connections. Since the addition of a new worklist handler to the system can be achieved by a slight change of a database and the crash of a worklist handler does not imply disconnection of the TMI from the client, the system is very scalable and resilient. In the following sections, we describe the functions of each component in more detail.

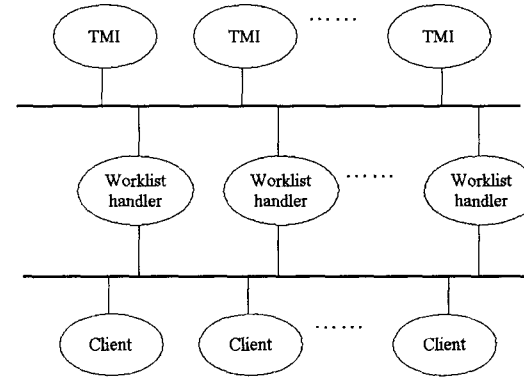

Fig. 9 Conceptual view of Connection among TMIs, Worklist Handler and Client

3.3 Support of Multiple Workflow Architectures

In this subsection, we explain how ICU/COWS supports multiple workflow architectures. Since ICU/COWS can create TMIs on any sites that the TMIF module is prepared, the deployment of TMIFs roughly decides the workflow system architectures that ICU/COWS can cover. If only one TMIF module is deployed in a workflow server, ICU/COWS can support only centralized workflow system architectures. If TMIF modules are deployed on multiple workflow servers, ICU/COWS can support decentralized or distributed workflow system architectures also. Whether ICU/COWS supports decentralized or distributed workflow system architectures is decided by its choice of the placement of TMIs for one process instance. To support fully distributed workflow system TMIF modules should

be deployed on all the client sites. Which is usually implemented with the help of Web browsers.

Now we explain how the user request for workflow architecture is processed in ICU/COWS. When the creation of control/monitoring oriented workflow instance has been requested, GTMIG selects a workflow server and all the TMIs are generated in the server using the TMIF. The selection of a workflow server is determined by which GTMIG is involved because there is only one GTMIG in each workflow server. If the requested workflow type is application server oriented, GTMIG searches feasible workflow servers and creates TMIs on the selected workflow servers in a distributed fashion. The decision of which TMI should be created on which workflow server can be guided by process designer at design time. But if there is no guide input from the process designer, GTMIG decides the workflow server based on the physically close site of the TMI to the application server.

If the requested workflow type is mobile client oriented, GTMIG creates TMIs on client sites. But the GTMIG should be careful when the client is not logged in yet because it cannot create TMIs for the client immediately. In that case GTMIG asks the logging module to create the TMI as soon as the client get logged in on behalf of it. For the created TMIs in client sites but not processed before the client computer turned off, some recovery mechanism should be devised so that it can recover to the original state. That is, TMIs for mobile environment should be designed to manage to the situation when the TMI is disconnected with the workflow server. More deliberate TMI design is required. We do not described the details of the design which is out of scope of this paper.

4. Performance Test of Each Architecture

Although we analyzed the characteristics of each architecture in section 2, quantitative performance value of each architecture could be another interesting aspect comparing each architectures. In this section, we illustrate the performance results evaluated on the early version of the proposed system. Early version of the system has been developed on the JDK1.2 environment using basic CORBA naming services. Several tests on the workflow servers of different machines connected with the 10Mbps LAN in our laboratory have been performed to factor out the characteristics of the system. Although the result of the tests may not represent the system characteristics exactly and the result may be changed as the additional features are added to the system, some important characteristics of each workflow system architecture specially focused on distributed workflow systems have been made out.

4.1 Effects of Multiple Workflow Servers

Since multiple workflow servers collaborate with one another in the distributed workflow system, they may enhance the total performance of the workflow system. We tested the performance effect of the multiple workflow servers mainly in the user viewpoint. While in the system viewpoint the throughput that is the number of jobs processed per second should be measured, in the user viewpoint the average response time of the system reacting to the user requests is measured. 100 workflow instances with 300 activities connected sequentially are generated every 10 seconds and we assumed that each activity step takes 1 second to complete its work. Currently once the invocation of a workflow instance is requested to the workflow system, one of

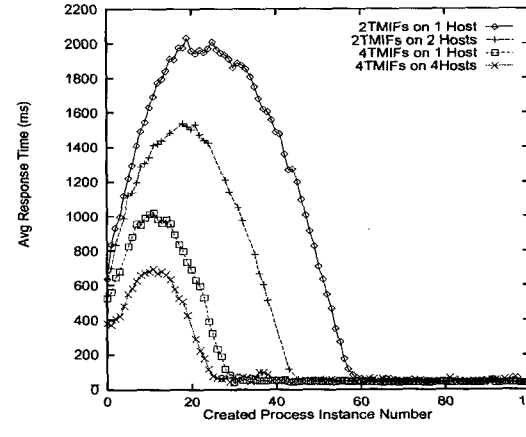

Fig. 10 Average Response Time of Each Activity Step

the workflow server received the request creates all the TMIs(in this case 300 TMIs) to control each activity. The average response time of the created process instances has been measured on 1 to 4 hosts with one or more TMIFs in each server. The Figure 10 shows the performance result of the test. 4 server system showed 1.5 to 3 times better processing power than the single workflow system in congestion situation. From the performance result, we can see that the number of hosts and the number of TMIFs are the sensitive factors for performance enhancement of the workflow system.

