INFORMATION TECHNOLOGY JOURNAL

Information Technology Journal 12 (19): 4992-4998, 2013
ISSN 1812-5638 / DOI: 10.3923/itj.2013.4992.4998
© 2013 Asian Network for Scientific Information

Flexible Workflow Management through Distributed CORBA Objects

[1,2]Li, Ya, [1] Tong, Xiong and [2]Wang Hai-rui
[1]Faculty of Land and Resource Engineering,
[2]Faculty of Information Engineer and Automation,
Kunming University of Science and Technology, Kunming, 650093, China

Abstract: The study describes an application composition and execution environment implemented as a workflow system that enables sets of inter-related tasks to be carried out and supervised in a flexible manner. Concerning the serious deficiency of flexibility in the current workflow systems, we describe how our workflow system meets the requirements of interoperability, scalability, flexibility, dependability and adaptability. With an additional route engine, the execution path will be adjusted dynamically according to the execution conditions so as to improve the flexibility and dependability of the system. A dynamic register mechanism of domain engines is introduced to improve the scalability and adaptability of the system. Finally we put emphasis on describing process executing sequence of our system. The system is general purpose and open: It has been designed and implemented as a set of CORBA services. The system serves as an example of the use of middleware technologies to provide a fault-tolerant execution environment for long running distributed applications. The system also provides a mechanism for communication of distributed components in order to support inter-organizational WFMS.

Key words: Workflow management, Flexibility, CORBA objects

INTRODUCTION

Workflow Management Systems (WFMS) have attracted much interest as they increase the efficiency of an organization by automating organizational processes. Many WFMS have been developed and successfully used. However, these systems still have some limitations. First, most such applications are rarely built from scratch; rather they are constructed by composing them out of existing applications and protocols. It should therefore be possible to compose an application out of component applications in a uniform manner, irrespective of the languages in which the component applications have been written and the operating systems of the host platforms. Application composition however must take into account individual site autonomy and privacy requirements. Second, the resulting applications can be very complex in structure, containing many temporal and data-flow dependencies between their constituent. First of all, agents inherit three powerful characteristics applications. However, constituent applications must be scheduled to run respecting these dependencies, despite the possibility of intervening processor and network failures. Third, the execution of such an application may take a long time to complete and may contain long periods of inactivity, often due to the constituent applications requiring user interactions. It should be possible therefore to reconfigure an application dynamically because, for example, machines may fail, services may be moved or withdrawn and user requirements may change. Fourth, facilities are required for examining the application's execution history (e.g., to be able to settle and disputes). So, a durable 'audit trail' recording the interactions between component applications needs to be maintained. Taken together, these are challenging requirements to meet.

Bearing the above observations in mind, we present the architecture of FlexFlow workflow system designed and implemented by us. Our system meets the requirements of interoperability, scalability, flexibility, dependability and adaptability implied by the above discussion. It represents a significant departure from these; our system architecture is decentralized and open: it has been designed and implemented as a set of CORBA services.

The study is organized as follows. In Section 2 we give a short overview of FlexFlow workflow system. Section 3 describes the functionalities and the collaborations of our system. Section 4 describes the executing process. We conclude with the results of our research and point out the work which should be done in the future in Section 5.

Corresponding Author: Li, Ya, Faculty of Information Engineer and Automation,
Kunming University of Science and Technology, Kunming, 650093, China

FLEXFLOW SYSTEM

We discuss how our system has been designed to meet the requirements stated earlier, namely: interoperability, scalability, flexibility, dependability and adaptability.

