PAPER *Special Section on Software Agent and Its Applications*

Interorganizational Workflow Execution Based on Process Agents and ECA Rules

Donghui LIN[†a)], **Huanye SHENG**[††], ***Nonmembers****, and* **Toru ISHIDA**[†], ***Member***

SUMMARY Flexibility, adaptation and distribution have been regarded as major challenges of modern interorganizational workflow. To address these issues, this paper proposes an interorganizational workflow execution framework based on process agents and ECA rules. In our framework, an interorganizational workflow is modeled as a multiagent system with a process agent for each organization. The whole execution is divided into two parts: the *intra-execution*, which means execution within a same organization, and the *inter-execution*, which represents interaction between organizations. For *intra-execution*, we use the method of transforming the graph-based local workflow into block-based workflow to design general ECA rules. ECA rules are used to control internal state transitions and process agents are used to control external state transitions of tasks in the local workflows. *Inter-execution* is realized by process agent interaction protocols. The proposed approach can provide flexible execution of interorganizational workflow with distributed organizational autonomy and adaptation. A case study of offshore software development is illustrated for the proposed approach.

key words: *interorganizational workflow, ECA rules, workflow execution, process agent*

1. Introduction

Workflow management [13] has been widely adopted as an important technology to manage business processes. In recent years, with the global expansion of distributed computing environments, computer mediated collaboration has been increasing among organizations. In such cases, interorganizational workflow is expected to be created to cross organizational boundaries inside an enterprise or between enterprises [1].

There are several major problems in managing interorganizational workflow. For example, flexibility and adaptation are challenging issues [2], especially when the execution is across organizations. Existing workflow management systems (WfMS) do not effectively address issues related to environments distributed across organizations. A central and monolithic workflow engine as suggested by Workflow Management Coalition (WfMC) reference model [13] is not sufficient to support the autonomy of enterprises. Multiagent system provides distributed platform, which has been regarded as a promising approach to solving many problems in workflow management [12]. However, it is rarely discussed in previous research that how to use agent technology to achieve the goal of supporting distribution and flexibility with adaptation and autonomy for interorganizational workflow execution.

To address these issues, this paper proposes a framework for interorganizational workflow execution based on Event-Condition-Action (ECA) rules and process agents. The whole interorganizational workflow is modeled as multiagent system with a process agent for each organization. Therefore, process agents of organizations preserve autonomy of organizations with flexibility. We design general ECA rules to make workflow execution automatic with adaptation for different organizations. The framework will provide effective execution mechanisms for interorganizational workflow.

In the proposed framework, we divide the interorganizational workflow execution process into two parts: the *intra-execution*, which means execution within a same organization, and the *inter-execution*, which represents interaction between organizations. The *intra-execution* is distributed among organizations. Within each organization, there is an engine where a set of general ECA rules is defined for executing the internal workflow process. We use a method of transforming the graph-based workflow model into block-based workflow model [4], [10] to derive general rules from blocks. The execution of local workflow processes is controlled and monitored by process agents and rules. Rules are designed to control internal state transitions and process agents are used to control the external state transitions of tasks. Process agent of each organization interacts with that of other organizations to fulfill the *inter-execution*. Protocols are used to handle interactions, which involve all the organizations that have interaction with each other to address specific purposes.

This paper is organized as follows: Sect. 2 introduces related work. The overview of our approach is proposed in Sect. 3. Section 4 describes derivation of ECA rules. In Sect. 5, we explain process agent for controlling local workflow process and interacting among organizations. Section 6 is a case study and discussion of the proposed approach, followed by the conclusion in the last section.

2. Related Work

ECA rules have been used for workflow execution in previous research. Casati et al. design various rules for workflow management using patterns and propose a classification of

Manuscript received November 30, 2006.
Manuscript revised March 22, 2007.

[†]The authors are with the Department of Social Informatics, Kyoto University, Kyoto-shi, 606–8501 Japan.
[††]The author is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, HuaShan Road 1954, Shanghai, 200030, China.
a) E-mail: lindh@ai.soc.i.kyoto-u.ac.jp

DOI: 10.1093/ietisy/e90–d.9.1335

the rules [5], and present an approach of handling exception using ECA rules [6]. In [4], an automatic control mechanism of workflow execution is proposed, which combines traditional workflow process model and ECA rules to derive a general process control method. Those researches have demonstrated that ECA rules can control workflow automatically and deal with internal exceptions as well. Using ECA rules, workflow can be easily represented into executable forms. However, they mainly focus on internal process control and cannot provide flexibility in controlling task execution. Moreover, their approaches cannot deal with the issue of interorganizational workflow execution which this paper addresses.

Multiagent technology has been used in different ways in workflow management [12], e.g., to fulfill particular roles that are required by different tasks, to serve as part of the infrastructure associated with WfMS. There is also some work that concentrates on agent negotiation between organizations [3], however, how negotiation would affect the local workflows is not discussed. Different from existing work, our agent-based approach is combined with rule-based approach to deal with both control and monitor of tasks within organizations, and interaction among organizations.