4.2 Effects of the TMI Distribution Methods

Three different ways of TMI distribution is applied to analyze the effect of each method. In fully distributed TMI placement which is called worst grouping method, the next TMI of a TMI is always placed on the different server from the TMI. Thus this placement method inevitably incurs network traffics when a TMI completes and notify it to the next TMI. In chunk TMI distribution method we call it best grouping method, TMIs are divided evenly to the severs and all the TMIs' neighbor TMI are placed in the same server except the first and the last TMIs. Therefore the completion of a TMI does not incur network traffics to notify its completion to the next TMI except the last TMI in the server. In random TMI distribution method, TMI placement is decided randomly, therefore the network overhead is can be considered to be medium when compared with the previous methods. Figure 11 shows the throughput change according to TMI grouping methods. As expected the best grouping method showed 3 times better performance result than worst grouping method when the number of concurrent instances reaches 100.

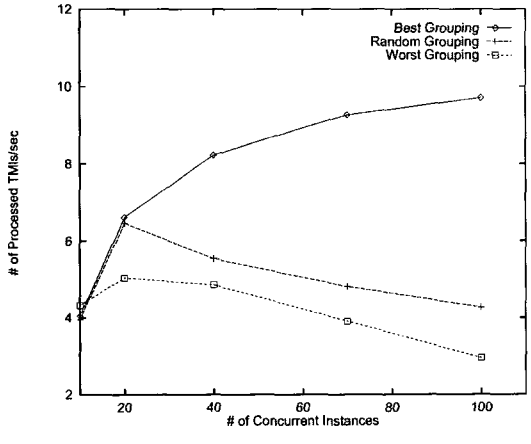

Fig. 11 System Throughput Variation According to TMI Grouping Methods

4.3 Effects of TMI Creation Methods

Two kinds of TMI creation method are tested. In compile time TMI creation, all the TMIs are created before the first one starts. Thus compile time TMI creation does not have TMI creation overhead at all once the process instance is started. However compile time TMI creation may create TMIs that will not be executed at all. In run time TMI creation only the TMIs that will be executed are created right before the execution. Therefore the TMI creation overhead is included in the run time overhead of a process instance. This may become serious overhead in production workflow system. Figure 12 compares the response time of the two methods. As can be seen from the Figure, the overhead of the run time TMI creation method is considerable. Thus for time critical production workflow compile time TMI creation method is more recommended.

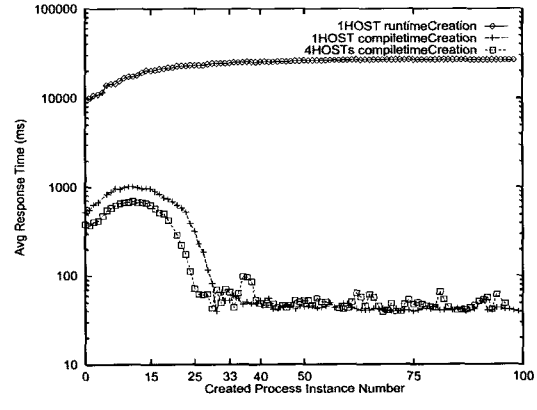

Fig. 12 Comparison of Average Response Time between Compile Time and Run Time TMI Creation

4.4 Effects of Workflow Server Distribution

The configuration of workflow servers also another

factor to the performance results. For example, when multiple servers are connected in WAN environment the network overhead will be much more obvious. However when the workflow servers are resident in clustering environment where each node is connected with more fast networks such as 150Mbps, contrary result to the WAN environment will be resulted. We are planning to test the performance in these environments in the near future.

5. Related Works

Considerable work has been done on the research of the workflow system [1, 6, 7, 8, 9, 10, 11]. But most workflow systems fall into the centralized monolithic workflow system. Relatively few works have been done on distributed workflow system. ORBWork [4, 13] is one of the well known distributed workflow system on CORBA environment for METEOR2 workflow model. They orplacementnize the system with fixed task managers which manage the tasks that is assigned to the task manager specified by the IDL interface [13, 14]. The task managers and tasks of ORBWork correspond to TMI and tasks respectively of our system. While the task managers of ORBWork are resident as a workflow server component the TMI is an transient object instance. Thus our system is more reactive and flexible than the ORBWork.

