2

Studies on Flexible Workflow Model with Database Technologies

Takeo Kunishima

December 1996

Abstract

Workflow management system (WFMS), one of the groupware supporting asynchronous distributed interaction, becomes remarkable not only in practical business area but also in research area.

WFMSs mainly support well-structured collaborative works using explicitly defined flows of works (workflows). They controls the invocation order in workflows automatically, and manages several resources of the work.

As a number of workflows run concurrently using shared resources of organizations, transaction management with concurrency control is an important technology for WFMSs. In this sense database technologies are indispensable for the infrastructure of WFMSs, and many researchers have studied about transaction management in WFMSs.

Data management is another important role of database technologies in WFMSs. As WFMSs must manage many data such as workflow descriptions, status of progress, activity environments, and activity products, WFMS products use DBMSs in their backends. However, the role of DBMSs in WFMSs is no more than as repositories. There is no standard data model for workflows even in the research level.

In this thesis we propose a flexible framework of workflow management suitable for database technologies, *workflow base*. In this model,

- A workflow is defined as a set of objects (*activity objects*), each of which corresponds with the unit of work in workflows. Two kinds of flows, horizontal flows and vertical flows, are defined. Both flows are treated as constraints among activity objects, hence they are created dynamically from the definitions of activity objects. This makes database management of workflows to be easier than ordinary workflow models.

- Integrity constraints over workflows are defined on a set of activity objects in database. They can be checked in a similar way with ordinary integrity check on database management systems.

- The concept of workflow instantiation is also defined based on generalization/specialization hierarchies of workflows. This makes relationships among workflows clearer, and workflows more reusable.

- Execution model of workflow base is defined based on production systems. This model deals with dynamic dispatch of subworks as well as ordinary static flows in the same manner.

Database features of workflow base are discussed from the various viewpoints. Loopback flows are defined using ECA rules, a basic concept of active databases; extensions on workflow base dealing with time constraints and resource constraints are introduced; database operations over workflows based on relational algebra are also introduced, which realize general purpose view functions and query functions in workflow management systems; agents as an executer of the units of work are defined formally as a problem solver in a heterogeneous distributed environment. Finally we give a system architecture of workflow base.

Acknowledgment

First I would like to thank deeply my supervisor, Professor Yahiko Kambayashi at Kyoto University. He gave me the opportunity to study databases and workflow management systems, and continuously suggested many helpful ideas since then. He also extensively reviewed this thesis and suggested many improvements.

I would also express my thanks to Professor Katsumasa Watanabe at Nara Institute of Science and Technology (NAIST). He continuously supported my study in Kyoto University and gave me the best environment for my doctoral research.

I would like to thank Professor Shuzo Yajima at Kyoto University. I had studied with him at master course of Kyoto University. He also introduced me to Professor Kambayashi for carrying out my wishes to begin researches about databases.

Associate Professor Kazumasa Yokota at Kyoto University discussed about almost all part of this thesis and gave many helpful perspectives to me about formalization of workflow model. Workflow model in Chapter 3 would not be constructed without the discussions with him. I would like to acknowledge his contribution.

Associate Professor Shinji Kimura, Mr. Kazuyoshi Takagi, and the members of Watanabe Laboratory at NAIST, cooperated to give me a good environment for my study.

There are many other people I would like to thank. I discussed many ideas with Associate Professor Hiroshi Nunokawa at Miyagi University of Education and the members of his research group, especially Mr. Masahiro Hiji and Mr. Norihisa Segawa. They have the similar research directions with me in the area of computer-supported cooperative work, information media, and databases. Mr. Segawa now studies with me at NAIST. Associate Professor Tsuneo Ajisaka at Kyoto University, Dr. Hiroyuki Tarumi at NEC Corp., and Mr. Isao Kojima at Electrotechnical Laboratory (ETL) suggested many useful comments from the viewpoints of software engineering, groupware, and active database systems, respectively. Dr. Hiroki Takakura at NAIST has helped me both in Kambayashi Laboratory and in NAIST. Associate Professor Masatoshi Arikawa (now at Hiroshima City University), Ms. Masako Watanabe (Kyoto University), Mr. Shintaro Meki (now at Okayama Prefectural University), Mr. Hideyuki Takada (now at Mitsubishi Electric Corp.) also helped me when I was in Kambayashi Laboratory. The members of Kambayashi Laboratory offered the computer environments to me every time I go to Kyoto University.

Finally, I would like to thank all people to help me, especially my family and my friends in Choir Hamoru KOBE.

Contents

Chapter 1

Introduction

1.1 Background

During the recent two or three years, computer networks permeate into several areas of our everyday life.

The Internet started just as a computer wide area network among the computer research sections of universities, companies, and some public organizations. At that time, the users of the Internet communicate each other through the Internet by electronic mails, network news, or character-based realtime applications such as electronic phones. These are only the usages of the Internet.

However, this situation around the Internet has been drastically changing as several technologies are progressed in the areas of computer and digital networks. Now the Internet is no longer an experimental object for computer researchers; it is an infrastructure of several business activities, including personal activities. Applications on the Internet has also changed from character based to multimedia based. World Wide Web (WWW), a multimedia hypertext over the Internet, is an example of hot multimedia based applications among the Internet users.

Groupware is another hot application over the Internet. It is a general term for the technologies supporting human communications or cooperative work over the computer networks, such as electric meetings, cooperative writing, etc. Ellis [EGR91] defined the term "groupware" as:

> Computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment.

Although the concept of groupware is proposed in 1978 [JLJL82], it had not been practical until the recent years because of the lack of its infrastructures, such as computer powers and the network bandwidth. As this problem has been gradually solved, many researches and developments of groupware are carried out.

Ellis categorizes groupware into four types from notions of time and space [EGR91]. They are:

- *Face-to face interaction.* This type supports cooperation in the same place and the same time. Electric meeting room system [SFB+87, SBF+87] can be shown as its example.
- *Asynchronous interaction.* This type supports cooperation in the same place but in the different times.
- *Synchronous distributed interaction.* This type supports cooperation in the same time but in the different places. Group editor [FS86] and distributed electric meeting system [CMB+90] are some of the examples.
- *Asynchronous distributed interaction.* This type supports cooperation in the different times and the different place. These are the examples: task coordination systems [WF86, FGHW88], information filtering systems [MGL+87], office procedure systems [IO91, Suc83, CL84], and hypertexts [CB88].

Workflow management system (WFMS) [GHS95] is one of the remarkable groupware not only in practical business area but also in research area. WFMSs, belonging into the groupware supporting asynchronous distributed interaction according to Ellis's categorization, mainly support structured collaborative works using explicitly defined flows of works (workflows). They controls the invocation order in the workflows and manages several resources of the work automatically.

As a number of workflows run concurrently using shared resources of organizations, transaction management with concurrency control is an important technology for WFMSs. In this sense database technologies are indispensable for the infrastructure of WFMSs, and many researchers have studied about transaction management in WFMSs [GH94, RS94, KS95, AAA+96].

Another contribution of database technologies for WFMSs is data management. As WFMSs must manage many data such as workflow descriptions, status of progress, activity environments, and activity products, WFMS products use DBMSs in their backends. However the role of DBMSs in WFMSs is no more than as repositories. There is no standard data model for workflows even in the research level, though some researchers pointed out the importance of data sharing in computer supported cooperative work [GS87] and workflow data models [AS96].

If DBMSs support WFMSs more closely in their data management, WFMSs can provide more useful and powerful functions. This is our standpoint[KK95, KY96, YKN96, YKN97]. DBMSs' supports bring the following advantages into WFMSs:

- By managing all workflow descriptions in one DBMS, it is easy to resolve duplications or conflicts among the workflows. This leads to efficient workflow management.
- Management of workflow hierarchies makes reuse of workflows possible.
- Powerful view functions can be provided. For example, private schedule can be obtained as a view of workflows.
- Workflows can be easily updated, changed, or reorganized even if they are in progress.
- An integrated work environment can be provided by managing both product data and process data.

As real offices are open [Hew86], groupware should support office works flexibly, even in the procedural works. Hence these advantages are useful for flexible workflow management systems.

In this thesis we propose a flexible framework of workflow management suitable for database technologies, *workflow base*. The features of this model and the merits are as follows:

1. A workflow is defined as a set of objects (*activity objects*), each of which corresponds with the unit of work in workflows. This makes database management of workflows to be easier than ordinary workflow models.
2. Two kinds of flows, horizontal flows and vertical flows, are defined. Ordinary workflows can be described by using these flows. Both flows are treated as constraints among activity objects, hence they are derived dynamically from the definitions of activity objects.
3. The concept of generalization/specialization workflow hierarchies is introduced. This makes relationships among workflows clearer, and

workflows more reusable. The concept of workflow instantiation is also defined based on these hierarchies.
4. A rule-based execution model of workflow is defined. This model deals with dynamic dispatch of subworks as well as ordinary static flows in the same manner.
5. Integrity constraints over workflows are defined. As they are defined on a set of activity objects in database, they can be checked in a similar way with ordinary integrity check on database management systems.

