WW-FLOW: *Web-Based Workflow Management with Runtime Encapsulation*

YEONGHO KIM, SUK-HO KANG, AND DONGSOO KIM
Seoul National University
JOONSOO BAE
LG-EDS Systems
KYUNG-JOON JU
Computer & Software Technology Laboratory, ETRI

Runtime encapsulation of nested process models supports flexible workflow management in a distributed heterogeneous environment. Through its modular design, the WW-flow system provides a hierarchical control scheme over complex processes and subprocesses. Implemented in Java, the workflow engine and client interfaces are portable and platform-independent.

We use the term *workflow* to describe a business process that is executed and managed automatically by a computer system. The definition usually includes all the tasks, tools, procedures, and organizations involved, as well. A workflow management system (WfMS) is an application that uses a computer representation of the workflow logic to define, manage, and execute the process.[1]

Consider the workflow process in a collaborative engineering strategy for new product development. It may involve cooperation among design, manufacturing, assembly, testing, quality control, and purchasing departments, and may include both suppliers and customers.[2] While some of these functions are performed internally, others are carried out by external organizations that might, for example, design specialized components or develop numeric controller machine codes for computer-aided manufacturing work. Not only is such a workflow process complex, but it is also subject to change due to interim results—prototype test and engineering change request processes, for instance. Moreover, when subprocesses for developing component parts proceed in parallel, new requirements can be introduced along the way.

Interoperable WfMSs would enable these organizations to cooperate more effectively by readily sharing information and keeping pace with each other throughout the development process. To this end, we identify two main requirements for a WfMS: modular design/execution and Internet compliance. To support modularity, we have adopted an IDEF0-like nested modeling technique to model the workflow processes.[3] This method, which we call nested process modeling, allows the process manager to break a complex business process into several subprocesses and provide a structure for hierarchically arranging them. If the WfMS is also fully compliant to Internet protocols, users can have ready access to the system from any location. Moreover, maintenance is straightforward in a Web-based system, and client programs can be easily updated because they need not be installed for each user. Our Web-based workflow management system, WW-flow, was designed to manage complex business processes in a dis-

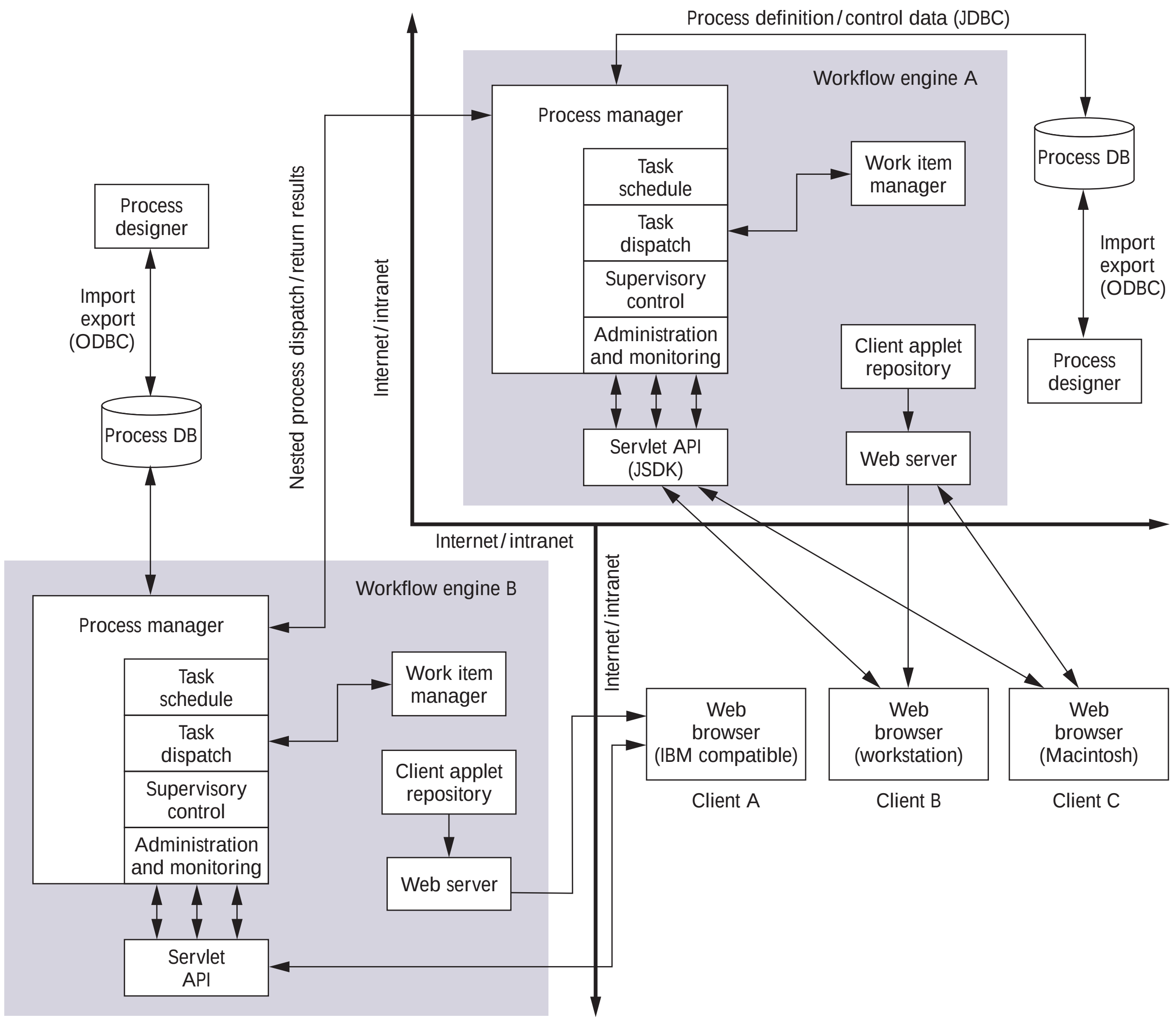


Figure 1. Overall architecture of WW-flow. Workflow engines can interoperate with others using nested process models and runtime encapsulation. All client programs are downloaded from the workflow engine via Web interfaces, so any client user can participate in the workflow system.

tributed and heterogeneous environment, such as the World Wide Web.