3. Overview of the Framework

In previous research, van der Aalst proposes several types of workflow interoperability [1], among which the loosely coupled interorganizational workflow is common in the real world. The model that we discuss is based on loosely coupled interoperability, where the whole interorganizational workflow is made up of local workflows, which may be active in parallel over different organizations. We represent the interorganizational workflow model by using workflow graphs which provide a visual means for users to understand the semantics of the workflow process easily. A graph-based interorganizational workflow model is shown in Fig. 1 by the example of offshore software development. There are two organizations, the outsourcer (a Japanese company) and the supplier (a Chinese company). There is a local workflow for each organization, which represents the process by tasks and the relations among the tasks, such as sequential relation, parallel relations (represented by AND-split and AND-join), iterative relation and so on. A complete workflow includes the detailed description of tasks, such as task executors (human, application or services), related data, scheduling and so on.

Figure 2 shows the overall conceptual framework of the execution mechanism of interorganizational workflow by our approach. The execution mechanism is divided into two parts: the *intra-execution*, which means the workflow execution within a same organization, and the *inter-execution*, which represents the workflow execution (interaction) between organizations. The following three modules are included in *intra-execution*, among which the workflow process execution control module is the core issue of this research.

Workflow process definition and instance module. Process definition creates workflow schema. A process instance is the execution representation of a process and is created based on the process definition. A graph-based workflow editor can be used for workflow definition.

Workflow process execution control module. The execution of the workflow process instances is controlled by the process agent and ECA rules. We use ECA rules to describe all the state transitions of the tasks (e.g., ready, running, suspended, committed and so on) in the workflow process instance. Therefore, the workflow process can be automatically executed according to the ECA rules. The execution of a rule in the ECA rule engine means a step of state change of a certain task and the whole process instance. An ECA rule is executed while the engine is triggered by an event under a certain condition, e.g., during the execution of the workflow process, the start of a task is always triggered by the execution completion of its predecessor task. Events in ECA rules include *internal events* that can directly trigger rules in the engine, and *external events* such as the start of a process instance, the success of executing a task and so on. We control the external events by the process agent. The process agent keeps the information of the execution status of the

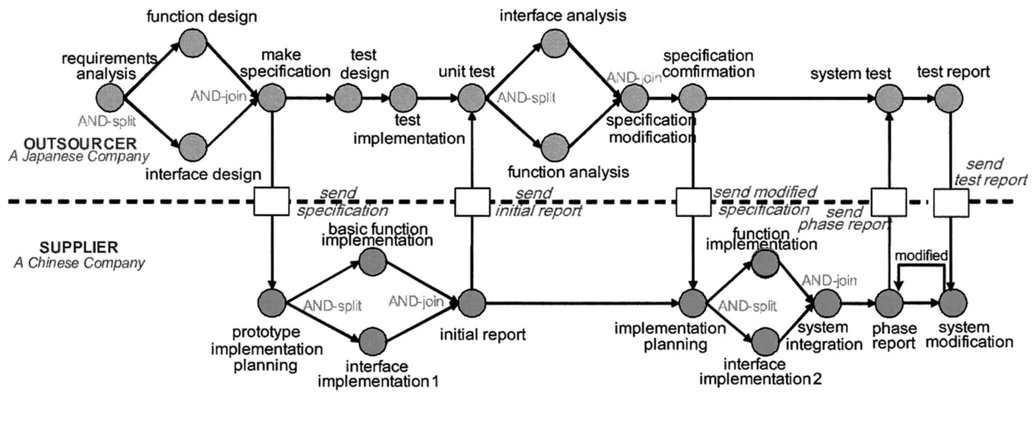

Fig. 1 Graph-based interorganizational workflow model: an example of offshore software development between a Japanese company (the outsourcer) and a Chinese company (the supplier).

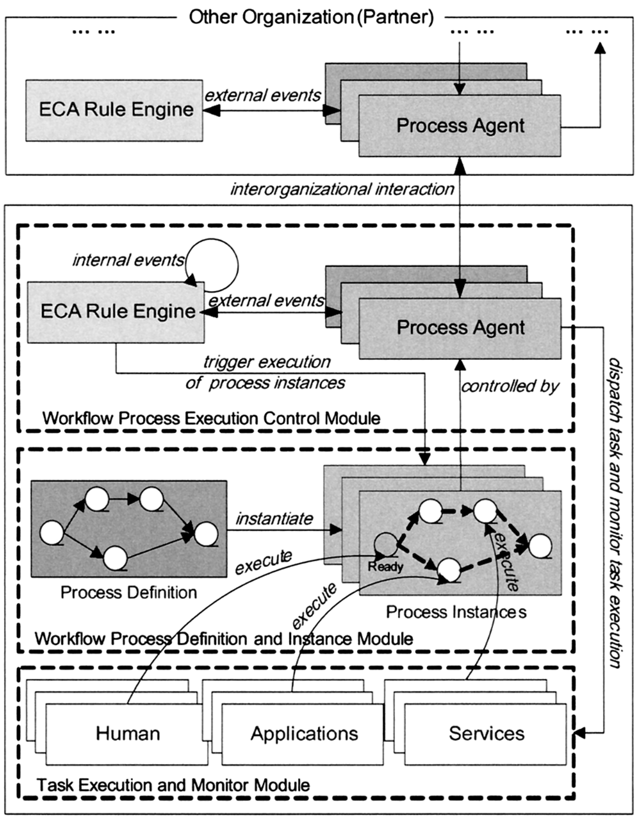
Other Organization (Partner)
ECA Rule Engine
external events
Process Agent
interorganizational interaction
internal events
ECA Rule Engine
external events
Process Agent
trigger execution of process instances
controlled by
Workflow Process Execution Control Module
dispatch task and monitor task execution
instantiate
execute
Ready
execute
execute
Process Definition
Process Instances
Workflow Process Definition and Instance Module
Human
Applications
Services
Task Execution and Monitor Module