We also discuss several database features of workflow base from the various viewpoints. Loopback flows are defined using ECA rules, a basic concept of active databases; extensions on workflow base to deal with time constraints and resource constraints are defined; database operations over workflows based on relational algebra are introduced, which realize general purpose view functions and query functions in workflow management systems; agents as an executer of the units of work are defined formally as a problem solver in a heterogeneous distributed environment. And finally, we discuss about system architecture of workflow base.

1.2 Outline of the Thesis

The remainder of this thesis is as follows. In Chapter 2, preliminaries for the discussions of the latter chapters are provided. First we explain the basic concepts of workflow management systems and the requirements to workflow management systems are discussed. Secondly, brief explanations about definite clauses, logic programming, production rules, and ECA (Event-Condition-Action) rules are shown. We utilize these concepts in the definition of workflow base and its extensions. Finally, we explain the concept of *agent* as a heterogeneous distributed cooperative problem solver. The term "agent" is used in several contexts of distributed cooperative environment. We use "agent" as an executer of the units of work in workflows. The concept of agent is defined from this context. This definition is utilized in Chapter 6.

Workflow base, a formal model of workflow database, is proposed in Chapter 3. This is the core model of this thesis. Its basic idea is that a workflow is represented as a set of units of work and the constraints among the units. This idea makes workflow management using database technologies to be easier than other workflow models. Section 3.1 defines the structure of workflows in workflow base. *Activity objects* representing the units of work in workflows, two kind of flows based on message passing between activity objects and on part-of hierarchy over activity objects, and several constraints on *workflow templates* are defined in this section. Section 3.2 gives an execution model of workflow templates based on production system. In Section 3.3 instantiation concept of workflow templates is defined using specialization hierarchies of workflows. Based on the preceding discussions, workflow base is defined in Section 3.4.

Some extensions on workflow base are discussed in Chapter 4. Although workflow base defined in Chapter 3 supports basic functions indispensable for workflows, some extended features such as loopbacks, time constraints, resources constraints, etc., are necessary for workflows. We discuss these extensions: loopback flows based on ECA rules are defined in Section 4.1; time constraints and some applications on them such as scheduling are shown in Section 4.2; the constraints of resources on workflows are discussed in Section 4.3. We also discuss a method for resource reallocation in the time of violations in resource constraints in

this section. And finally, we show that exclusive lock mechanism causes horizontal flows dynamically in Section 4.4.

Chapter 5 describes about database operations on workflow base. If the operations on workflows are provided, workflow management will be more flexible and powerful: View functions on workflows, dynamic change on workflows with keeping integrity constraints, etc. As workflow base manages workflows in a style suitable for database management, the operations on workflows can be easily provided. In this chapter, an operation set based on relational algebra is proposed.

Chapter 6 is devoted into reconsiderations about agents. In the previous chapters, we consider agents as an executer of the units of work. We give another definition of agents in workflow base, as a problem solver enclosed in a capsule. As workflow base is organized in heterogeneous distributed environment, agents are essentially also heterogeneous. Hence in workflow base, mechanisms that heterogeneous agents must coordinate each other. We show such a mechanism by providing an *environment* for message-passing between agents.

We discuss about how to implement workflow base, mainly from system architecture point of view in Chapter 7. Though workflow base is closely related to database systems, it has various features not found in traditional database systems, such as an execution model based on production systems. We first investigate system requirements to implement workflow base, and then show a system architecture of workflow base.

Related researches are shown in Chapter 8, with comparisons to workflow base. There are many researches about workflow management systems. Moreover, there are also similar concepts as workflows in various research areas, such as groupware, process modeling, database, and software process engineering. In this chapter, we pick up several related researches from three research areas — groupware, process modeling, and database — and compare them with workflow base especially from the workflow model point of view.

We conclude with discussions of future work in Chapter 9.

Chapter 2

Preliminaries

2.1 Workflow Management Systems

2.1.1 Basic Concepts

Workflow is a concept for automating or reengineering business processes in an organization [AS94, GHS95, Sch96]. However, there is little agreement for the features a workflow management system must provide. This is because the term "workflow" uses in various situations: business process specification, business process automation, business process reengineering, etc. Moreover, in the domain of workflow management, products are developed earlier than researches with strict definitions or theorems. Here are a few examples of workflow products [GHS95]: Lotus Notes from Lotus Development Corp., FloWare from Recognition International, Staffware from Staffware Corp., DocuFLOW from Inventor Inc., WorkMAN from Reach Software Corp., ActionWorkflow from Action Technologies, TeamLinks for Pathworks from Digital Equipment Corp., etc.

In 1996, Workflow Management Coalition [wfC], international organization for standardization of workflow software technology, published a document about workflow software terminology [Wor96]. This document explains almost all important concepts of workflow model.

A workflow defines a collection of *process instances* organized to accomplish some business processes, the order of process instances to be invoked, and several conditions under process instances invocations. In [Wor96] there are five kinds of the invocation order:

- AND-Split. A single thread of control splits into two or more parallel processes.
- AND-Join. Two or more parallel executing processes converge into a single common thread of control.
- OR-Split. A single thread of control makes a decision upon which branch to take when encountered with multiple alternative workflow branches.
- OR-Join. Two or more alternative processes workflow branches reconverge to a single common process as the next step within the workflow.
- Iteration. The repetitive execution of one or more workflow processes until a condition is met.

Each process instance is performed by human, by a team of humans, or by softwares. Moreover, each process instance may have subprocess instances. Therefore workflows can be constructed hierarchically.

A workflow is often described as a directed graph whose nodes and arcs represent business tasks and their invocation order respectively. Note that the graph may have cycles because business tasks sometimes repeats until their aims are attained. Figure 2.1 shows an example of

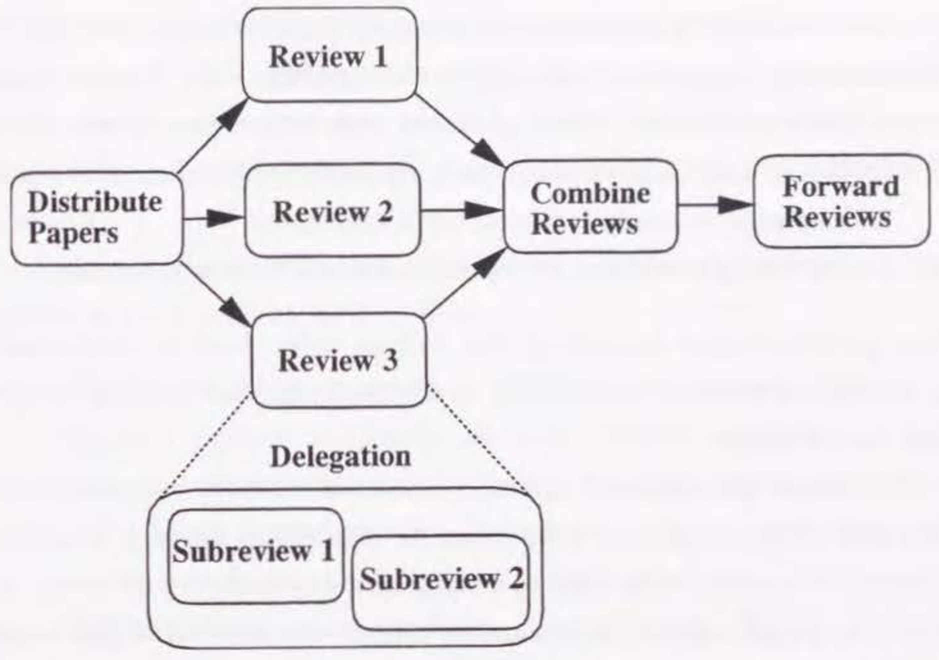

Figure 2.1: Workflow Example: Paper Review Workflow

workflow. In this workflow, the processes "Review 1", "Review 2", and "Review 3" are executed concurrently. Hence, the invocation order from "Distribute Papers" to "Review 1", "Review 2", and "Review3" should be "AND-Split"; the order from "Review 1", "Review 2", "Review 3" to "Combine Reviews" should be "AND-Join".

Workflow management is a technology to support automation or reengineering using workflows. It mainly organized from these four procedures:

1. defining workflows by analyzing the business processes to be managed.
2. instantiating workflows by assigning several variables with real in-

stances — for example, organization role name with human name.