SYSTEM DESIGN

Figure 1 shows WW-flow's overall architecture. The system has five main components: the process designer, process database, workflow engine, work item manager, and workflow client.

The *process designer* is a stand-alone client application that lets a user model business processes. It translates process models into a format the workflow engine can interpret and exports them to the database using Open Database Connectivity (ODBC) to communicate. The process designer also supports nesting, which allows a model in the process database to be imported, modified, and then embedded into other processes as a subprocess.

The *workflow engine* creates or updates the process's control schedule according to the process execution status. It dispatches each task to the appropriate performers according to corresponding process definitions in the process database. One of WW-flow's distinguishing features is that multiple workflow engines can interact with each other using our runtime encapsulation mechanism, which means that a task performer can even be another workflow engine at a remote site.

The system in the figure is installed at two sites,

Research on Web-Based Workflow

Rapid technological advances in Web-based WfMSs have extended the workflow environment across the Internet. One of the first Web-based WfMSs, WebWork, was developed at the University of Georgia's LSDIS Lab and implemented using CGI and JavaScript.[1] Dartmouth College's DartFlow, on the other hand, uses transportable agents, as well as CGI, and Java technologies.[2] There are now several commercial WfMSs, including InConcert 2000, Staffware 2000, Metro, and Ultimus, that provide client interfaces for the Web environment. Some systems, such as i-Flow, implement a Java workflow engine as well as Web interfaces.

As for workflow standards, the international Workflow Management Coalition (WfMC) has proposed a reference model for a WfMS framework.[3] The model consists of five components (all of which are incorporated into WW-flow):

- process definition tools,
- workflow enactment service,
- workflow client applications,
- invoked applications, and
- administration and monitoring tools.

WfMC has recommended standards for interfacing these components with workflow engines. They also define four interoperability models for communication among heterogeneous workflow systems:

- connected discrete,
- hierarchical,
- connected indiscrete, and
- parallel synchronized.

Our nested process modeling approach adopts and generalizes the hierarchical model, which allows a parent-child relationship between processes.

The Process Interchange Format (PIF) working group is researching an interchangeable format for process specifications, enabling different WfMSs to transparently exchange process definitions.[4] The Internet Engineering Task Force (IETF) SWAP working group has also proposed using the Simple Workflow Access Protocol (SWAP).[5] This protocol is designed to integrate work providers and asynchronous services and provide for their interaction across the Internet. The integration and interactions consist of controlling and monitoring the work using HTTP and transferring structured information encoded in XML.

References

1. J. Miller et al., "The Future of Web-Based Workflows," *Proc. Int'l Workshop on Research Directions in Process Technology*, Nancy, France, 1997.
2. T. Cai, P.A. Gloor, and S. Nog, "DartFlow: A Workflow Management System on the Web using Transportable Agents," Tech. Report PCS-TR96-283, Dartmouth College, Hanover, N.H., 1996; available online at http://www.cs.dartmouth.edu/reports/authors/Cai,Ting.html.
3. D. Hollingsworth, "The Workflow Reference Model," Workflow Management Coalition Spec., WfMC-TC-1003, Jan. 1995.
4. J. Lee et al., "The PIF Process Interchange Format and Framework," PIF Working Group, May 1996.
5. G.A. Bolcer and G. Kaiser, "SWAP: Leveraging the Web to Manage Workflow," *IEEE Internet Computing*, vol. 3, no. 1, Jan./Feb. 1999, pp. 85-88.

for example, and system A can dispatch a task (which is actually a subprocess) to system B while executing a process. System B then executes the subprocess and returns the results to system A. All the workflow engine's functions are implemented in pure Java to ensure platform independence, which is vital to interoperability. The engine also provides supervisory control services for active process monitoring, statistical reporting on completed processes and tasks, and expediting, aborting, or suspending certain tasks.

Once the workflow engine releases the tasks, the *work item manager* takes control of them, checking them out to users upon request. In addition to this pull approach, the engine uses a simple push technique whenever a task is released to a user by automatically e-mailing them an alert. Since e-mail can be read using portable devices like PDAs and mobile phones, this approach particularly aids mobile and infrequent users.

The *workflow clients* provide interfaces for users classified according to their roles and responsibilities. Classifications include initiators, task performers, supervisory managers, and system administrators. All the user interfaces are developed using Java applets and can be downloaded via any Web browser, which provides location transparency. We use servlets for communication between the workflow engine and clients.[4] The client interface is interlinked with a Web-based document management system.