- **Interoperability:** The system has been structured as a set of CORBA services to run on top of a CORBA-compliant ORB thereby supporting interoperability including the incorporation of existing applications
- **Scalability:** There is no reliance on any centralized service that could limit the scalability of workflow applications. A dynamic register mechanism of domain engines is introduced to improve the scalability and adaptability of the system
- **Flexibility:** The system provides a uniform way of composing a complex task out of transactional and non-transactional tasks. This is possible because the system supports a simple yet powerful task model permitting a task to perform application specific input selection (e.g., obtain a given input from one of several sources) and terminate in one of several outcomes, producing distinct outputs. With an additional route engine, the execution path will be adjusted dynamically according to the execution conditions
- **Dependability:** The system has been structured to provide dependability at application level and system level. Support for application level dependability has been provided through flexible task composition mentioned above that enables an application builder to incorporate alternative tasks, compensating tasks, replacement tasks etc., within an application to deal with a variety of exceptional situations. The system provides support for system level dependability by recording inter-task dependencies in transactional shared objects and by using transactions to implement the delivery of task outputs such that destination tasks receive their inputs despite finite number of intervening machine and network failures.
- **Adaptability:** The task model referred to earlier is expressive enough to represent temporal (dataflow and notification) dependencies between constituent tasks. Our application execution environment is reflective, as it maintains this structure and makes it available through transactional operations for performing changes to it (such as addition and removal of tasks as well as addition and removal of dependencies between tasks). Thus the system directly provides support for dynamic modification of workflows (ad hoc workflows). The use of transactions ensures that changes to schemas and instances are carried out atomically with respect to normal processing.

FlexFlow architecture is shown in Fig. 1.

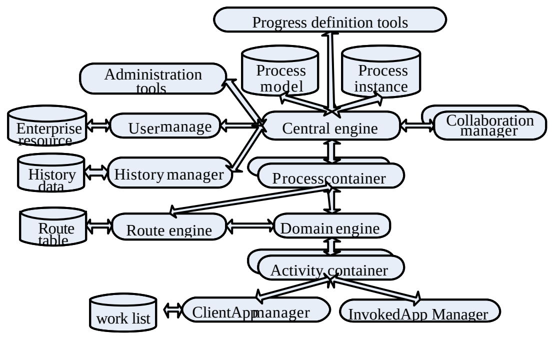

Fig. 1: FlexFlow distributed architecture

FUNCTIONAL MODULES

Process definition tools: Process definition tools have functions of analyze, model, compose, describe and document a business process. This utility might seem a composite one, but actually the above functions share something in common. These facilities are applied to the process definition during build time. The resulting definition is not operable without instantiation.

Central controller: A central controller supports the runtime environment of a WFMS which refers to a process scope and not to the atomic-task level. Communication among system modules and coordination are the most visible runtime control activities. Additionally it is responsible for managing the process definition repository and the process instance repository, such as import, export and query process definitions and process instance etc. Changes on processes may be applied either on the static definitions, either dynamically, on the executing instance. The central controller may get attributes' values from a specific definition. So it can also retrieve a list of process definitions that fulfill certain criteria and finally, they can retrieve the whole definition itself. It is able to interpret the process definition language. As more general contribution it also creates, manages and deletes each process container and supervises the running states of each process instance.

Process container: A process container is responsible for the execution of one particular case. For each work case and for each sub work case a new process instance is created. It controls executing process and state transition of a process instance, such as transmitting sliced model of the process instance according to routing information and collecting state information of activity instance container, etc. A process container considers the process as a finite state machine, thus it controls it through state transitions-actions. All actions carried out by an activity container are the result of the execution of a certain strategy decided when in a specific state. The decision making abilities of the container and his strategy selection, eventually provide him with the process control. It not only provides services of registration and close domain engines, but also obtains relevant information. Every time when adding a new domain engine or start-up an activity container, the registration service will be called. Accordingly, every time when closing an activity container, the close service will be called. It requests its first domain engine to create the first activity.

Domain engine: A domain engine supports the runtime environment of a WFMS which refers to the atomic-task level. According to the functions of each department enterprise is divided to several domains. It responses for the process container or for the previous domain engine to creates, manages and deletes activity container and supervises the running states of each activity instance. Through cooperation of all domain engines can successfully complete a series of business processes.

Activity container: An activity container is responsible for the control and management of the activity implementation, such as triggering synchronization, start-up task, scheduling follow-up activities and interaction with the client application, etc. If the implementation of activity is successful, according to local routing information it will determine which domain engine next activity can implement on. Otherwise it will inform failure message to the process container, so that the process container can trigger a new routing selection for the remainder of process instance.

Route engine: The rout engine and route tables are located on the same host. The route tables classified according to domains, store configuration information and registration information of each domain engine. The rout engine is mainly responsible for querying in the route tables and in accordance with load balancing algorithm allocating rout for each activity in relevant domain. In addition it is responsible for maintaining the update of routing tables. So the domain engine on which each activity of the process of different types runs is variable. Once exception occurs, it will be submitted to rout engine to trigger a new routing selection for the remainder of process instance.