The result of the work of Notel [2, 5] is 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.

RainMan [3] is another distributed workflow system developed in pure Java using Java RMI as primary transport mechanism. They use source-performer model where sources generate service requests and performers execute the service. The service requesters could be workflows and the service providers could be people or applications or workflow system. They try to derive a standard interoperability interface between a wide variety of heterogeneous workflow applications. However the system also does not mention about architectural aspects of workflow system.

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 a workflow system should accommodate all kinds of workflow and each workflow type matches well to certain workflow architecture, it is ideal that a workflow system covers various workflow architectures and changes its architecture for the type of the input workflow.

In this paper we proposed and implemented a workflow system which covers various workflow architecture in a single workflow system. The system can change its architecture because the execution object for a task is in the form of object instance and the object instance can be placed anywhere the system wants if simple object instance generator is prepared. Currently the generator can be installed fairly easily with the help of Web browser.

Early version of the system has been implemented and several tests have been performed to study the characteristics of each workflow architecture. We found that the increment of the number of distributed workflow servers could be used to increase the total workflow system throughput. Response time is get shorted as the number of server increases. The placement of TMIs in distributed workflow servers is another factor influencing the workflow system performance because different TMI displacement implies different network latency involvement.

References

[1] Workflow Management Coalition Specification Document, "The Workflow Reference Model," Version 1.1, November 1994.

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

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

[4] Das S., Kochut K., Miller J., Seth A. and Worah, D., "ORBWork: A reliable distributed CORBA-based workflow enactment system for METEOR2," Tech. Report No. UGA-CS-TR 97-001, Dept. of Computer Science, University of Georgia, 1997.

[5] Parrington G.D., Shrivastava S.K., Wheater S.M., and Little M.C., "The design and implemented of Arjuna," USENIX Computing Systems Journal, Vol. 8(3), pp.255-308, 1998.

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

[7] Clarence A. Ellis, Keddara K and Rozenberg G., "Dynamic Change within Workflow Systems," Proceedings of the ACM SIGOIS Conference on Organizational Computing Systems, Milpitas, CA., 1995.

[8] Kwang-Hoon Kim and Clarence A. Ellis, "A Framework for Workflow Architectures," University of Colorado/Department of Computer Science, Technical Reports, CU-CS-847-97, Dec. 1997.

[9] Dong-Soo Han and Hyang-Jae Park, Design and Implementation of Web Based Business Process Automating HiFlow System," Journal of KISS(C): Computing Practices, Vol. 4, No. 1, Feb. 1998.

[10] G. Alonso and H.J. Schek, "Research Issues in Large Workflow Management Systems," In Proceedings of the NSF workshop on workflow and process automation in information system, Athens, GA, May 1996. URL: http: //LSDIS.cs.uga.edu/edu/activities/NSF-workflow

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

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

[13] X. Wang, "Implementation and Performance Evaluation of CORBA Based Centralized Workflow Schedulers," Master's thesis, University of Georgia, August 1995.

[14] J. A. Miller, A. P. Sheth, K. J. Kochut, and X. Wang. "CORBA-based Run-Time Architectures for Workflow Management Systems," In Journal of Database Management, Vol. 7, No. 1, pp. 16-27, Winter 1996.

[15] Dong-Soo Han, Jae-Yong Shim, "Connector Oriented Workflow System for the Support of Structured Ad hoc Workflow," Proceddings of the Thirty-Third Hawaii International Conference on System Sciences (HICSS-33), January 4-7, 2000, Maui, Hawaii (CD-ROM).

한 동 수

1989년 서울대학교 계산통계학과 학사. 1991년 서울대학교 계산통계학과 석사(전산과학). 1996년 일본 교토대학교 정보공학부 박사(컴퓨터 공학). 1991년 ~ 1992년 (주)삼성전자 연구원. 1996년 4월 ~ 1996년 7월 일본 NEC C&C 연구소 연구원. 1996년 9월 ~ 1997년 10월 (주)현대정보기술 책임연구원. 1997년 11월 ~ 현재 한국정보통신대학원 정보공학부 조교수. 관심분야는 분산 워크플로우 관리 시스템, 병렬 컴파일러, 고성능 컴퓨팅

심 재 용

1994년 서강대학교 전자계산학과 학사. 1996년 서강대학교 전자계산학과 석사. 1998년 3월 ~ 현재 한국정보통신대학원 대학교 공학부 박사과정. 1996년 1월 ~ 1998년 2월 (주)현대정보기술 정보기술 연구소 연구원. 관심분야는 상호연동성과 적응성 특징을 갖는 워크플로우 관리기술, 정확성을 보장하는 워크플로우 관리기술, 분산 객체 기술, 이동컴퓨팅 기술