NESTED PROCESS MODELING

Nested process modeling is key to the runtime encapsulation method we employ in the WW-flow system. It allows the process designer to break a complex business process into several subprocesses and provides a structure for hierarchically arrang-

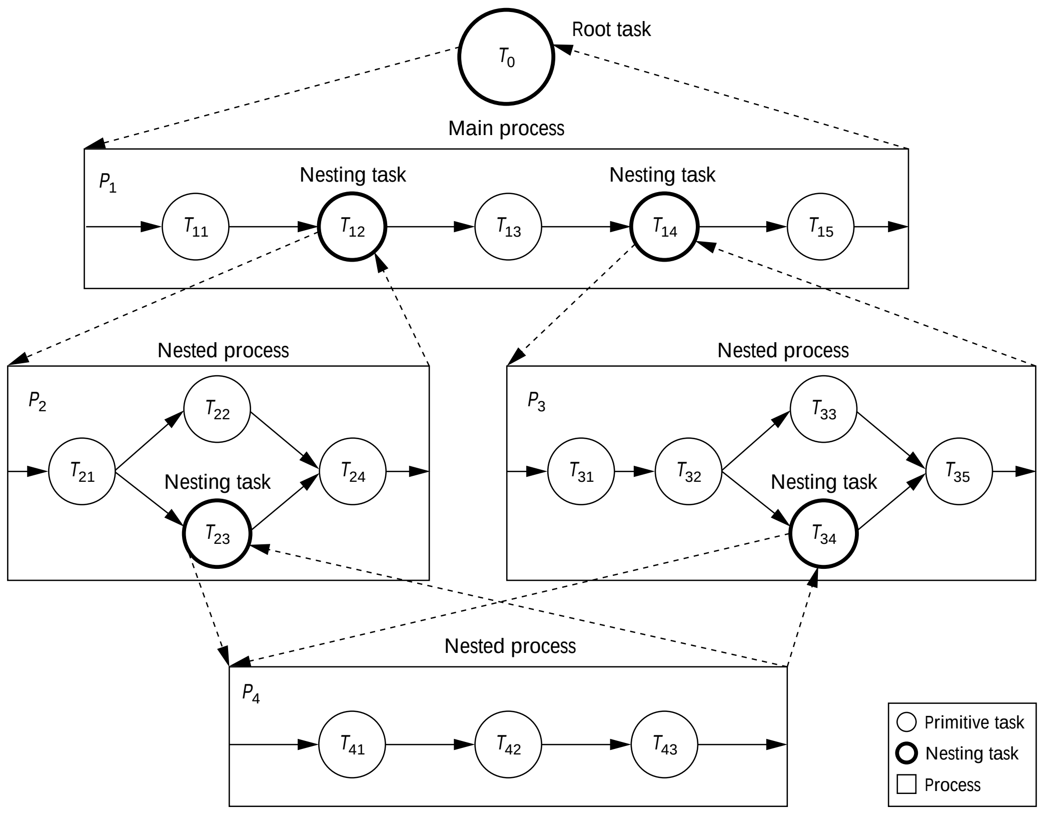


Figure 2. An example of a nested process structure.

ing them. The database schema supports the nested model and the task status types used for designing the control mechanism.

Nested Processes

A workflow process is conventionally defined by its constituent tasks and the precedence relations among them. A task is thus considered a component of the process. In a nested process, on the other hand, a task can further be mapped onto another process model with more detailed specifications. The task then becomes both a parent of one process and a child of another higher level process.

As shown in Figure 2, tasks in the nested process structure are classified into two types: primitive and nesting. A *primitive task* cannot be broken down into smaller elements. *Nesting tasks* are all deployed into subprocesses. That is, task details are modeled in the subprocess. In Figure 2, the *root task* T_0 is at the uppermost level, and its detail is modeled in process P_1. Within P_1, the nesting tasks T_{12} and T_{14} are again deployed to processes P_2 and P_3. This type of hierarchical relationship can appear at an unlimited number of levels. The figure also illustrates how a process definition, such as P_4, can be used several times.

Many traditional WfMSs rely on one-dimensional, flat process models, in which a process definition includes every detail of the process from beginning to end. Nested process modeling provides several advantages for use in a distributed environment.

- The simple top-down method enables the design and analysis of very complex processes.
- Its theoretical background exploits object-oriented approaches (for example, encapsulation, polymorphism, and inheritance) to modeling processes and developing workflow management systems.
- Frequently used process models can be provided in libraries, which increases the reusability of process models and thus reduces the efforts needed to design processes.
- Since each process is an autonomous unit that is manageable and controllable in a distributed environment, it forms the basis of encapsulation during runtime.
- The nested model can be easily extended to a model for interoperability between heterogeneous and distributed workflow systems.