3. executing instantiated workflows. It is similar with the execution of finite automaton. First an initial task (state) is set, and when receiving transition events, tate transitions are invoked with some actions such as sending E-mail for notifications.
4. restructuring workflows for optimization of business processes.

These procedures are invoked at the endless order; that is, restructuring workflows results into defining new workflows, and starting a new procedure sequence.

A workflow management system consists of software components to store and interpret process definitions, to create and manage workflow instances to execute them, and to control the interactions with users. In order to store process definitions and to manage execution status of workflows, databases are indispensable component of workflow management systems.

2.1.2 Previous Researches

Workflow management systems provide methodologies to support [GHS95]:

1. business process modeling to capture business processes as workflow specifications.
2. business process reengineering to optimize specified processes.
3. workflow automation to generate workflow implementations from workflow specifications.

In addition, workflow management systems support asynchronous distributed cooperative work whose structure is well-defined in most cases.

Another methodology of workflow management systems is transaction management. That is, cooperative work supported by workflow management systems is routine work, and many numbers of transactions, some of these sharing office resources, run concurrently during workflow management.

Hence, in the workflow area, researches and developments had been advanced from these three directions:

- *Process modeling approach.* This approach mainly focuses on business process modeling, business process automation, business process reengineering including dynamic change mechanism, etc. Many formal models of business process have been proposed based on this approach: Petri-net based [Ish86, IO91, BN95, EKR95], state transition diagram based [HK89, HLN+90, SAM91, JMR92, Swe93, INMS96], distributed knowledge-base based [TLA91, JMR92, Ple95, MCC95, Rob96], etc.
- *Groupware approach,* [SMK90, KCM91, MMWFF92, Mah93, TTY95] for example. This approach views workflow management system as a kind of groupware supporting cooperative activities of human beings in a distributed environment. It mainly focuses on communication theory [WF86, Mah93], flexible support for human communications [ML84, BTKdlT93, BN95, IHH96], system architecture including GUI [KCM91], toolkits for implementing WFMSs [FKB95], etc.
- *Database approach,* [GH94, RS94, KS95, AAA+96] for example. This approach mainly views from transaction management from database point of view, such as transaction model supporting transaction hierarchies and concurrency control,

2.1.3 Benefits for Workflow Management by Using Database Technologies

As described in the previous section, databases bring some benefits into workflow management systems, especially in transaction management area. Transaction management in databases are useful when workflow management systems become very large, for example involving thousand of users in wide area networks.

However, as mentioned in [AS96], applying database technologies to workflow management bring several benefits other than transaction management, such as interoperability among several workflow management systems. Interoperability is one of the very serious problems in workflow management area, because workflow management technologies have been leaded by commercial products, no interoperability in each other.

Database technologies will be helpful for resolving interoperability problem in workflow management by providing general purpose workflow manipulation languages, like SQL in data management. Therefore database technologies can provide the infrastructure for managing workflows in more general and flexible way than in conventional WFMSs technologies.

2.2 Definite Clauses, Production Rules, and ECA Rules

In this thesis, we use definite clause based logic programming such as pure Prolog, production rules, and ECA (Event-Control-Action) rules to control workflows. We assume that the readers are familiar with them. Here we introduce their basic concepts, definitions, and behaviors.

2.2.1 Definite Clauses and Logic Programming

Any closed formula in the first order logic can be transformed into a set of clauses, each of which is in the following form:

$$p_1 \vee p_2 \vee \cdots \vee p_n \vee \neg q_1 \vee \neg q_2 \vee \cdots \vee \neg q_m$$

where p_i, q_j $(0 \leq i \leq n,\ 0 \leq j \leq m)$ are atomic formulas (atoms).

A clause with at most one positive atom is called a *clause*, written as follows:

$$\begin{array}{l} p \leftarrow q_1, q_2, \cdots, q_m. \\ p \leftarrow . \\ \leftarrow q_1, q_2, \cdots, q_m \end{array}$$

where p is a positive atom and q_i is a negative atom. The first and the second clauses are called *definite clauses*, a set of which is called a *program* or a *database*. The third clause is called a *goal*. Without loss of generality, we can assume that the second clause does not have any variables, as in extensional databases of deductive databases. The second one is called a *fact*, and the first one is called a *rule*. The left hand side of $\leftarrow$ is a *head* and the right hand side of $\leftarrow$ is a *body*.

As usual, we can define three kinds of formal semantics: declarative semantics as the minimum Herbrand model, procedure semantics such as SLD resolution, and least fixpoint semantics. Here we focus on the procedure semantics. Given a query ?-q to a program P, query processing is represented as the following sequence of pairs of a set of goals and a set of substitutions:

$$(G_0, \emptyset) \rightarrow (G_1, S_1) \rightarrow \cdots \rightarrow (G_i, S_i) \rightarrow \cdots \rightarrow (\emptyset, S_n)$$

where $G_0 = \{q\}$, and if $p \leftarrow p_1, p_2, \cdots, p_m \in P$, $q' \in G_i$, and $p\theta = q'\theta$, then

$$G_{i+1} = G_i \cup \{p_1\theta, p_2\theta, \cdots, p_m\theta\}$$
$$S_{i+1} = S_i \cup \theta$$

To control workflows, we treat an atom corresponding to a work: i.e., a rule $p \leftarrow p_1, p_2, \cdots, p_n$ means that works $p_1, p_2, \cdots, p_n$ should be done to complete a work p. In other word, if p is activated by and receives inputs in some arguments as a result of unification, p activates subgoals $p_1, p_2, \cdots, p_n$ and sends inputs to them as a substitution. If subgoals are executed successfully, they return outputs as new substitution. If a subgoal is defined by another rule, it activates the corresponding rule.

2.2.2 Production Rules

A production rule is a basic component of an expert system and defined as

if *condition-part*, then *action-part*,

where the condition-part consists of multiple conditions. Here, we denote a production rule as $w \Leftarrow w_1, w_2, \cdots, w_n$. In this thesis, we consider each condition as the completion of its corresponding work, and an action as a newly activated work.

Differently from definite clauses, we evaluate production rules forwardly as one way information passing from condition-part to action-part as usual.

2.2.3 ECA Rules

We introduce an ECA (event-control-action) as an extension of a production rule for efficient processing. The semantics of a ECA rule is that

if an event E occurs, then a condition C is evaluated, and if C is satisfied, then an action A is executed.

Comparative with a production rule, an ECA rule has some advantages for our application, workflow:

- In workflow, it is easier to model events such as feedback and analyze of failure.
- We can classify rules according to kinds of events and optimize them.
- An event corresponds to the timing of evaluation, while conditions correspond to the contents of evaluation.

Recently ECA rules are used in the context of active databases [MD89], while we use them to classify kinds of flows among works.

2.3 Agent as an Executer

A workflow defines a set of works and their structured flows, where each work may be executed either automatically by a program, or by a person: i.e., simply speaking, the executer of work can be abstracted as a *problem solver* or an *agent*. In this thesis, we use a *problem solver* as a general term for a database system, a knowledge-base system, a constraint solver, an expert system, an application program, and a person, and we employ a concept *agent*, proposed by a heterogeneous distributed cooperative problem solver, *Helios*[YA94, AYT95], as an abstracted problem solver with the same protocol.

A basic concept (in *Helios*) is an *agent*, defined as follows:

$$\begin{array}{lcl} \text{agent} & := & (\text{capsule, problem-solver}) \\ & | & (\text{capsule, environment, } \{\text{agent}_1, \cdots, \text{agent}_n\}) \end{array}$$

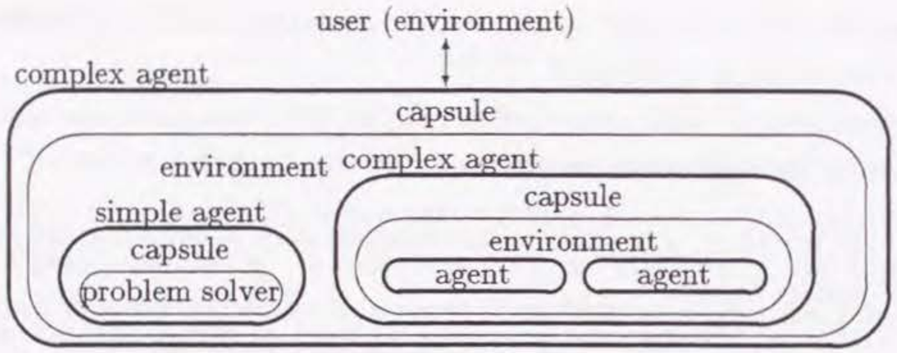

Figure 2.2: Basic Model of Agent

A *simple* agent is defined as a pair of a *capsule* and a problem solver: intuitively, a problem solver is wrapped with a capsule as in Figure 2.2. A *complex* agent is defined as a triple of a capsule, an *environment*, and a set of agents ($\text{agent}_1, \cdots, \text{agent}_n$), where an environment is a field where $\text{agent}_1, \cdots, \text{agent}_n$ can exist and communicate with each other. Intuitively, as a pair of an environment and a set of agents can be considered also as a problem solver, a new agent can be defined by wrapping them by a capsule. That is, an agent can be also hierarchically organized. Figure 2.2 shows such structures.