Fig. 2 The overall framework of interorganizational workflow execution, including *intra-execution* and *inter-execution*.

process instance by tracing the execution of ECA rules and monitoring the execution of tasks.

Task execution and monitor module. Tasks in workflow process instances are executed by some roles. The process agent dispatches a task when it gets the information that the task is ready for execution, and monitors the whole execution process of the task by interacting with human, applications or services (or their agents). By monitoring task execution, the process agent can get external events and then send such events to the ECA rule engine to trigger some new events.

As for *inter-execution*, interaction between process agents of the organizations is used. When process instance execution of an organization comes to an interaction point with another organization, the process agents of the two organizations interacts with each other to control the execution.

4. ECA Rules for Workflow Execution

Workflow execution can be described by task state transitions. Basic task states include *Disabled*, *Ready*, *Running*, *Committed*, *Aborted* and *Suspended*. The initial state of a task is *Disabled*, which is changed into *Ready* when its predecessor finishes execution, and then is further changed into *Running* if all conditions for execution are satisfied. After the successful execution of the task, its state turns into *Committed*. The state of a task can also be *Aborted* if it is aborted due to some failures, or *Suspended* if it is suspended. The transitions of the tasks can be totally represented and executed automatically by ECA rules. ECA rules, with the form "on *event* if *condition* do *action*" specify to execute the action automatically when the event happens, provided the condition holds. As is shown in Fig. 3, we have an ECA rule engine which contains a set of rules based on the state transition of tasks for each organization.

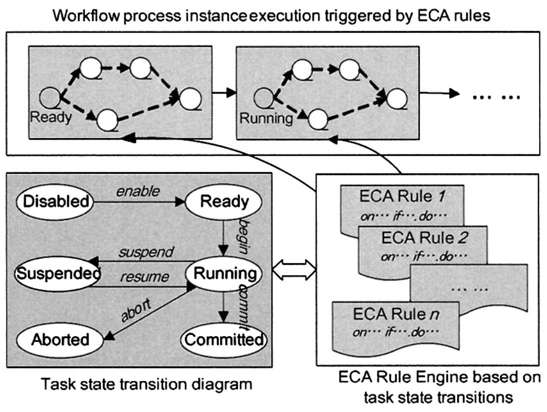

Fig. 3 ECA rules and workflow execution.

In the approach we present in this paper, we first create graph-based interorganizational workflow, through which each organization can understand the workflow process. Further, a set of ECA rules are established based on the graph-based workflow as a rule engine which describes and conducts the whole execution of the workflows. ECA rule engines are distributed among the organizations because each organization needs to have its own engine to execute the local workflow. The execution status of the workflow can be displayed to each organization through the user interface. In this section, we mainly present how to create ECA rules based on the graph-based interorganizational workflow which is described in Sect. 3.1.

4.1 Blocks in the Workflow Process Model

Workflow specification can be understood from a number of different perspectives. In this paper, we focus on the control-flow (process) perspective because this is the essential perspective of the workflow specification [2]. It describes information about tasks and the execution orders (dependencies between tasks). This paper mainly deals with basic constructs of *sequence*, *iteration*, *splits* (AND, OR and XOR) and *joins* (AND, OR and XOR). Based on general constructs, Gokkoca et al. [11] defines seven types of block, namely, *serial*, *and parallel*, *or parallel*, *xor parallel*, *contingency*, *conditional* and *iterative* blocks. By this means, we can transform a graph-based workflow into the block-based workflow. The transformation from graph-based workflow to block-based workflow is similar to approaches in previous researches [5], [10]. Take the local workflow of the supplier in Fig. 1 for example, the graph-based workflow process can be transformed into following blocks: (1) **and parallel block**: *P1 = (basic function implementation & interface implementation 1)*; *P2 = (function implementation & interface implementation 1)*; (2) **iteration block**: *I1 = (Condition(modified); phase report; system modification)*; (3) **serial block**: *S1 = (prototype implementation planning; P1; initial report; implementation planning; P2; system integration; I1)*. From above transforming example, we can observe that the block detection always begins from the most inner part of the workflow process. Moreover, blocks might be mutually embedded. Therefore, the whole workflow process can always be transformed to a single block with other blocks embedded, e.g., the above local workflow is finally turned into a serial block *S1*. We transform the graph-based workflow process into block-based workflow process to use the semantics of blocks to derive general ECA rules and automate the workflow execution.