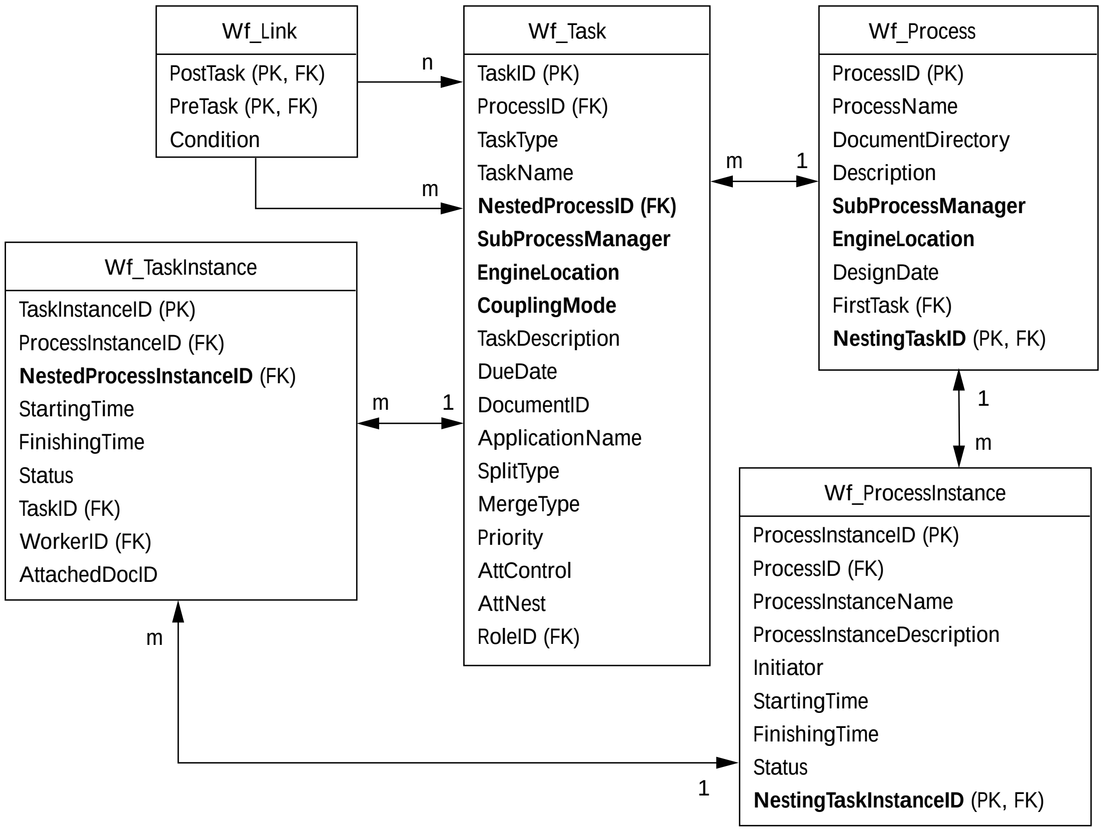


Figure 3. Database schema for nested processes. The schema shows the relationship between task, process, task instance, and process instance. The database tables are designed to maintain the nesting relations and attributes required for runtime encapsulation.

In our approach, a manager in charge of the whole process can design a model at the top level that includes several subprocesses, which are represented as abstracted tasks. At that level, they can be defined simply according to their required inputs and outputs. A participant with expertise in each area can then create the detailed definitions of the subprocesses. With our system they can also execute and control the nested processes.

Runtime Encapsulation

Runtime encapsulation requires that each subprocess have an interface to its nesting task. Therefore, the parent process model containing the nesting task defines the input and output of the subprocess. The model does not have to include the features specific to the subprocess because they can be defined at the site that executes it.

When a subprocess is called, the nesting task contacts the workflow engine that will execute and manage the subprocess, and the engine controlling the parent process forwards certain required information, including the input, due date, and output. If the engine does not have a process model for the subprocess, a new one must be created unless a model with the same format can be delivered from the parent engine. In either case, a subprocess model can be freely modified at the child workflow engine. Because encapsulation localizes the modification, it does not affect the parent process. Upon completion, the engine sends the results to the higher level engine.

Runtime encapsulation facilitates cooperation among different departments and organizations, and it improves WfMS scalability by enabling distributed workflow engines to manage subprocesses independently. Separating complex processes into sets of smaller units also improves stability within the WfMS: problems are isolated and thus cannot influence other processes. By localizing definition modifications, runtime encapsulation makes it easy to dynamically adapt process models.

Table 1. Return code and task status.

Return code	Task status	Description
n/a	NotReady	Initial state
n/a	Ready	A task is ready to begin, but not executed yet.
Assign	Doing	A task is currently being executed.
Finish	Done	A task is normally terminated.
Suspend/Resume	Suspended	A task is temporarily stopped.
Reject	Rejected	A task is rejected.
Abort	Aborted	The whole process is cancelled.
Fail	Failed	A task is failed, which can start an alternative task.

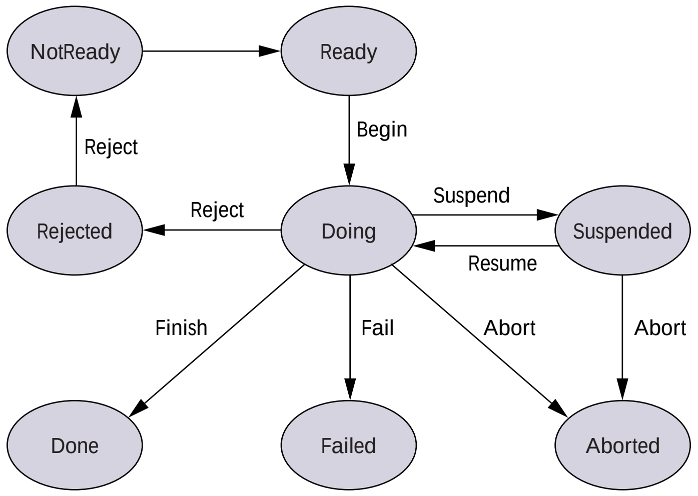


Figure 4. State transition diagram.

Database Schema for Nested Process Models

Figure 3 shows a database schema, including table names and major attributes, for storing and maintaining nested process models. The Wf_Process and Wf_Task tables store static information concerning the structural definitions of processes, whereas the Wf_ProcessInstance and Wf_TaskInstance tables contain operational information that is generated while a process instance is being executed.

For a nesting task within the Wf_Task table, the NestedProcessID attribute field points to the corresponding nested process. For a primitive task the attribute is assigned a Null value. The NestingTaskID attribute in Wf_Process indicates the task from which a process is nested. The cardinality of 1 : *m* between Wf_Process and Wf_Task indicates that a process can be nested by multiple nesting tasks, which supports multiple usage of a subprocess. A similar relationship exists between Wf_ProcessInstance and Wf_TaskInstance.

The Wf_Process knows only the first task of a process, which is specified in the FirstTask attribute, but all the succeeding tasks can be identified by referring to the precedence relations in Wf_Link. The CouplingMode attribute in the Wf_Task indicates the coupling mode between a nesting task and its nested process. Together with the SubProcessManager and EngineLocation, this attribute is required to support runtime encapsulation.

Control Mechanism

WW-flow's component modules interact by exchanging messages (return codes) and data. Our control mechanisms are similar to those reported in Kumar and Zhao[5] and Casati et al.[6] Table 1 lists the return codes we use for interactions between the workflow engine and normal clients. After completing its assigned work, a normal client replies to the workflow engine with one of the return codes.