A capsule and an environment are defined as follows:

capsule	:=	(agent-name, methods, self-model,
		translation-rules, negotiation-strategy)
environment	:=	(agent-names, common-type-system,
		negotiation-protocol, ontology)

An agent name in a capsule is an identifier of the corresponding agent, and *agent names* in an environment specify what agents exist in the environment. *Methods* in a capsule define *import* and *export* method protocols of the corresponding agent. An agent with only import methods is called *passive* and an agent with both methods is called *active*: that is, only an agent which sends new messages through export methods can negotiate with other agents. A *common type system* in an environment enforces all agents under the environment to type all messages strongly. A *self model* in a capsule defines what the agent can do. An environment extracts necessary information from self models in agents to dispatch messages among agents. Under a *negotiation protocol* in an environment, each agent defines a *negotiation strategy* to communicate with other agents. An *ontology* defines the transformation of the contents of messages among agents, while a capsule converts the syntax and type of messages between the common type system and the intrinsic type system of the corresponding problem solver. These information is defined in *CAPL* (CAPsule Language) and *ENVL* (ENVironment Language).

Although various information is defined in each environment and each agent, a *message* among agents is in the form of a global *communication protocol* consisting of the message identifier, the identifier of a sender agent, the identifier of a receiver agent, a transaction identifier, and a message. A message identifier is common in a query message and answer messages. A transaction identifier is used to identify a negotiation process as a transaction, which can be nested.

A user can play three roles in *Helios*: an end user, an outermost environment, and a problem solver if he keep the above same protocol.

For communication between a user and an agent, a user can give his user model, which corresponds to a common type system and data structures defined in an outermost capsule. Given a user model to an agent, its capsule transforms all messages between the user and the agent.

A user is defined as the outermost environment where there is only one (simple or complex) agent. If an internal agent cannot solve a problem,

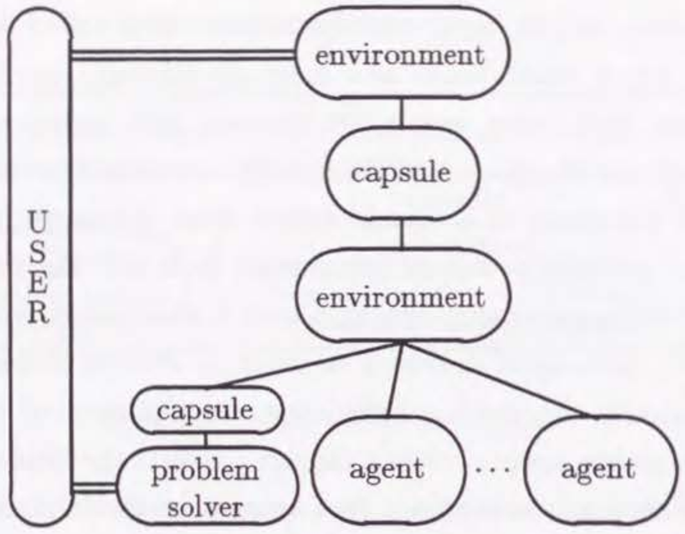

Figure 2.3: User as an Environment and an Agent

the problem is thrown out in the outer environment. Hence, a user receives unsolvable problems finally. If the user returns the answer to the agent, the agent continues to process the suspended message.

Furthermore, a user may be defined also as an agent, that is, a user can process a message sent by its capsule and return its result to the capsule. This feature helps not only prototyping a system, but also constructing a groupware environment, if multiple users are defined as agents. In this thesis, we use such features of an agent as an executer of a work. Such models make prototyping multi-agent programming in our model easier.

Relations among users and agents are shown in Figure 2.3.

Chapter 3

A Workflow Model with Database Technologies

In this chapter a formal workflow data model is proposed. This model represents a workflow as a *workflow template* (WFT), a set of *activity objects.* Each activity object corresponds to each office work units of the workflow. An activity object is regarded as a transformation function of *message objects*, executed by an agent or by sub activity objects as subroutines. Two kinds of flows, *horizontal flows* and *vertical flows*, are derived from input-output relationships and part-of relationships between activity objects, respectively. An execution model of a WFT is defined as a production system which treats horizontal flows and vertical flows as production rules and definite clauses, respectively.

We also propose an instantiation mechanism of workflows in a formal way. A partial ordering of WFTs is defined based on Smyth orderings of its components. Instantiation of a WFT is treated as an assignment over this ordering.

Based on these formal models, a workflow database, *workflow base*, is formerly defined. We also discuss about its integrity constraints.

3.1 Workflow Model

An *activity object*, which corresponds to a unit of work, is recursively defined as follows:

$$a = (I, O, P, S) \text{ where } \begin{aligned} I &= \{i_1, \cdots, i_n\} \ (n \geq 1) \\ O &= \{o_1, \cdots, o_m\} \ (m \geq 1) \\ P &= \text{string} \\ S &= \text{WFT} \end{aligned}$$

where I is a set of inputs of a, O is a set of outputs of a, P is an agent who is responsible for the execution of a, and S is a WFT (defined in the following) of a, which executes subworks of a. Intuitively, a receives I, P executes its necessary work, and a sends O. During the execution, if necessary, a divides I, dispatches them to S, monitors their execution processes, and composes O from their results.

Strictly speaking, an activity object is defined as a quintet (a, I, O, P, S), whose identifier is a. In this paper, for simplicity, we denote simply a or $a = (I, O, P, S)$. Further, when I, O, and P are singletons, $\{\}$ is omitted, if there is no misunderstanding.

A *workflow template* (WFT) W is defined as a set, $\{a_1, \cdots, a_n\}$, of activity objects $a_1, \cdots, a_n$ $(n \geq 0)$. Exactly, it is $(W, \{a_1, \cdots, a_n\})$, the identifier of which is W. For simplicity, we denote $W = \{a_1, \cdots, a_n\}$, as in an activity object.

Now we can define workflows. There are two kinds of flows in a WFT as follows:

1. *horizontal flow* "$\Longrightarrow$"

$$a_1 \Longrightarrow a_2 \overset{def}{=} (O_1 \supseteq I_2)$$
$$\{a_1, a_2, \cdots, a_n\} \Longrightarrow a \overset{def}{=} (\sum_{i=1}^{n} O_i \supseteq I) \wedge \forall j \neg(\bigcup_{i=1}^{n} O_i - O_j \supseteq I)$$

The second definition specifies the minimality of inputs.

2. *vertical flow* "$\longrightarrow$"

$$a \longrightarrow a_i \overset{def}{=} a_i \in S \text{ where } a = (I, O, P, S)$$

If $a_1 \longrightarrow a_2$, a_2 is called a *child* of a_1 and a_1 is a *parent* of a_2.

Consider an example. There are four activity objects, a_1, a_2, a_3, and a, such that $O_1 = \{o_1, o_2\}, O_2 = \{o_2, o_3\}, O_3 = \{o_3, o_1\}$, $I = \{o_1, o_2, o_3\}$. The possible workflows are defined as follows:

$$\{a_1, a_2\} \Longrightarrow a, \ \{a_2, a_3\} \Longrightarrow a, \ \{a_3, a_1\} \Longrightarrow a$$

On the other hand, $\{a_1, a_2, a_3\} \Longrightarrow a$ is not a workflow because it violates the minimality condition.

To define various classes of workflows from a WFT, we define several restrictions:

1. *closed* WFT:
   Consider any activity object $a = (I, O, P, S)$ in a WFT W. If $a_i \in W$ for any $a_i \in S$, then W is called *closed*.

2. *acycle* WFT:
   We define transitive closures of $\Longrightarrow$ and $\longrightarrow$:

   (a) $a_1 \overset{+}{\Longrightarrow} a_2 \overset{def}{=} a_1 \Longrightarrow a_2 \vee ((a_1 \overset{+}{\Longrightarrow} a_1') \wedge a_1' \Longrightarrow a_2)$

(b) $a_1 \xrightarrow{+} a_2 \overset{def}{=} a_1 \longrightarrow a_2 \vee ((a_1 \xrightarrow{+} a_1') \wedge a_1' \longrightarrow a_2)$

If $\forall a \in W.\ \neg(a \overset{+}{\Longrightarrow} a)$, then W is called *acyclic* in $\Longrightarrow$, and if $\forall a \in W.\ \neg(a \xrightarrow{+} a)$, then W is called *acyclic* in $\longrightarrow$. Note that a cyclic workflow corresponds to a feedback of a work.