4.2 Deriving ECA Rules from Blocks and External Events

Since our proposed workflow execution mechanism is mainly based on state transitions of tasks, it is necessary to define the transactions for the state transition diagram in Fig. 3. Therefore we use following transaction primitives: Enable, Begin, Commit, Suspend, Resume, Abort. Further, we define the history base H in the ECA rule engine, which is a finite set of transactions that have occurred during workflow execution. Events in ECA rules include *internal events* and *external events*. Internal events can be described by transaction primitives. External events involve the operations during task execution, including START and END of the process instance, READY, DISPATCH, STOP, SUCCESS, FAIL to execute tasks. Therefore, ECA rules can be divided into *internal ECA rules* and *external ECA rules*. By using transaction primitives and history base, we describe internal ECA rules according to the semantics of blocks in workflow process. Similarly, we use semantics of external events to derive external ECA rules. Internal rules can be executed inside the ECA rule engine without interaction with task execution and monitor module, however, external rules need to interact with the process agent to get the external events. We list external ECA rules in Table 1 and internal ECA rules for blocks in Table 2.

5. Process Agent in the Interorganizational Workflow Model

In this paper, ECA rules are designed based on general constructs in graph-based workflow model, which can be applied in different workflow processes and adaptive to be specialized according to different process definitions. Therefore, in an interorganizational workflow, rules can be used in local workflow for all organizations. Moreover, general ECA rules do not necessarily need modification even when process definition is dynamically changed during execution. It is also very convenient to modify rules in the engine. However, to design general ECA rules for workflow process, workflow data and role model are not defined in rules. Therefore, we use process agent to handle the external events control of each workflow process instance for *intra-execution*. Process agent is also required to interact with process agents of other organizations to provide *inter-execution*. For a loosely coupled interorganizational workflow, process agents are suitable to provide a distributed platform because agents are loosely coupled components forming an open system.

The usage of process agents has advantages for interorganizational workflow execution. Firstly, within each organization, process agent controls and monitors the execution of tasks, which provides flexibility to the local workflow execution. Process agent can dynamically control the external execution status of tasks according to the change of environment. Process agent can also dispatch tasks to suitable execution roles according to the strategies. Secondly, process agents facilitate cooperation among organizations, which preserves the autonomy of organizations. In our approach, we use standard interaction protocols for interoperability among organizations, which is essential for interorganizational workflow. Therefore, organizations cooperate with each other even if they have different process definition tools. By this means, local workflows can be executed distributedly and autonomously.

5.1 Process Agent in Intra-Execution

Within an organization, process agent is in charge of controlling workflow process instance. To achieve this goal, process agent is required to interact with ECA rule engine by external events to continue process state transition. Moreover, process agent needs to dispatch tasks that are ready for execution and monitor tasks that are in execution. Process agent for intra-execution of an organization includes following actions.

(1) Start the workflow process instance when the organization is ready, create an external event $START(PI)$ and send it to the ECA rule engine.

(2) Check the conditions of tasks that are enabled for execution, create an external event $READY(x_i)$ if execution condition of an enabled task is satisfied. $READY(x_i)$ is required for an enabled task to start.

(3) Dispatch tasks that are ready for execution (when receiving the external event $DISPATCH(x_i)$) . Executors of tasks might be pre-defined, negotiated or dynamically adjusted and discovered.

(4) Monitor task execution and receive execution event. Since tasks are executed by executors (human, applications or services), process agent needs to interact with executors or their execution agents to get detailed information of task execution. Therefore, process agent can create external events of execution results such as $SUCCESS(x_i)$, $FAIL(x_i)$ and $STOP(x_i)$. Then, it sends the events to ECA rule engine to trigger new events. Different execution results need different actions. $SUCCESS(x_i)$ might trigger execution of succeeding task, $FAIL(x_i)$ might need some excep-

Table 1 External ECA rules.