In the nested process model, every subprocess is associated with a nesting task, and thus both tasks and processes have the same status codes. All tasks are initialized as *NotReady*. When all preceding tasks are completed, the status changes to *Ready*. The workflow engine then assigns a Ready task to a performer, and the status becomes *Doing*. Completion of the task changes the status depending on the code returned. The state transition diagram in Figure 4 shows how a task's final status can be *Done*, *Failed*, or *Aborted*.

WORKFLOW ENGINE

The workflow engine interprets process definitions and creates and executes process instances. It also performs housekeeping of data required for supervision and system administration.

Process Control Procedure

A process flow can be either serial or parallel. For the parallel process flows, we consider four types of split—concurrent, alternative, exclusive, or conditional—each of which can be associated with the same type of merge.[1] Figure 5 is a flowchart of the process control procedure, which we have grouped into five sectors. Each handles certain important functions.

1. When the engine receives a task return code

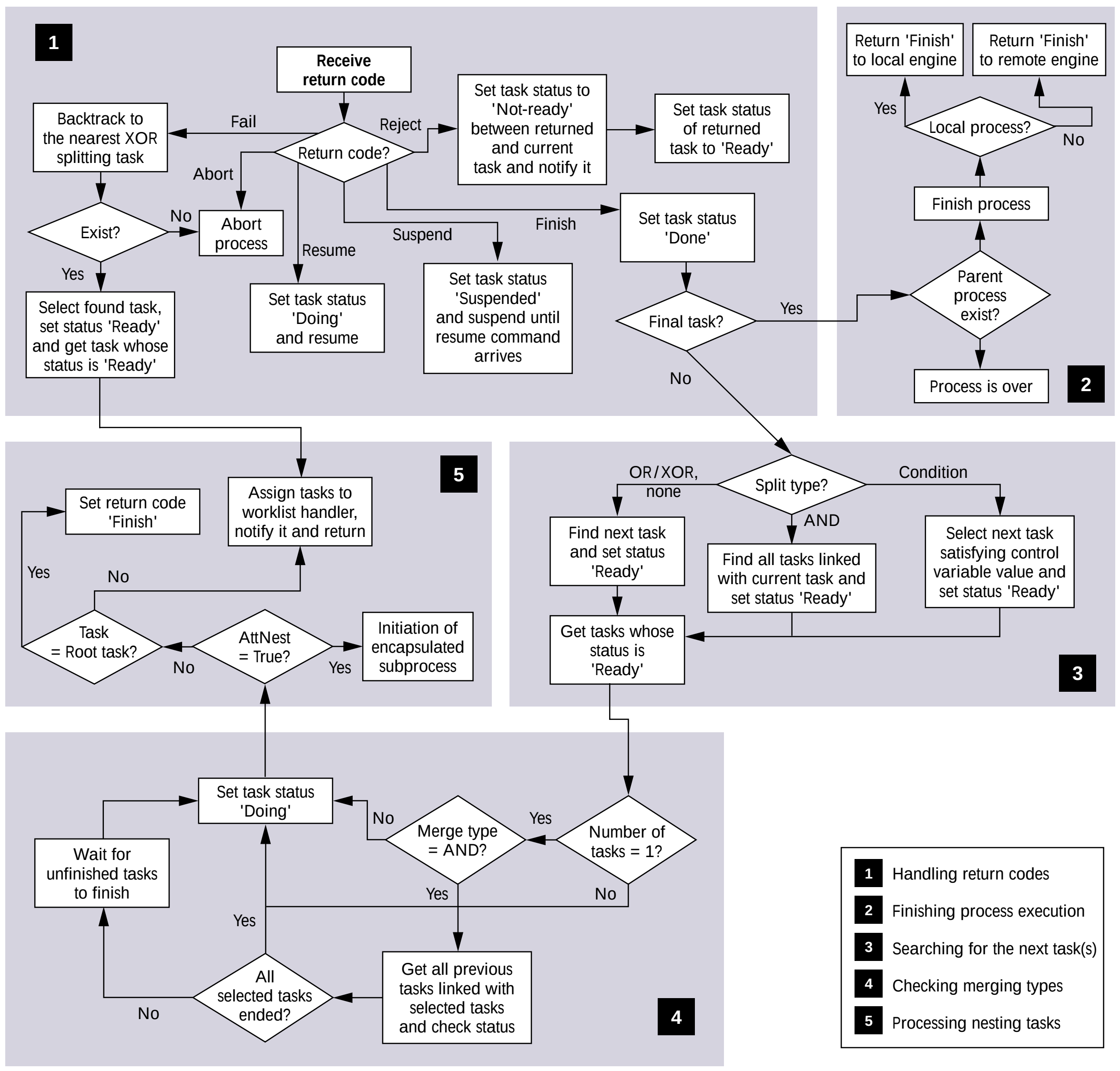

1
Receive return code
Return code?
Fail
Reject
Abort
Resume
Suspend
Finish
Backtrack to the nearest XOR splitting task
Set task status to 'Not-ready' between returned and current task and notify it
Set task status of returned task to 'Ready'
Exist?
No
Yes
Abort process
Select found task, set status 'Ready' and get task whose status is 'Ready'
Set task status 'Doing' and resume
Set task status 'Suspended' and suspend until resume command arrives
Set task status 'Done'
Final task?
Yes
No
2
Return 'Finish' to local engine
Return 'Finish' to remote engine
Local process?
Yes
No
Finish process
Parent process exist?
Process is over
5
Set return code 'Finish'
Assign tasks to worklist handler, notify it and return
Yes
No
Task = Root task?
AttNest = True?
No
Yes
Initiation of encapsulated subprocess
3
Split type?
OR/XOR, none
AND
Condition
Find next task and set status 'Ready'
Find all tasks linked with current task and set status 'Ready'
Select next task satisfying control variable value and set status 'Ready'
Get tasks whose status is 'Ready'
4
Set task status 'Doing'
Merge type = AND?
Number of tasks = 1?
No
Yes
Wait for unfinished tasks to finish
All selected tasks ended?
Get all previous tasks linked with selected tasks and check status
1 Handling return codes
2 Finishing process execution
3 Searching for the next task(s)
4 Checking merging types
5 Processing nesting tasks

Figure 5. Process control flowchart.

from a client, it calls an appropriate function determined by the return code.