3. *redundant* WFT:
   For any $a = (I, O, P, S)$, if $O \subseteq I$, then a is called *redundant*. A WFT containing redundant activity objects is called a *redundant* WFT. Note that, even if two activity objects $a_1 = (I_1, O_1, P_1, S_1)$ and $a_2 = (I_2, O_2, P_2, S_2)$ have relations of $I_1 \subseteq I_2 \wedge O_2 \subseteq O_1$, we consider they are not redundant if $P_1 \neq P_2$.

4. *triangle* of WFT:
   Let $\rightsquigarrow = \overset{+}{\Longrightarrow} \cup \xrightarrow{+}$. Then if $\exists a.(a_1 \Longrightarrow a_2 \vee a_1 \longrightarrow a_2) \wedge a_1 \rightsquigarrow a \wedge a \rightsquigarrow a_2$, then there exists two ways between a_1 and a_2. A WFT containing a_1 and a_2 with such a relation is called *triangle*.

Workflow is defined as a WFT W satisfying these restrictions:

$$\forall\, a_1 \in W, \exists\, a_2 \in W.\ a_1 \rightsquigarrow a_2 \vee a_2 \rightsquigarrow a_1,$$

and W constitutes a connected graph.

This definition is easily extended into a set of workflows. Consider a set, S, of workflows, each of which is connected to another workflow in S: that is,

$$\forall w_1 \in S, \exists w_2 \in S.$$
$$((w_1 = a \rightsquigarrow a' \wedge w_2 = a' \rightsquigarrow a'') \vee (w_2 = a \rightsquigarrow a' \wedge w_1 = a' \rightsquigarrow a'')),$$

and S constitutes a single graph. We extend the above definition and call such a set of workflows a workflow generally. The above definitions,

closedness, acyclicity, and redundancy, are also considered in this definition of a workflow. In the case of triangle restriction, it is extended as follows: $S \Longrightarrow a \wedge \exists S' \subseteq S.\ S' \rightsquigarrow a' \wedge a' \rightsquigarrow a$. Remark that we can consider a set of workflows also as a set of activity objects because flows are represented only implicitly.

Consider two workflows, S_1 and S_2. Ordering between S_1 and S_2 is defined by subset relation: i.e., $S_1 \subseteq S_2$. In this order, a maximal set in a WFT called a *maximal* workflow. In a maximal workflow, an activity object without any parents is simply called a *parent*. Generally, a WFT defines multiple workflows in the sense of this definition.

A closed WFT guarantees at least one definition of a closed workflow, which corresponds to the unit of a complete work: On the other hand, an unclosed workflow includes a definition of a work, lacking some activity objects to which a work might be submitted.

3.2 Execution Model

To execute a WFT for a work, we must define its execution model. The execution model of a WFT consists of two models, *P-box* and *C-box*, each of which corresponds to horizontal and vertical flows, respectively.

A horizontal flow defines the following production rules:

$$\begin{aligned}
\text{If } a_1 \Longrightarrow a_2, &\quad \text{then } a_2 \Leftarrow a_1 \\
\text{If } a_1 \Longrightarrow a_3 \text{ and } a_2 \Longrightarrow a_3, &\quad \text{then } a_3 \Leftarrow a_1, a_2 \\
\text{If } a_1 \Longrightarrow a_2 \text{ and } a_1 \Longrightarrow a_3, &\quad \text{then } a_2 \Leftarrow a_1 \text{ and } a_3 \Leftarrow a_1 \\
\text{If } a_1, &\quad \text{then } a_1 \Leftarrow .
\end{aligned}$$

The right hand side of $\Leftarrow$ is a set of conditions, while the left hand side is an action. This rule is evaluated forwardly as in ordinary production rules. That is, by receiving all end conditions of the corresponding activity objects, the corresponding activity object to the action is activated. An activity object itself is represented as a fact.

A production system, as a set of such production rules, is stored in **P-box** in the WFT.

On the other hand, a vertical flow represents a set of definite clauses as in logic programming such as Prolog. That is, when an activity object a generates children $a_1, a_2, \cdots, a_n$, the execution model is represented as follows:

$$a \leftarrow a_1, a_2, \cdots, a_n$$

Intuitively, constraints and binding information are propagated from a to $a_1, a_2, \cdots, a_n$, and if all children objects end successfully, then a ends its execution successfully. That is, these rules are evaluated backwardly as in Prolog.

Such a set of definite clauses is stored in a **C-box** in the WFT. As dynamically generated child objects are also activity objects, they are stored in P-box, not in C-box.

Consider an example (Figure 3.1). Such a process is defined as a WFT in Figure 3.2. In a real review process, we must instantiate "submitted-paper", "chair", "secretary", "PC-member", and so on. We will introduce the instantiation concept of workflow in the next section. Remark that multiple instances of a child object, "review", must be dynamically generated, after "chair" received "all-submitted-papers".

The corresponding execution model is shown in Figure 3.3.

First, three production rules, "receive", "dispatch-1", and "send $\Leftarrow$ dispatch-1" are generated in P-box. After activating "dispatch-1", n activity objects, "dispatch-2_1","dispatch-2_2", $\cdots$, "dispatch-2_n" are generated as instances of "dispatch-2". Such activity objects are gener-

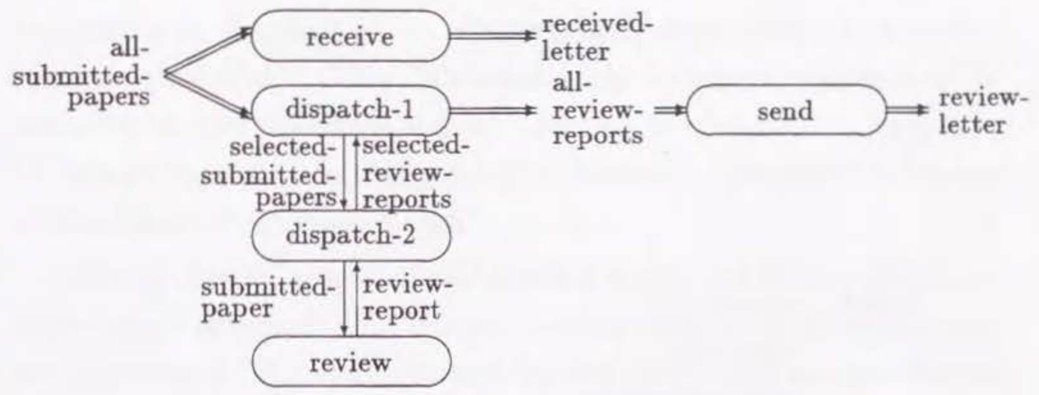

Figure 3.1: Review Process

review-process	=	{receive, dispatch-1, dispatch-2, review, send}
receive	=	(all-submitted-papers, received-letter, secretary, {})
dispatch-1	=	(all-submitted-papers, all-review-reports, chair, {dispatch-2})
dispatch-2	=	(selected-submitted-papers, selected-review-reports, PC-member, {review})
review	=	(submitted-paper, review-report, reviewer, {})
send	=	(all-review-reports, review-letter, secretary, {})

Figure 3.2: WFT of a Review Process

$$\begin{array}{ll}
\text{P-box:} & \text{receive}\\
 & \text{dispatch-1}\\
 & \text{send}\Leftarrow\text{dispatch-1}\\
 & \left.\begin{array}{l}\text{dispatch-}2_1\\ \text{dispatch-}2_2\\ \vdots\\ \text{dispatch-}2_n\\ \vdots\end{array}\right\} \begin{array}{l}\text{Dynamically generated}\\ \text{activity objects}\end{array}\\
\text{C-box:} & \text{dispatch-1} \leftarrow \text{dispatch-}2_1\text{,dispatch-}2_2,\cdots\text{,dispatch-}2_n.\\
 & \text{dispatch-}2_1 \leftarrow \text{review}_{11}\text{,review}_{12},\cdots\text{,review}_{1l}\\
 & \text{dispatch-}2_2 \leftarrow \text{review}_{21}\text{,review}_{12},\cdots\text{,review}_{2m}\\
 & \vdots\\
 & \text{dispatch-}2_n \leftarrow \text{review}_{n1}\text{,review}_{n2},\cdots\text{,review}_{nk}
\end{array}$$

Figure 3.3: Execution Model of a Review Process

ated in P-box and its corresponding rule (definite clause), "dispatch-1 $\leftarrow$ dispatch-2_1, dispatch-2_2, $\cdots$, dispatch-2_n", which controls their execution, is generated in C-box. After each activity object "dispatch-2_i" is activated, its corresponding activity objects, "review$_{i1}$",$\cdots$, "review$_{ir}$", for "review" and its rule, "dispatch-2_i $\leftarrow$review$_{i1}$,$\cdots$,review$_{ir}$", is stored in P-box and C-box, respectively.