ClientApp manager: A clientapp manager separates users from the implementation details of the manager completes human-machine interaction activities. This category embraces the interaction between client applications and the core of WFMS. It can manage the artificial task record, maintain and renew task state. For artificial activities, users must login the system to query, finish his work and notify of completion or other work status conditions.
InvokedApp Manager

The activities invoked by invokedapp manager are the same with those of the previous interface, except that manager do not communicate with users but with applications. Applications act more autonomously. That is to say, the function of enterprise applications is packaged into CORBA service object with unified interface. Different enterprise applications may be packaged into different CORBA service object. In processes the special activity is designed to common object, while existing program is packaged into package object. It proposed in enriches its functionality by exploiting its autonomous collaboration with other applications.

User manager: A user manager is the user interface for the human resource or the interface for some tool which can do tasks automatically (such as printers and scanners). The agent represents the resource in the system and negotiates the resource allocation on behalf of the resource and exchanges the necessary information for the execution of a task. It manages system users and its roles, such as user role query, users' logon, exit and recording users' missions, etc.

History manager: A history manager manage process instances, historical query and statistical analysis conveniently, the control data of completed processes is stored into the historical database. It also notifies the central controller if the data has changed.

Administration tools: Administration tools are the management interface of the system. It includes overall and local monitoring and management, such as start-up service, monitoring process instances and activity instances. Administration tools gather the data in the system that is necessary to analyse workflow, such as execution times and resource utilization. It also gathers this data associated with a particular case and (after the case is finished) sends this data to the history manager for persistent storage. It provides the feedback mechanism required for process re-engineering by sensing the anomalies.

Collaboration manager: Collaboration managers are other enactment services component (Workflow interoperability interface). A fundamental objective of the WF standards and of the WfMC itself is to allow workflow systems produced by different vendors to seamlessly interoperate. There are different levels of interoperation and plenty of connection architectures (Li *et al*., 2012a; b; c).

EXECUTING PROCESS

To illustrate the management process of the system, we describe the workflow executing process by UML sequence diagram as following.

The administrator, user or application starts executing a work case by putting a job token in the place called create case (step 1). According to real parameters, the Central Controller “CC” retrieves a process instance that fulfill certain criteria and creates the Process Container “PC1” (step 2). Then “PC1” applicants for route selection to the Route Engine “RE” (step 3). According to a certain load balancing algorithm, “RE” retrieves a relevant Domain Engine “A” in route tables and queries “A” whether it is balanceable (step 4). If true, “A” responses to “RE” (step 5) and registers to “PC1” (step 6). “PC1” creates its instantiation, such as initializing relevant data, mapping formal parameters-real parameters and etc (step 7). Then “PC1” reports process instance information (step 8) and status “CaseStarting” (step 9) to “CC”. “PC1” triggers the first task on “A”. So “A” creates Activity Container “AC1” which completes initializing the first task (step 10-13). Thus far the process instance turns into “CaseRunning” status. The process of executing a work case is shown in Fig. 3 and this process is repeated for every business process.

As shown in Fig. 4, after task has been done the Activity Container “AC1’ on Domain Engine “A” begins to handle subsequent transition (step 2) and through calculating transitions triggers next task on Domain Engine “B” (step 3). “B” responses for the request to create the Activity Container “AC2’ (step 4) and triggers precursor transition synchronously (step 6). After handling subsequent transition “AC1” reports its “TaskFinished” status to “PC1” and withdraws itself. So far a whole task has completed. Then “AC2’ begins to prepare the workflow data (step 9) and after necessary data are ready it begins to do its task. This process is repeated for every task.

The system adopts following data distribution strategy to obtain better balance of efficiency, reliability and the system complexity. The workflow relevant data is transferred dynamic among knowledge library in various Activity Containers according to the needs of activity instances. The Process Container is responsible for recording current location of process instance data. Before Activity Container starts a task, data needed should be migrated to local knowledge library. Activity Container must migrate and lock “write” variables to avoid conflict of “write” access. Otherwise Activity Container does not migrate “read” variables but keeps copies. Figure 5 shows this mechanism of preparing data.