2. When the last task of a process returns the Finish code, the process terminates and the nesting task is complete. As every process is nested to a task, the whole process is complete if the nesting task is the root task.
3. The engine checks the split type, identifies the next task to be executed, and changes the status to Ready.
4. The engine checks the merge type of the next task to make sure it is executable. If it is a concurrent merge, for example, the next task's execution must be postponed until all preceding tasks are complete.
5. Runtime encapsulation is implemented in the nesting task processing sector.

The process control procedure manages the interaction between the engine and clients. The procedure runs whenever a task return code is received from client users, and assigns tasks to relevant clients. Central to the interaction is the control of nesting tasks through runtime encapsulation.

Implementation of Runtime Encapsulation

The pseudocode for processing nesting tasks is provided in Figure 6. The procedure first checks with the appropriate workflow engine, and if the engine

```
void Handling_Nested_Process(TaskInstanceID id)
{
    TaskInstanceID cid;
    ProcessInstanceID pid;
    UserID uid;
    ProcessDefinition pd;
    NestingTaskInformation ntinput;
    CouplingMode cmode;
    If(Check_If_Nesting(id)) {
        If(Check_Responsible_Engine(id)) {
            If(Check_If_Designed(id)) {
                pid=Create_Process_Instance(id);
                cid=Get_First_Task(pid);
                Set_Task_Status(cid, READY);
                Handling_Nested_Process(cid);
            }
            Else
                Create_New_Process_Model(id);
        }
        Else
            Notify_to_Remote_Engine (id, pd, ntinput, cmode);
    }
    Else {
        uid=Get_Available_User(id);
        Assign_Task(id);
        Set_Task_Status(id, DOING);
        Notify(uid);
    }
}
```

Figure 6. Pseudocode for processing nesting tasks.

running the procedure is the same one that will execute the subprocess, the procedure checks the subprocess model's availability. If it is unavailable, a new model is designed and executed. Once the model is in place, the procedure creates an instance of the subprocess, gets its first task, and sets the task's status to Ready. Since this task can also be a nesting task, the procedure calls the Handling_Nested_Process() function recursively.

If the nested process is to be executed at a remote engine, the procedure notifies the remote engine and passes control to it with TaskInstanceID. While executing this subprocess, the WW-flow system applies runtime encapsulation. To implement the mechanism, we have considered three coupling modes.

- *Tight coupling*. The engine executing the nesting task has the authority to control and monitor the nested subprocess in the remote engine. The control function allows expediting, aborting, suspending, or resuming every task.
- *Intermediate coupling*. The engine can monitor the status of the subprocess in the remote engine, but cannot control it.
- *Loose coupling*. The engine has no controlling or monitoring authority. The subprocess is executed independently, and the execution detail is completely hidden from the engine, which must wait for the final results.

A designer specifies a relevant coupling mode as an attribute of every nesting task. When notifying a remote engine of a nesting task, the coupling mode is transmitted in the cmode argument. The other three arguments transmitted are id, pd, and ntinput, which respectively contain the nesting task identification, subprocess model, and other required information, such as input documents, due date, and description. Transmitting the subprocess model is optional because the remote engine can use the model provided by the host engine or its own model. It is also possible to create a new model or modify an existing one before launching the subprocess.

For a primitive task, the engine identifies a performer, sets the task status to Doing, and dispatches the task to the responsible user. When the performer is defined as a group of people, the engine chooses one of them by calling a workload balancing procedure. The engine then e-mails the performer a task-notification message. This is implemented using the Java mail API.

WW-FLOW CLIENTS

There are two types of workflow users: *build-time* and *runtime*. A build-time user creates process models using Process Designer, and a runtime user, or task performer, carries out actual tasks. For the runtime user, we provide Web-based client modules.

Process Designer

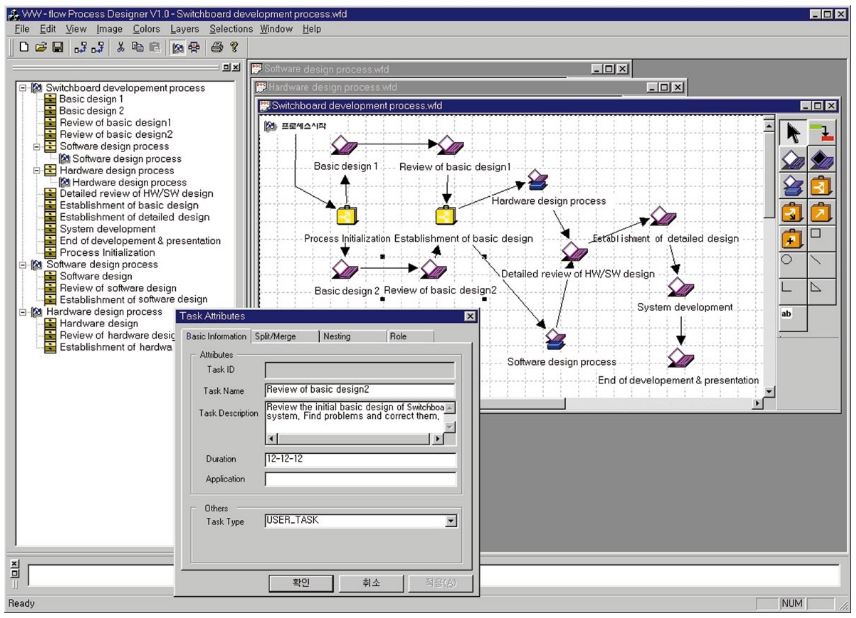
The *process designer* provides build-time functionality for the system. It is implemented in Visual C++, and the GUI components were developed using the Objective Diagram class library from Stingray Inc. The GUI lets a process designer create and edit nested process models.