Activity objects, generated and inserted into P-box during execution, do not cause any conflict to existing activity objects, because they are newly generated. Therefore, dynamic update of P-box is an conservative extension and does not change its semantics and does not cause new conflicts.

3.3 Workflow Instance

To apply WFTs defined in the previous sections to real works, we must instantiate inputs, outputs, agents, and so on. Such instantiated WFTs are called *workflow instances* (WFI). Although a WFI is an instance of a WFT, they are essentially the same. Here we use a WFI as a WFT to execute a real work.

First we define ordering between objects and, as the results, ordering between WFTs. Consider a domain $\mathcal{M}$ of message objects consisting of inputs and outputs, and a domain $\mathcal{P}$ of agents. An activity object $a = (I, O, P, S)$ is basically a function from I to O, defined as follows:

1. $I \in 2^{\mathcal{M}}$, $O \in 2^{\mathcal{M}}$
2. $P \in 2^{\mathcal{P}}$
3. $a \in \mathcal{A}$, $S \subseteq \mathcal{A}$, where $\mathcal{A}$ is a set of activity objects, defined as

follows:

$$\mathcal{A} \;:=\; 2^{\mathcal{M}} \times 2^{\mathcal{M}} \times 2^{\mathcal{P}} \times 2^{\mathcal{A}}$$

Although P can be defined as a function on $\mathcal{M}$, we take another domain $\mathcal{P}$, because we consider that two activity objects with the same inputs and the same outputs, but the different agents should be defined as different objects.

For an activity object $a = (I, O, P, S)$, we consider assignment θ of I to $I' = \{i_1, i_2, \cdots, i_n\}$, O to $O' = \{o_1, o_2, \cdots, o_m\}$, and P to $P' = \{p_1, p_2, \cdots, p_k\}$. $\mathcal{M}, \mathcal{P}$ are assumed to be partially ordered sets by $\sqsubseteq_{\mathcal{M}}$ and $\sqsubseteq_{\mathcal{P}}$, respectively. We usually omit the subscripts of $\sqsubseteq$, for simplicity. The assignment is denoted as follows:

$$\theta = \{I/\{i_1, \cdots, i_n\}, O/\{o_1, \cdots, o_m\}, P/\{p_1, \cdots, p_k\}\},$$

which corresponds to specialization with the following relations:

$$\begin{aligned} I' \sqsubseteq_S I &\Leftrightarrow \forall i \in I, \exists i' \in I'. i' \sqsubseteq i \\ O' \sqsubseteq_S O &\Leftrightarrow \forall o \in O, \exists o' \in O'. o' \sqsubseteq o \\ P' \sqsubseteq_S P &\Leftrightarrow \forall p \in P, \exists p' \in P'. p' \sqsubseteq p \end{aligned}$$

That is, $\sqsubseteq_S$ is Smyth ordering, which is necessary and sufficient condition of being assignment.

The orderings between two activity objects $a = (I_1, O_1, P_1, S_1)$ and $a' = (I_2, O_2, P_2, S_2)$ is defined as follows:

$$\begin{aligned} a \sqsubseteq a' \;\stackrel{def}{=}\; & I_1 \sqsubseteq_S I_2 \wedge O_1 \sqsubseteq_S O_2 \\ & \wedge P_1 \sqsubseteq_S P_2 \wedge S_1 \sqsubseteq_S S_2. \end{aligned}$$

where the orderings of S_1 and S_2 is defined as:

$$S_1 = \{a_1, \cdots, a_n\} \sqsubseteq_S S_2 = S_1\theta \stackrel{def}{=} \{a_1\theta, \cdots, a_n\theta\}.$$

A partial assignment into S is also defined as follows:

$$a' \sqsubseteq a \;\Leftrightarrow\; \{a'\} \cup S \sqsubseteq_S \{a\} \cup S$$

An assignment θ to a WFT W is defined as $W\theta = \{a_1\theta, \cdots, a_m\theta\}$. As WFTs and WFIs are the same, we can define ordering among them by using assignment.

Consider the previous example (Figure 3.1). To instantiate "review-process" into "CODAS-review", we define the following assignment:

all-submitted-papers/{paper$_1$,paper$_2$,$\cdots$,paper$_n$}
chair/Kambayashi
PC-members/{Masunaga, Uemura, Makinouchi, Tanaka, $\cdots$}
secretary/{Takada}
$\vdots$

Using this assignment, a WFT "review-process" is instantiated into "CODAS-review". Such instantiation can be denoted as follows:

review-process/CODAS-review

We can activate a WFI by such instantiation.

3.4 Workflow Base

Various information defined in the previous sections are stored in a workflow database, that is, *workflow base* (WFB). For simplicity, we assume that identifiers of WFT/WFI, activity objects, message objects and agents are global in a workflow base.

A workflow base is defined by a set of WFTs (including WFIs) and $(\mathcal{M}, \sqsubseteq_{\mathcal{M}})$, $(\mathcal{P}, \sqsubseteq_{\mathcal{P}})$. Now we generalize the definitions of WFTs in Section 3.1 to include the identifier of the upper WFT and assignment. That is, each WFT W $(= W'\theta)$ is defined as follows:

$$\begin{array}{ll} \text{definition:} & W = (W',\, \theta,\, \{a_1, \cdots, a_m\}) \\ & a_1 = (I_1,\, O_1,\, P_1,\, S_1) \\ & \vdots \\ & a_n = (I_n,\, O_n,\, P_n,\, S_n) \\ \text{execution model:} & \text{P-box: } \cdots \\ & \text{C-box: } \cdots \end{array}$$

Note that the execution model can be generated from the definition initially, however dynamically generated activity objects and rules are not stored initially.

As for a workflow base, there is an integrity constraint:

- Specialization hierarchy of WFTs: Ordering among WFTs are consistently defined by $(\mathcal{M}, \sqsubseteq_{\mathcal{M}})$ and $(\mathcal{P}, \sqsubseteq_{\mathcal{P}})$.

Furthermore, we can impose various integrity constraints defined in Section 3.1, according to applications' requirements.

3.5 Summary

In this chapter a formal workflow data model, *workflow base* was proposed. This model represents a workflow as a *workflow template* (WFT), a set of *activity objects*. Each activity object corresponds to each office work units of the workflow, executed by an agent or by sub activity objects as subroutines.

Two kinds of flows, *horizontal flows* and *vertical flows*, are derived from input-output relationships and part-of relationships between activity objects, respectively. An execution model of a WFT is defined as a production system which treats horizontal flows and vertical flows as production rules and definite clauses, respectively. This execution model

deals with both static flow control and dynamic dispatching of subworks in the same matter.

We also proposed an instantiation mechanism of workflows in a formal way. A partial ordering of WFTs is defined based on Smyth orderings of its components. Instantiation of a WFT is treated as an assignment over this ordering.

Based on these formal models, a workflow database, *workflow base*, was formerly defined. We also discussed about its integrity constraints.

Chapter 4

Extensions on Workflow Base

In Chapter 3, a workflow is modeled as a set of activity objects, each of which consists from its inputs and outputs, the responsible agents, and subworks. This simple modeling is useful to give formal and powerful frameworks into WFMSs.

On the other hand, real office works are more complex than this modeling. First they might have feedback loops in most cases; that is, when the results of some works in workflows do not satisfied the previously defined requirements, the works or a sequence of the works will be redone until the requirements are satisfied. However, the workflow model in Chapter 3 cannot support feedback flows.

Second weakness of the workflow model in Chapter 3 is about the constraints of office works. There are several constraints about the resources of the organizations, such as deadlines, funds, materials, persons, etc. These constraints affect the workflow structures and the instantiation process of workflows. These affections are occurred dynamically even in the execution phase of workflows. Moreover, the constraints interfere each other. For example, the agent responsible with some work is also responsible with its subworks whose responsible agents are not defined.

In this Chapter, we discuss some extensions to workflow base to deal with more complex office works. First we show the way to realize loopback flows into workflow base, using ECA rules, in Section 4.1. In Section 4.2 and Section 4.3 extensions for dealing with *time constraints* and *resource constraints* are discussed, respectively. In Section 4.2, after introducing *time plugin* for extending activity objects with time attributes, we discuss interferences between time constraints and flows in workflow base, scheduling problem on workflow base, and time adjustment. In Section 4.3, two resource constraints, resource sum equality and resource sum inequality are introduced, and the way for resource reallocation which is done when the resource constraints are violated is discussed. And finally, we show in Section 4.4 dynamic horizontal flows caused by exclusive lock mechanism.