As shown in Fig. 6 we suppose conditions of the third task are two precursor transitions of and-join relation. After Activity Container “AC1” completed its task, it requests Domain Engin “C” to trigger the third task (step 3). “C” finds its activity instance is not exiting, so it firstly creates Activity Container “AC3” (step 4) and triggers a transition (step 6). As the same, after Activity

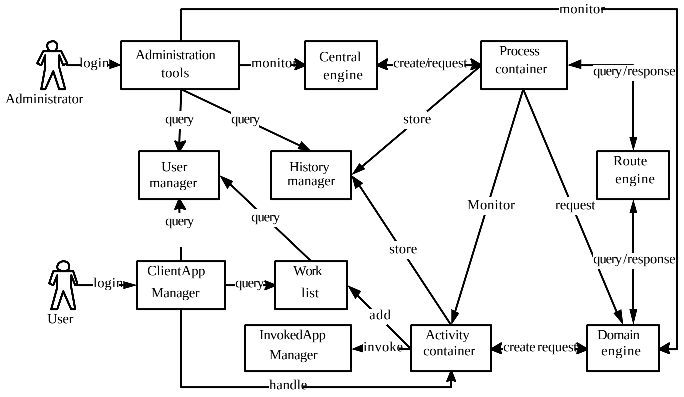

Fig. 2: Functions of FlexFlow modules

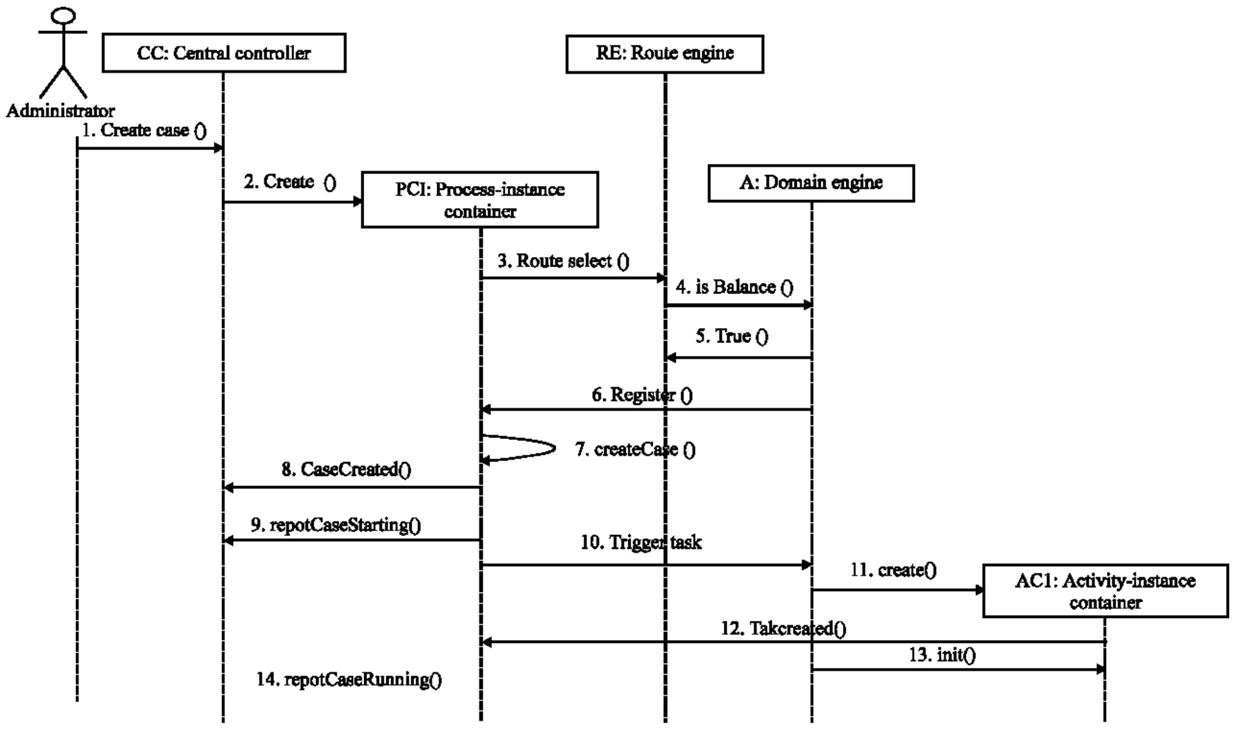

Fig. 3: The sequence diagram for creating a new work case