Figure 7 is an example of a hierarchically nested model. The overall structure of a process can be viewed in a tree structure through the designer's *process hierarchy window*, which is similar to a folder structure. Selecting a nesting task in this window activates the diagram window containing the corresponding nested task. The process designer also includes functions for error checking in process models, exporting completed models to databases, and importing subprocesses for nesting.

Web-Based Clients

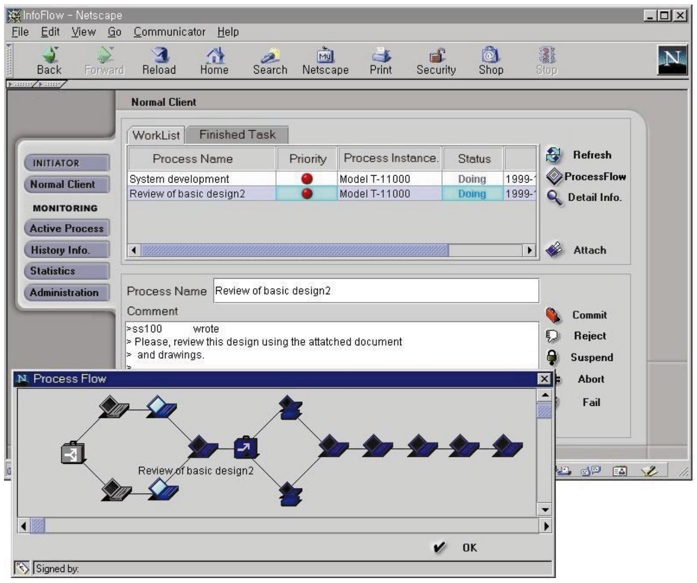
WW-flow's client modules are Java applets developed using JBuilder 2.0 and JDK 1.2. The system has four client interfaces:

- The initiator module launches instances of process models.
- The system administrator module provides general functions for managing users, roles, authority, and so on.
- The *normal* client interface allows performers to receive task information from a workflow engine and respond to it with their results.
- The *supervisory control and monitor* module is an authorized user that can monitor the state of processes being executed and control the expediting, suspending, resuming, or aborting of tasks. It also provides statistics on finished processes, including overall performance, task history, workloads for each task performer, and so forth.


WW - flow Process Designer V1.0 - Switchboard development process.wfd
File Edit View Image Colors Layers Selections Window Help
Switchboard developement process
Basic design 1
Basic design 2
Review of basic design1
Review of basic design2
Software design process
Hardware design process
Detailed review of HW/SW design
Establishment of basic design
Establishment of detailed design
System development
End of developement & presentation
Process Initialization
Software design
Review of software design
Establishment of software design
Hardware design
Review of hardware design
Establishment of hardware design
Software design process.wfd
Hardware design process.wfd
Switchboard development process.wfd
프로세스시작
Basic design 1
Review of basic design1
Hardware design process
Process Initialization
Establishment of basic design
Establishment of detailed design
Detailed review of HW/SW design
Basic design 2
Review of basic design2
System development
Software design process
End of developement & presentation
Task Attributes
Basic Information
Split/Merge
Nesting
Role
Attributes
Task ID
Task Name
Task Description
Duration
Application
Others
Task Type
확인
취소
적용(A)
Ready
NUM

Task Name: Review of basic design2
Task Description: Review the initial basic design of Switchboa system. Find problems and correct them.
Duration: 12-12-12
Task Type: USER_TASK

Figure 7. Process Designer. The user can create and edit a diagram model of a nested process by a simple mouse click. Task attributes are input through the pop-up window.



Process Name	Priority	Process Instance.	Status	
System development	●	Model T-11000	Doing	1999-
Review of basic design2	●	Model T-11000	Doing	1999-

Process Name: Review of basic design2
Comment:
```
>ss100      wrote
> Please, review this design using the attatched document
>  and drawings.
```

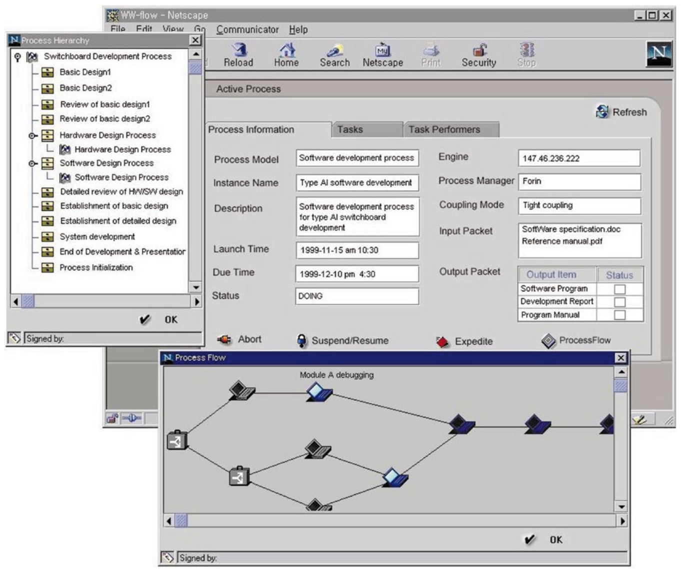
Figure 8. Normal client interface.