4.1 Loopback Flows

In order to deal with feedback flows of workflows, we must provide conditional flow control mechanisms. Some conditions for occurring feedback are evaluated at the source node of feedback flows, and feedback is occurred when and only when the conditions are satisfied. If the conditions are not satisfied, feedback is not occurred and ordinary flows are activated. In this sense, flow branch mechanism should be provided at the source node.

We use *Event-Condition-Action (ECA) rules* [MD89] to realize feedback flows in WFB. ECA rule is a basic concept of *active database systems*, DBMSs that allows users to specify actions to be taken automat-

ically when certain conditions arise. An ECA rule has three attributes: *event*, *condition*, and *action*. Intuitively, semantics of the rule is simple: when the event occurs, evaluate the condition; and if the condition is satisfied, execute the action.

For two activity objects $a_1, a_2 \in W.a_1 \overset{+}{\Longrightarrow} a_2$ where W is a WFT, a *feedback horizontal flow* from a_2 to a_1 is defined as a special horizontal flow such as:

$$E, a_2 \Longrightarrow a_1$$

where E is a set of *feedback events*. Intuitively, this flow means that the outputs of a_2 are sent to a_1 when and only when all the feedback events in E is occurred. Similarly, for activity objects $a_1, \cdots a_n, a \in W.\{a_1, \cdots, a_n\} \overset{+}{\Longrightarrow} a$, a feedback horizontal flow from a to $\{a_1, \cdots, a_n\}$ is defined as:

$$E, a \Longrightarrow \{a_1, \cdots, a_n\}.$$

Note that we do not deal with feedback vertical flows in this thesis. From the definitions in Chapter 3, vertical flows are defined from the inclusion relationships among activity objects. Hence feedback vertical flows, i.e. loop of the partial orderings defined on inclusion relationships, are beyond from the conventional set theory.

The semantics of feedback horizontal flow is defined using ECA rules:

$$\text{If } E, a_2 \Longrightarrow a_1, \quad \text{then } a_1 \Leftarrow E(a_2).$$
$$\text{If } E, a \Longrightarrow \{a_1, \cdots, a_n\}, \quad \text{then } a_1 \Leftarrow E(a), \cdots, a_n \Leftarrow E(a).$$

The left hand side of $\Leftarrow$ is an action; while the right hand side of $\Leftarrow$ includes a set of events as well as a set of conditions. $E(a)$ means a set of events, with message objects passed into the sink activity object of the feedback horizontal flow. When the event set E is occurred, the rules that include E in its right hand side are activated, then the activity object in its left hand side is activated. Remark that any production rules without event part is not activated in this case. This means that any ordinary horizontal flows are activated when a feedback horizontal flows is activated, and vice versa.

The rules corresponding with feedback horizontal flows are stored in P-box of the WFT, as same as ordinary horizontal flows. No extension is needed into the semantics of the production system of P-box.

In order to implement facilities to evaluate some conditions when E is occurred, it is very simple: just put conditions $c_1, \cdots, c_m$ into the left hand side of a feedback horizontal flows, as follows:

$$E, c_1, \cdots, c_m, a_2 \Longrightarrow a_1.$$

This flow defines the ECA rule

$$a_1 \Leftarrow E(a_2), c_1, \cdots, c_n.$$

Intuitively the semantics of this rule corresponds to that of the ordinary ECA rules: When the events E are occurred, the conditions $c_1, \cdots, c_n$ are evaluated; all conditions are evaluated as true, then the activity object a_1 is activated. This extension enables multiple branch of flows, as follows:

$$E, c_1, a_2 \Longrightarrow a$$
$$E, c_2, a_2 \Longrightarrow a'$$

The first flow occurs when c_1 is true, while the second flow occurs when c_2 is true. That is, flows will branch by the evaluation results of the conditions c_1 and c_2.

4.2 Time Constraints

In general, several constraints about time can be considered on workflows such as:

- *startline constraints.* When the process can be started from.
- *deadline constraints.* When the process must be finished by.
- *duration constraints.* Constraints about time duration among processes.
- *prediction constraints.* Constraints about estimation time of the process.

We show some examples of each constraint:

- startline constraints.

  Consider again the workflow example in Figure 3.1. Review report format is distributed to the reviewers by ftp, but the preparation of ftp service was not in time for dispatch of submitted papers. In such a case, the reviewers must wait for writing review reports until ftp service is available.

  In this example, startline constraint plays an role of implicit flows. The workflow in Figure 3.1 does not explicitly describe review report sheets as message objects through vertical flows from "dispatch-1" to "dispatch-2", or from "dispatch-2" to "review".

- deadline constraints.

  In Figure 3.1, as deadline of the process "send" is fixed at the stage of call-for-papers, each unit of work in the workflow has a deadline determining backwardly from the deadline of "send".

- duration constraints.

  "Receive" process in Figure 3.1 must be started at least one day after, for example, from receiving a submitted paper.

In some cases, processes in workflow must be executed simultaneously. For example, "send-review-letters" process in Figure 3.1 must be started at the same time for all authors of the submitted papers. This kind of simultaneity can be considered as a duration constraint of no duration.

- prediction constraints.

  When a reviewer receives submitted papers, he estimates the time to be required for the reviews, and then decides whether to undertake reviewing. If he decides to undertake reviewing but he is too busy to review all papers by himself, he may redistribute the papers to those who are working under him. In such a case, he estimates how long each staff takes to review a paper.

In order to deal with these constraints, we propose *time constraints* on workflow-base.

4.2.1 Time Plugin

First, *time plugin* for an activity object $plug_T$ is defined as follows:

$$plug_T \overset{def}{=} (s, e, d, p)$$

where s is the time a must be started; e is the time a must be finished by; d is the duration period which a must be done; p is the prediction period of a. An activity object can be *plugged in* with time plugin $plug_T$ such as:

$$a_T = (I, O, P, S, s, e, d, p)$$

where $a = (I, O, P, S)$ is an activity object. A WFT consisting from activity objects with time plugin is called a *WFT with time plugin*.

To execute a_T successfully, a_T must satisfy these two inequalities:

1. $e - s \geq d$ (*duration inequality*). Because a_T must be executed in the time period d, a_T cannot satisfy either s or e if a_T does not satisfy this inequality.

2. $d \geq p$ (*estimation inequality*). As a_T must be executed in the time period d, the estimate time of a_T must be less than d.

To apply WFTs with time plugin to real works, they are instantiated in the similar way with instantiation of WFTs without time plugin in Section 3.3. Consider a domain $\mathcal{T}$ of time and a domain $\mathcal{D}$ of time period, with partial orderings $\sqsubseteq_{\mathcal{T}}, \sqsubseteq_{\mathcal{D}}$, respectively. For an activity object with time plugin $a_T = (I, O, P, S, s, e, d, p)$, we consider assignment θ of s to s', e to e', d to d', and p to p', where $s' \sqsubseteq_{\mathcal{T}} s, e' \sqsubseteq_{\mathcal{T}} e, d' \sqsubseteq_{\mathcal{D}} d, p' \sqsubseteq_{\mathcal{D}} p$, respectively, in addition to the ordinary assignment into activity objects. Instantiation of a_T is defined by an assignment θ to a_T, denoted as $a_T\theta$. An assignment θ to a WFT with time plugin W_T is defined as $W_T\theta = \{a_{T1}\theta, \cdots, a_{Tn}\theta\}$.

If no assignment of s is included in θ, it is assumed that an assignment s/c is omitted, where c is the current time on the agent P'.

An assignment θ including either e/e' or d/d' is called a *correct assignment on time plugin*. In a correct assignment on time plugin,

- it is assumed that an assignment $e/s' + d'$ is omitted if e/e' is not included;

- it is assumed that an assignment $d/e' - s'$ is omitted if d/d' is not included.

If p/p' is not included in a correct assignment on time plugin, it is assumed that p/d' is omitted.

4.2.2 Time Constraints over Flows

Time constraints of activity objects which connect by horizontal or vertical flows interfere each other. In this section, relationships between time constraints and the flows in workflow base are discussed.

First consider horizontal flows. For activity objects with time plugin a_1 and a_2 that have a horizontal flow $a_1 \Longrightarrow a_2$, these inequalities must be satisfied:

1. $s_1 + p_1 \leq s_2$
2. $e_1 + p_2 \leq e_2$
3. $e_1 \leq s_2$

Similarly, for a horizontal flow $\{a_1, \cdots, a_n\} \Longrightarrow a$, these inequalities must be satisfied:

1. $\max(s_1 + p_1, \cdots, s_n + p_n) \leq s_2$
2. $\max(e_1 + p, \cdots, e_n + p) \leq e$
3. $\max(e_1, \cdots, e_n) \leq s$

These are obvious from the definition of horizontal flows.