Description of External Transitions		External ECA Rules
Event	Semantic Description	
$START(PI)$	Start of process instance enables the block that represents whole workflow process.	ER_1 : ***on*** $START(PI)$ ***if*** *null* ***do*** $Enable_B$
$READY(x_i)$	Ready of an enabled task causes the start of task execution; ready of a suspended task causes the resume of that task.	ER_2 : ***on*** $Enable_{x_i}$ ***if*** $(READY(x_i)) \wedge (Enable_{x_i} \in H)$ ***do*** $Begin_{x_i}$
		ER_3 : ***on*** $Suspend_{x_i}$ ***if*** $(READY(x_i)) \wedge (Suspend_{x_i} \in H)$ ***do*** $Resume_{x_i}$
$STOP(x_i)$	Stop of a running task makes it suspend.	ER_4 : ***on*** $STOP(x_i)$ ***if*** $(Begin_{x_i} \in H)$ ***do*** $Suspend_{x_i}$
$SUCCESS(x_i)$	Success of executing a task makes it commit.	ER_5 : ***on*** $SUCCESS(x_i)$ ***if*** $(Begin_{x_i} \in H)$ ***do*** $Commit_{x_i}$
$FAIL(x_i)$	Failure of executing a task makes it abort.	ER_6 : ***on*** $FAIL(x_i)$ ***if*** $(Begin_{x_i} \in H) \wedge \neg(Commit_{x_i} \in H)$ ***do*** $Abort_{x_i}$
$DISPATCH(x_i)$	The start or resume of a task makes it dispatched for execution.	ER_7 : ***on*** $Begin_{x_i}$ ***if*** $(Begin_{x_i} \in H)$ ***do*** $DISPATCH(x_i)$
		ER_8 : ***on*** $Resume_{x_i}$ ***if*** $(Resume_{x_i} \in H)$ ***do*** $DISPATCH(x_i)$
$END(PI)$	Commit of the block that represents the whole workflow process makes the end of the process instance.	ER_9 : ***on*** $Commit_B$ ***if*** $(Commit_B \in H)$ ***do*** $END(PI)$

Table 2 Internal ECA rules for each block type.

Descriptions of Internal Transitions		Internal ECA Rules
Block Type	Semantic Description	
serial	$B = (x_1; x_2; \ldots; x_n)$ Tasks are executed consecutively. A task is enabled when its prior task succeeds. Block aborts if any task aborts.	IR_1 : ***on*** $Enable_B$ ***if*** $(Enable_B \in H)$ ***do*** $Enable_{x_1}$
		IR_2 : ***on*** $Commit_{x_i}$ ***if*** $(Commit_{x_i} \in H)$ ***do*** $Enable_{x_{i+1}}$
		IR_3 : ***on*** $Commit_{x_n}$ ***if*** $(Commit_{x_n} \in H)$ ***do*** $Commit_B$
		IR_4 : ***on*** $Abort_{x_i}$ ***if*** $(Abort_{x_i} \in H)$ ***do*** $Abort_B$
and parallel	$B = (x_1 \& x_2 \& \ldots \& x_n)$ Tasks are executed concurrently. Block is completed provided completion of all tasks. Block fails if any task fails.	IR_5 : ***on*** $Enable_B$ ***if*** $(Enable_B \in H)$ ***do*** $Enable_{x_i}$
		IR_6 : ***on*** $Commit_{x_i}$ ***if*** $\forall i(i \in n)(Commit_{x_i} \in H)$ ***do*** $Commit_B$
		IR_4 : ***on*** $Abort_{x_i}$ ***if*** $(Abort_{x_i} \in H)$ ***do*** $Abort_B$
or parallel	$B = (x_1 \| x_2 \| \ldots \| x_n)$ Block succeeds if there exists one task that succeeds in executing. Block fails if all tasks abort.	IR_5 : ***on*** $Enable_B$ ***if*** $(Enable_B \in H)$ ***do*** $Enable_{x_i}$
		IR_7 : ***on*** $Commit_{x_i}$ ***if*** $(Commit_{x_i} \in H)$ ***do*** $Commit_B$
		IR_8 : ***on*** $Abort_{x_i}$ ***if*** $\forall i(i \in n)(Abort_{x_i} \in H)$ ***do*** $Abort_B$
xor parallel	$B = (x_1 \| \| x_2 \| \| \ldots \| \| x_n)$ If there is one task that completes, block succeeds and all the other tasks abort.	IR_5 : ***on*** $Enable_B$ ***if*** $(Enable_B \in H)$ ***do*** $Enable_{x_i}$
		IR_9 : ***on*** $Commit_{x_i}$ ***if*** $(Commit_{x_i} \in H)$ ***do*** $(Commit_B) \wedge (\forall j(j \neq i) Abort_{x_j})$
		IR_8 : ***on*** $Abort_{x_i}$ ***if*** $\forall i(i \in n)(Abort_{x_i} \in H)$ ***do*** $Abort_B$
contingency	$B = (x_1, x_2, \ldots, x_n)$ Each task has a priority. Task with highest priority executes first. If fails, try the second highest, the third... Block succeeds if any task completes.	IR_1 : ***on*** $Enable_B$ ***if*** $(Enable_B \in H)$ ***do*** $Enable_{x_1}$
		IR_{10} : ***on*** $Abort_{x_i}$ ***if*** $(Abort_{x_i} \in H)$ ***do*** $Enable_{x_{i+1}}$
		IR_7 : ***on*** $Commit_{x_i}$ ***if*** $(Commit_{x_i} \in H)$ ***do*** $Commit_B$
		IR_{11} : ***on*** $Abort_{x_n}$ ***if*** $(Abort_{x_n} \in H)$ ***do*** $Abort_B$
conditional	$B = ((C_1, x_1) \| (C_2, x_2) \| \ldots \| (C_n, x_n))$ All tasks have conditions compared with *or parallel* block. Only the task that satisfies the condition executes.	IR_{12} : ***on*** $Enable_B$ ***if*** $(Enable_B \in H) \wedge C_i$ ***do*** $Enable_{x_i}$
		IR_{13} : ***on*** $Abort_{x_i}$ ***if*** $(Abort_{x_i} \in H) \wedge C_i$ ***do*** $Abort_B$
		IR_{14} : ***on*** $Commit_{x_i}$ ***if*** $(Commit_{x_i} \in H) \wedge C_i$ ***do*** $Commit_B$
iterative	$B = (Condition(C); x_1; x_2; \ldots; x_n)$ The iterative condition gives a while loop between the start task and the end task on *serial* block. The loop continues until the iterative condition becomes false or any task aborts.	IR_1 : ***on*** $Enable_B$ ***if*** $(Enable_B \in H)$ ***do*** $Enable_{x_1}$
		IR_2 : ***on*** $Commit_{x_i}$ ***if*** $(Commit_{x_i} \in H)$ ***do*** $Enable_{x_{i+1}}$
		IR_{15} : ***on*** $Commit_{x_n}$ ***if*** $(Commit_{x_n} \in H) \wedge C$ ***do*** $Enable_{x_1}$
		IR_{16} : ***on*** $Commit_{x_n}$ ***if*** $(Commit_{x_n} \in H) \wedge \neg C$ ***do*** $Commit_B$
		IR_4 : ***on*** $Abort_{x_i}$ ***if*** $(Abort_{x_i} \in H)$ ***do*** $Abort_B$