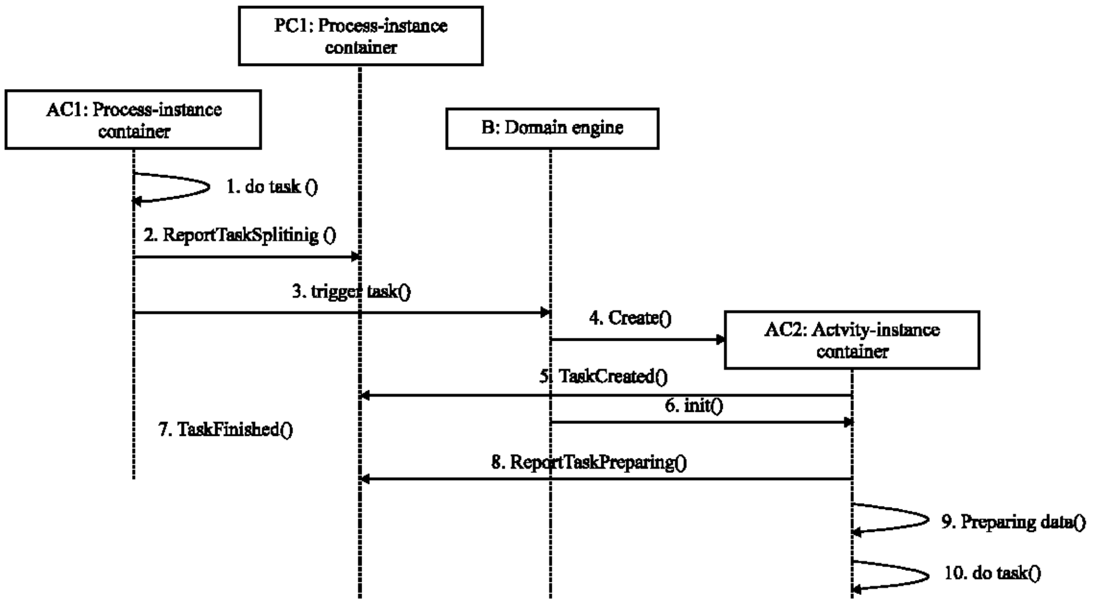

Fig. 4: The sequence diagram for executing tasks sequentially

Container "AC2" completed its task, it requests Domain Engin "C" to trigger the third task (step 9). Now "C" finds its activity instance "AC3"is exiting, so it directly triggers a transition (step 10). So far precursor transition conditions of "AC3" has been satisfied, it begins to do its task (step 13-14). Otherwise "AC1" and "AC2" exit.

A manual task is shown in Fig. 7. "AC1" generates work item, inserts it to ClientApp Manager "CAM" (step 1) and then turns into ready status (step 2) waiting