Figure 8 shows the interface for a normal client module managing a task list for each user (listed in the WorkList tab). The navigation links to the right of the tab let the user view the up-to-date task list, overall process containing the selected task, and detailed information. For example, the ProcessFlow button opens a window that highlights tasks with a status of Doing. With WW-flow's document management system, the user can also relate files to the task by clicking the Attach button.

High-level managers are generally more interested in abstracted information about entire processes, whereas low-level managers often require more detailed information concerning specific subprocesses. The runtime encapsulation proposed in this article supports this type of modularized monitoring service through a zoom in/out function.

Figure 9 shows a monitoring applet with a Process Hierarchy tree open in the window at the left. In our example, the user is monitoring a subprocess executed by a remote engine, and the information is displayed on the main monitoring window, including general process descriptions, engine location, coupling mode, and input and output packets. The subprocess model can also be presented in another window as shown at the bottom of the figure.

The coupling mode determines the abstraction level for monitoring and control. With tight coupling, the remote engine provides the output packet status as well as the subprocess model, launch time, due time, and a list of process managers. The supervisory control buttons at the bottom of the main window let the user intervene in the execution of the subprocess. In other modes, the buttons are disabled.



Switchboard Development Process
- Basic Design1
- Basic Design2
- Review of basic design1
- Review of basic design2
- Hardware Design Process
  - Hardware Design Process
- Software Design Process
  - Software Design Process
- Detailed review of HW/SW design
- Establishment of basic design
- Establishment of detailed design
- System development
- End of Development & Presentatior
- Process Initialization

Process Model: Software development process
Instance Name: Type AI software development
Description: Software development process for type AI switchboard development
Launch Time: 1999-11-15 am 10:30
Due Time: 1999-12-10 pm 4:30
Status: DOING
Engine: 147.46.236.222
Process Manager: Forin
Coupling Mode: Tight coupling
Input Packet: SoftWare specification.doc Reference manual.pdf
Output Packet:

Output Item	Status
Software Program	☐
Development Report	☐
Program Manual	☐

Figure. 9 Remote subprocess monitoring.

LESSONS LEARNED

A standardized process definition format is vital to complete interoperability among heterogeneous computing environments, and we believe the concept of nested modeling should be included in that standard. Our research has only partially achieved our goals because WW-flow can only interact with servers using the engine we have developed. True interoperability requires that all workflow engines involved in a process support a common set of APIs. One alternative to our method would be to use the SWAP-compatible process definitions in XML.

Another important issue for further research is to incorporate WfMSs into information systems such as enterprise resource planning, supply chain management, and order tracking systems so that they are interoperable. Agent technologies have great potential for aiding this type of integration.[7] ■

ACKNOWLEDGMENTS

This research was supported by the Korean Science and Engineering Foundation (KOSEF) under Grant No. 97-02-00-09-01-3. The authors would like to thank the anonymous referees and the guest editors, especially Charles Petrie, for their valuable comments in improving this article. Special thanks also go to Hyerim Bae, Wonchang Huh, Young-Myoung Ko, and Sangjin Lee for their countless overnight programming sessions.

REFERENCES

1. D. Hollingsworth, "The Workflow Reference Model," Workflow Management Coalition Spec., WfMC-TC-1003, Jan. 1995.
2. C.Y. Kim et al., "Distributed Concurrent Engineering: Internet-Based Interactive 3-D Dynamic Browsing and Markup of STEP Data," *Concurrent Engineering: Research and Applications*, vol. 6, no. 1, Mar. 1998, pp. 53-70.
3. P.G. Ranky, *Manufacturing Database Management and Knowledge-Based Expert Systems*, CIMware Ltd., Guildford, Surrey, England, 1990.
4. J.D. Davidson, "Java Servlet API Specification, v2.2," Sun Microsystems, Nov. 1998; available online at: http://java.sun.com/products/servlet/.
5. A. Kumar and J.L. Zhao, "Dynamic Routing and Operational Controls in Workflow Management Systems," *Management Science*, vol. 45, no. 2, Feb. 1999, pp. 253-272.
6. F. Casati et al., "Deriving Active Rules for Workflow Enactment," *Proc. 7th Int'l Conf. Database and Expert Systems Applications*, *Lecture Notes in Computer Science*, Springer-Verlag, 1996, pp. 94-110.
7. C. Petrie, S. Goldmann, and A. Raquet, "Agent-Based Process Management," *Proc. Int'l Workshop on Intelligent Agents in CSCW*, Deutsche Telekom, Dortmund, 1998, pp. 1-17.

Yeongho Kim is an assistant professor in the Industrial Engineering Department at Seoul National University (SNU), Korea. His research interests include workflow management systems, product data management, and Internet applications in engineering. Kim received a PhD from North Carolina State University at Raleigh, and BS and MS degrees from SNU.

Suk-Ho Kang is a professor in the Industrial Engineering Department at SNU. He has a BS in physics from SNU, an MS from the University of Washington, and a PhD from Texas A&M University, both in industrial engineering. His research interests are in intelligent manufacturing systems and CALS/EC.

Dongsoo Kim is a PhD candidate at SNU. He received BS and MS degrees in industrial engineering from SNU in 1994 and 1996, respectively. His research interests include workflow systems, agent technology, database systems, and Web applications development.

Joonsoo Bae received PhD, MS, and BS degrees from the Industrial Engineering Department at SNU. He currently works for LG-EDS Systems Inc. in South Korea. His research interests include system integration, active database systems, workflow systems, and e-business.

Kyung-Joon Ju received BS and MS degrees in industrial engineering from Korea University. He currently works for Computer & Software Technology Laboratory, ETRI, in South Korea. His research interests include workflow systems, supply chain management, and e-business.

Readers can contact the author at yeongho@snu.ac.kr.