tion handling process, and if $STOP(x_i)$ occurs, the executing task needs to wait to be resumed.

(5) End the workflow process instance if all the tasks are successfully executed (when receiving the external event $END(PI)$).

5.2 Agent Interaction Protocols for Inter-Execution

Interorganizational workflow environments can be modeled as multiagent systems. Within each organization, agent remains autonomy and heterogeneity. Across organizations there are interactions, which can be regarded as agent interaction problem. We use protocol to handle interactions that take place during workflow execution. The protocols involve all the organizations that have interaction with each other to address specific purposes, e.g., sending and receiving software specification between organizations in an offshore software development environment.

To achieve flexibility among organizations, a process agent is required not only to deal with basic interaction such as sending or receiving results, but also to support coordination and negotiation between organizations if the results need to be modified. Autonomous agents can negotiate with each other to execute the allocated task flexibly and dynamically [9]. In [8], several negotiation protocols are proposed such as auction protocol, heuristic protocol, argumentation protocol, contract net protocol and so on. As for the intercultural software development issues, contract protocol is most suitable since it deals with cooperative negotiation [3]. Figure 4 shows the examples.

The actions of process agent interaction among organizations would affect intra-execution. Figure 5 shows scenarios of how process agent interaction (as shown in Fig. 4) affects intra-execution. Simple agent interactions are execution conditions for certain tasks and the interaction actions trigger ECA rules in intra-execution. However, interaction with negotiation will cause internal state changes of certain tasks and therefore workflow adaptation should be conducted by process agent for further negotiation process. For example, in protocol (b) in Fig. 4, when the Outsourcer process agent sends *requestRevise()*, the Supplier process agent would change its local workflow to repeat executing current task with updated conditions. In some cases, a part of workflow process might be changed in the process of negotiation according to the strategies of process agents.

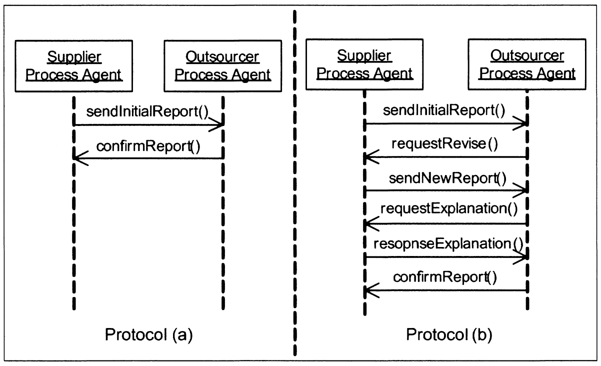

Fig. 4 Example of agent interaction protocols: (a) simple interaction protocol of "send" and "receive"; (b) interaction protocols with coordination and negotiation between process agents.

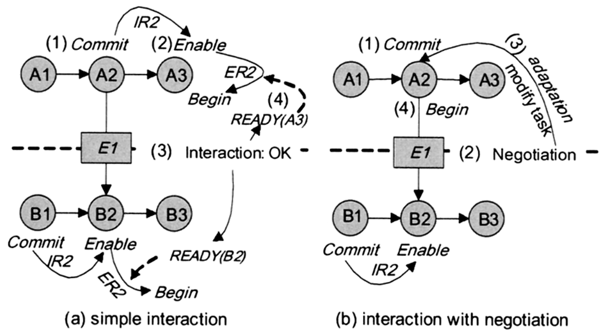

Fig. 5 Scenarios of intra-execution effects by process agent interaction in inter-execution.