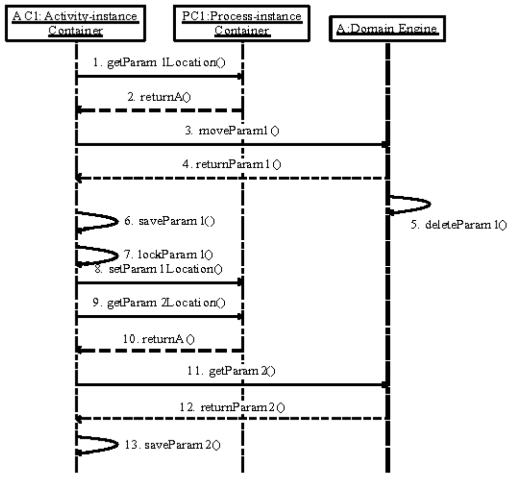

Fig. 5: The sequence diagram for preparing data

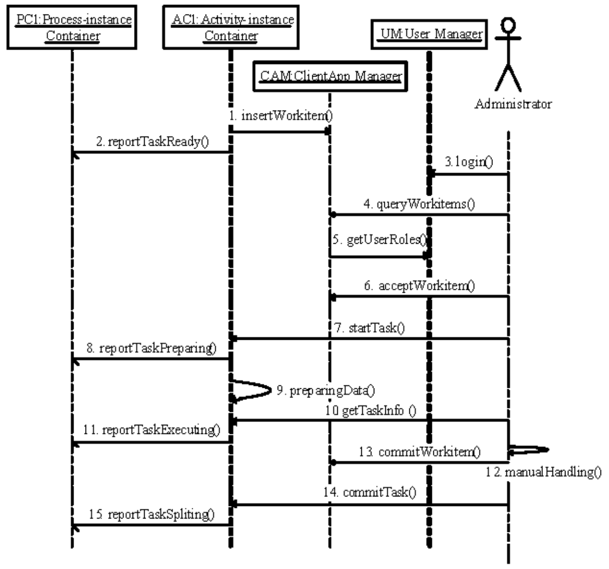

Fig. 7: The sequence diagram for manual tasks

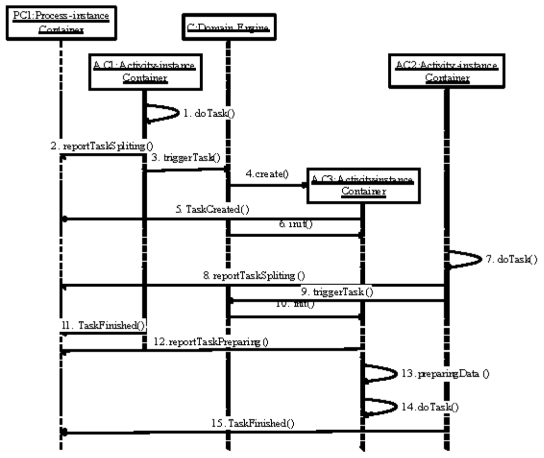

Fig. 6: The sequence diagram for executing and-join tasks

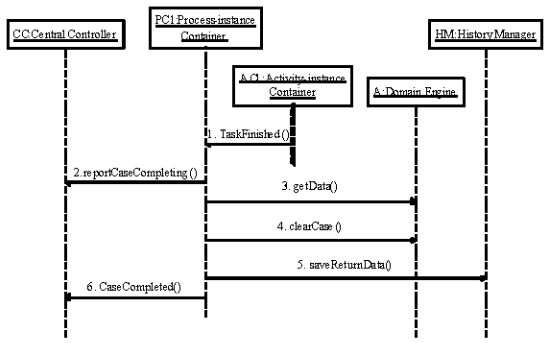

Fig. 8: The sequence diagram for treatments after finishing

for users. When some one logins system through User Manager "UM" (step 3) and queries work item from "CAM" (step 4), "CAM" queries user's role and authority from "UM" (step 5). If legal, user confirms to accept the task, inform "CAM" to lock it (step 6) and start "AC1" (step 7). "AC1" prepares necessary data (step9) and returns task information to user (step 10). After user handles the task (step 12), it is committed to "CAM" (step 13) and "AC1" (step 14).

After the last Activity Container "AC1" has finished its task (step 1) the Process Container "PC1" reports its completing status to the Central Controller "CC" (step 2) and save the process relevant data to the History Manager "HM" (step 3-5). Finally while it withdraws itself, the whole work case is finished as shown in Fig. 8.

CONCLUSIONS

In the study we have constructed the architecture of a prototype CORBA based workflow management system in a distributed environment which provides more interoperability and flexibility than existing WFMS. We implemented the described architecture and this study showed solutions for the problems arising while implementing this system. Our existing framework has been endowed with monitoring and feedback mechanisms, so that the various processes and cases can be studied, analyzed and the required feedback can be given to the workflow manager. Due to a distributed system, WFMS modular design not only reduces the complexity of the system, but also is easy to expand, as well as has good robustness and reliability. It could support workflow management at the desired level.

As part of our future work will be the integration of sophisticated resource management. One step would be to store all tasks that have been performed by a particular resource to allow more efficient resource scheduling [7].

ACKNOWLEDGMENT

This study was supported in part by NSFC under Grant No. 61263023 and No. 51174103. It also subsidized by key projects of Yunnan education department (2012Z107, 2013Z129).

REFERENCES

Li, Y., H.R. Wang and Z.B. Zhang, 2012. Multi-agent based workflow management systems design. Adv. Sci. Lett., 6: 727-731.

Li, Y., H.R. Wang, L. Zhang and X. Tong, 2012. Flexible distributed workflow management systems design based on CORBA. Applied Mech. Mater., 157-158: 839-842.

Li, Y., X. Tong, H.R. Wang, J.Y. Wang and Z.B. Zhang, 2012. Flexible distributed workflow system research. Adv. Mater. Res., 569: 688-692.