6. Case Study and Discussion

We study the case of offshore software development to explain the proposed approach. First, a graph-based interorganizational workflow model is created as shown in Fig. 1. Next, the graph-base workflow of each organization is automatically transformed into the block-based workflow. Then, the block-based local workflows of all the organizations are executed distributedly. The local workflows interact with each other at certain points by process agent interaction protocols. Table 3 shows part of workflow execution steps of the Supplier in the example of Fig. 1. As is presented in Sect. 4.1, the block-based workflow contains four blocks: S_1, P_1, P_2 and I_1, among which S_1 is the main stream. In the table, every step shows the current active block and tasks, current event, the ECA rule that the event would trigger, and the action of process agent. The event history stores all the executed events. Process agent of the local workflow process instance interacts with process agent from other organizations when there is an interaction point. By the control of ECA rule engine and process agent, the whole interorganizational workflow can be distributed for execution among organizations. Further, we explain the inter-execution by

Table 3 Interorganizational workflow execution example (the case of Supplier in Fig. 1).

	Block	Active Task	Event	ECA Rule	Action of Process Agent
0	–	–	–	–	Confirm receiving specification from process agent of the Outsourcer.
1	–	–	–	–	Start process instance and send external event $START(PI)$ to ECA rule engine.
2	–	–	$START(PI)$	ER_1	–
3	S1	–	$Enable_{S1}$	IR_1	–
4	S1	x_1: prototype implementation planning	$Enable_{x_1}$	ER_2	Check if the task is ready for execution. Create $READY(x_1)$ if condition is satisfied.
5	S1	x_1	$Begin_{x_1}$	ER_7	–
6	S1	x_1	$DISPATCH(x_1)$	–	Dispatch task for execution.
7	S1	x_1	–	–	Monitor the task execution and send external event $SUCCESS(x_1)$ to ECA rule engine if the task is successfully executed.
8	S1	x_1	$SUCCESS(x_1)$	ER_5	–
9	S1	x_1	$Commit(x_1)$	IR_2	–
10	S1	x_2: P1 (regarded as a "task" in S1)	$Enable(x_2)$	IR_5	–
11	P1	x_{21}: basic function implementation, x_{22}: interface implementation 1	$Enable_{x_{21}}$ $Enable_{x_{22}}$	ER_2	Check if the parallel tasks are ready for execution. Create $READY(x_{21})$ and $READY(x_{21})$ if execution conditions are satisfied.
12	...	...	...	...	...

the process agent interaction of *send initial report* in Fig. 4. The whole process is as follows. With coordination or negotiation, the whole interorganizational workflow execution becomes flexible.

(1) When process agent of the Supplier PA_S knows commitment of the task *initial report* by the ECA rule engine, it sends initial report to process agent of the Outsourcer PA_O.

(2) PA_O receives initial report from PA_S and replies to PA_S according to the received initial report. If the received report is OK, PA_O will send a confirmation to PA_S. Otherwise, it will send a revise request or explanation request to PA_S with requirement.

(3) If PA_S receives a confirmation, the READY condition of its next task *implementation planning* will be satisfied and a new rule will be triggered. The interaction steps end.

(4) If PA_S receives a revise or explanation request from PA_O, it will adjust the local workflow process and intra-execution state. The specification and state of task *initial report* will be changed for re-execution according to the received request.

(5) Repeat from step (1).

In our approach, we design ECA rules based on general definition of blocks to provide an adaptable and modular approach to achieving workflow execution. The general ECA rules can be adopted in different workflows because they are based on blocks. Further, in order to control and monitor the whole workflow execution process more flexibly, we introduce agent technology into our mechanism. We use process agent to control the execution of each task in a local workflow. Though the task dispatch mechanism is not the focus of this paper, process agent can provide flexibility in dealing with dynamic changes and exceptions. We also use agent interaction protocols to deal with the interactions between different organizations so that organizations can not only send basic message to each other, but also manage to coordinate and negotiate with each other. As a result, the whole mechanism can provide the execution of interorganizational workflow automation, flexibility, and adaptation. Moreover, our approach supports distributed execution of local workflows of different organizations, which provides autonomy of organizations.

Despite the effectiveness, there are some limitations in the proposed approach. Although ECA rules designed in this paper can be generally used, the assumption is that workflow model is graph-based. When designing the internal ECA rules, we only deal with basic types of blocks. How to extend the proposed approach to other types of workflow models and workflow model with more complicated structure might be interesting issues in future research. Moreover, although the agent-based approach provides autonomy and flexibility, there are still open issues that we have to deal with in future work. For example, when process agents are conducting coordination, some criteria like workflow soundness and optimal modification must be considered for local workflow adaptation.

7. Conclusion

Interorganizational workflow has been becoming more and more important in the global economic environment. However, there are several major challenges in the interorganizational workflow execution problem, including flexibility, adaptation and distribution. These issues are rarely covered together in previous research. In this paper, we propose a new framework for the interorganizational workflow execution based on process agents and ECA rules to address above issues.

To design the execution mechanism of interorganizational workflow, we divide the whole workflow execution into two parts: the *intra-execution* and the *inter-execution*. We design execution mechanisms for the two parts to achieve the goal. In this approach, the whole interorganizational workflow is modeled as a multiagent system with a process agent in each organization. Therefore, local workflows of organizations are distributed for execution and interaction with each other at certain points. For each local workflow, a method of transforming the graph-based workflow model into block-based workflow model is applied to derive general ECA rules from blocks. We further design ECA rules to control internal state transition and use process agent to control the external state transition of tasks in the local workflow process. Workflow execution across organizations is achieved by process agent interaction protocols.

The proposed approach can provide automatic execution of interorganizational workflow with flexibility and adaptation. It can also provide autonomy for local workflows. A case study of offshore software development is provided to demonstrate the effectiveness of our approach. Our future work will be the implementation and evaluation of the proposed approach based on process agents and ECA rules.

Acknowledgments

This research was partially supported by a Grant-in-Aid for Scientific Research (A) (18200009, 2006–2008) from Japan Society for the Promotion of Science (JSPS). The authors wish to thank Dr. Ahlem BEN HASSINE at National Institute of Information and Communications Technology for her discussion and advice on this paper.

References

[1] W.M.P. van der Aalst, "Process-oriented architectures for electronic commerce and interorganizational workflow," Information Systems, vol.24, no.8, pp.639–671, Dec. 1999.

[2] W.M.P. van der Aalst and M. Weskez, "Advanced topics in workflow management: Issues, requirements, and solutions," J. Integrated Design and Process Science, vol.7, no.3, pp.49–77, 2003.

[3] E. Andonoff and L. Bouzguenda, "Agent-based negotiation between partners in loose inter-organizational workflow," Proc. IEEE/WIC/ACM International Conference on Intelligent Agent Technology, pp.619–625, IEEE Computer Society, Washington, DC,

2005.

[4] J. Bae, H. Bae, S. Kang, and Y. Kim, "Automatic control of workflow processes using ECA rules," IEEE Trans. Knowl. Data Eng., vol.16, no.8, pp.1010–1023, Aug. 2004.

[5] F. Castai, S. Castano, M. Fugini, I. Mirbel, and B. Pernici, "Using patterns to design rules in workflows," IEEE Trans. Softw. Eng., vol.26, no.8, pp.760–765, Aug. 2000.

[6] F. Casati, M. Fugini, and I. Mirbel, "An environment for designing exceptions in workflows," Information Systems, vol.24, no.3, pp.255–273, May 1999.

[7] M. Divitini, C. Hanachi, and C. Sibertin-Blanc, "Inter-organizational workflows for enterprise coordination," in Coordination of Internet Agents, ed. A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf, pp.46–77, Springer-Verlarg, 2001.

[8] N. Jenning, P. Faratin, A. Lomuscio, S. Parsons, C. Sierra, and M. Wooldridge, "Automated negotiation: Prospects, methods and challenges," Int. Journal on Group Decision and Negotiation, vol.10, no.2, pp.199–215, 2001.

[9] Y. Jiang and J. Jiang, "A multi-agent coordination model for the variation of underlying network topology," Expert Systems with Applications, vol.2, no.29, pp.372–382, 2005.

[10] J. Mendling, K. Lassen, and U. Zund, "Transformation strategies between block-oriented and graph-oriented process modeling languages," Technical Report, JM-200510-10, WU Vienna, Oct. 2005.

[11] E. Gokkoca, M. Altinel, I. Cingil, E.N. Tatbul, P. Koksal, and A. Dogac, "Design and implementation of a distributed workflow enactment service," Proc. International Conference on Cooperative Information Systems (CoopIS), pp.89–98, 1997.

[12] M.P. Singh and M.N. Huhns, "Multiagent systems for workflow," International Journal of Intelligent Systems in Accounting, Finance and Management, vol.8, pp.105–117, 1999.

[13] WfMC, Workflow Management Coalition Terminology and Glossary (WFMC-TC-1011), Technical Report, Workflow Management Coalition, Brussels, 1996.

Donghui Lin received his ME degree in computer science and engineering at Shanghai Jiao Tong University, China in 2005. He is currently a PhD candidate in social informatics at Kyoto University, Japan. His research interests include multiagent systems, workflow management technologies, organization theory and intercultural collaboration.

Huanye Sheng is a professor of computer science and engineering at Shanghai Jiao Tong University, China. His research interests include intelligent human-computer interface, information economics, modern logistics management and natural language processing.

Toru Ishida is a professor of social informatics at Kyoto University, a leader of National Institute of Information and Communications Technology Language Grid project, and a guest professor at Shanghai Jiao Tong University. His research interests include autonomous agent and multiagent systems, semantic Web service and intercultural collaboration.