For a vertical flow $a \longrightarrow a_i$, these inequalities must be satisfied:

1. $s < s_i$
2. $e > e_i$
3. $d > d_i$
4. $p > p_i$

Note that p may not be larger than $\Sigma_{i=1}^{n} p_i$ because some of the sub activity objects may be executed concurrently.

4.2.3 Scheduling

An agent generally has a number of to-do works at the same time. In such a case, the agent must do scheduling among the works to put to-do priorities to them. In this section, we discuss scheduling problems using the time constraints defined in the previous section.

To simplify the discussions, we first consider scheduling between two activity objects. Let

$$\begin{aligned} a_1 &= (I_1, O_1, P, \emptyset, s_1, e_1, d_1, p_1) \\ a_2 &= (I_2, O_2, P, \emptyset, s_2, e_2, d_2, p_2) \end{aligned}$$

be activity objects with time plugin. These two activity objects are both responsible by an agent P because both have empty set of sub activity objects. From the time constraints point of view, we can categorize relationships between a_1 and a_2 into two cases: $(s_1 \leq s_2) \wedge (e_1 \leq e_2)$ and $(s_1 \leq s_2) \wedge (e_2 \leq e_1)$. Scheduling of a_1 and a_2 is as follows:

- In case of $(s_1 \leq s_2) \wedge (e_1 \leq e_2)$,

$$\begin{array}{ll} p_1, p_2 & \text{if } s_1 + p_1 + p_2 \leq e_2 \\ \text{fail} & \text{if } s_1 + p_1 + p_2 > e_2 \end{array}$$

- In case of $(s_1 < s_2) \wedge (e_2 < e_1)$,

$$\begin{array}{ll} p_{11}, p_2, p_{12} & \text{if } e_2 \leq s_1 + p_1 + p_2 \leq e_1 \\ & \text{and if } p_1 \text{ can be divided into subworks} \\ p_1, p_2 & \text{if } s_1 + p_1 + p_2 \leq e_2 \\ \text{fail} & \text{if } s_1 + p_1 + p_2 > e_1 \end{array}$$

"fail" means that scheduling of these two activity objects is failed. In this case, the agent P must negotiate with another agents to make the time constraints of the activity objects more loose. We do not discuss the negotiation process in this thesis.

Scheduling problem among more than three activity objects is much more difficult than that of two activity objects. In actual, the problem is treated as an linear programming problem. The detailed discussions from this point of view is beyond this thesis.

4.2.4 Time Adjustment

In the discussions above, we assume that all agents share one global clock. However, this assumption is too strict under distributed environment: each agent in general have the different clock, and there are time lags among them. In this section, we extend the time constraints on workflow base by loosening this assumption.

First we extend time plugin as follows:

$$plug_T \overset{def}{=} (c, s, e, d, p)$$

and an activity object with time plugin a_T is extended as:

$$a_T \overset{def}{=} (I, O, P, S, c, s, e, d, p)$$

where c is the current time of the clock which P has.

Consider a situation that the agent P dispatches a_T to the agent P'. In this case, P' adjusts time plugin of a_T as follows:

$$s := s - a - t;\; e := e - a - t;$$

where $a = c - c'$; t is the transmission time from P to P'.

4.3 Resource Constraints

In this section, we give a workflow of publishing process as another example of workflow which have more severe constraints about resources than the reviewing process.

Figure 4.1 shows an illustrative example of publishing process of magazine[1]. The process is defined as a sequence of two units: producing posters and producing magazine. Each unit consists from some subprocesses. In "planning" process, an outline of the printing matter such as its design, funds, total pages and contents in case of magazine, etc., is determined. Then the requests of producing contents are sent to authors. In case of producing magazine, all the contents are gathered for editing. Finally, camera readies are sent to the printing house for printing. We assume that posters and magazines use the same logo; hence a horizontal flow from "poster printing" to "magazine printing" is put to this workflow because the logo is first designed in "poster printing" process, then sent to "magazine printing" process.

There are many constraints about resources in the workflow in Figure 4.1. For example, the sum of the costs for two printing processes is determined beforehand; the number of total pages for the magazines is also determined in the "planning" process. Agent determines the costs of the subprocesses, or the number of pages for each articles, with satisfying the resource constraints. In the next section we formalize this kind of resource constraints about its sum.

[1]In Figure 4.1, horizontal flows and vertical flows are represented as thick arcs and thin arcs, respectively.

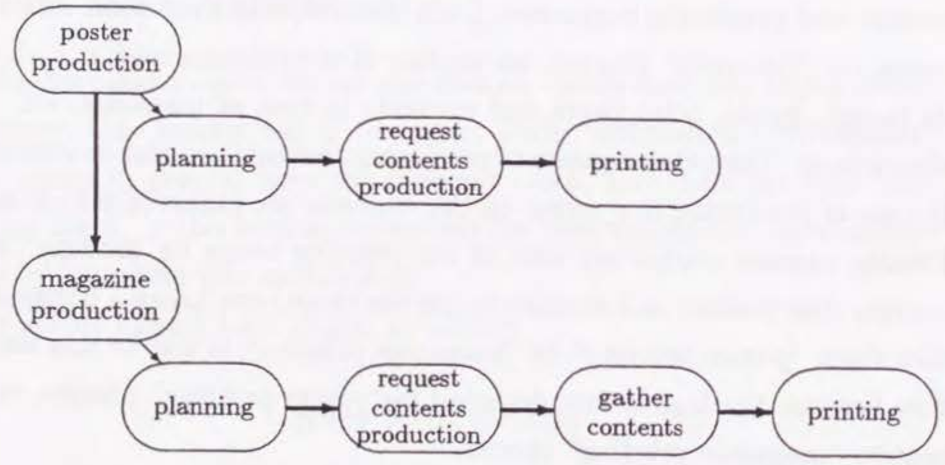

Figure 4.1: Workflow Example: Publishing Process

4.3.1 Resource Sum Constraints

Let $amount_R$ allocation amount of a resource R. *Resource plugin* $plug_R$ on a resource R is defined as

$$plug_R \overset{def}{=} (amount_R).$$

That is, $plug_R$ is a tuple with only one attribute. For an activity object $a = (I, O, P, S)$, an *activity object with resource plugin* a_R is defined as

$$a_R \overset{def}{=} (I, O, P, S, amount_R).$$

For a WFT $W = \{a_1, \cdots, a_n\}$, a *WFT with resource plugin* W_R is defined as $W_R = \{a_{1R}, \cdots, a_{nR}\}$.

All resource plugins and time plugin are compatible with each other. For example, an activity object with time plugin can be plugged in with a resource plugin, and vice versa.

For a WFT with resource plugin $W_R = (I, O, P, S, amount_R)$, we define the *resource sum equality* on R as:

$$\forall a \in W_R \wedge S \neq \emptyset.\ amount_R = \sum_{a' \in S} a'.amount_R$$

Intuitively, this equality means that the sum of the resources for subprocesses must be equal to the allocation amount of the parent process.

Note that some of the resources does not satisfy resource sum equality. Total costs in the workflow of Figure 4.1 is an example because some costs must be used in "poster printing" process or "magazine printing" themselves. Hence we consider another resource constraints, called *resource sum inequality*:

$$\forall a \in W_R \wedge S \neq \emptyset.\ amount_R \geq \sum_{a' \in S} a'.amount_R$$

As all resources in subprocesses are what the parent process allocated for the subprocesses, resource sum inequality must be satisfied in all resources. Resource sum equality and resource sum inequality are called generally as *resource constraints.*

4.3.2 Resource Reallocation

If one of the subprocesses can not satisfy the resource constraints determined by the agent of the parent process, the resource constraints of the parent process are also not satisfied.

See again the example in Figure 4.1. Consider a situation that an author writes his article of less pages than that of previously determined. This causes constraint violation about total pages. Hence the publisher must resolve the violation by some means: by requesting the author to write more pages, by requesting another author to write more pages, or by changing total pages of the magazine.

If an activity object with resource plugin a_R in WFT does not satisfy resource constraints, the agent of a_R sends a message "fail allocation in R" to the agents who are responsible with activity objects which a_R belongs. When an agent receives the message "fail allocation in R", it tries to reallocate resource R into sub activity objects. If the reallocation succeeds, the agent sends the reallocation results to the sub activity objects. If it fails, the agent sends "fail allocation in R" to parent activity objects again. We do not discuss about negotiations among agents during the reallocation process.

4.4 Horizontal Flows by Exclusive Locks

Under the situation that data is shared among agents in the same cooperative work, it is important to provide lock mechanisms for keeping integrities of data. Lock mechanisms previously proposed can be classified into two categories: exclusive locks and shared locks. Exclusive locks permit only one user to access locked data; on the other hand, shared locks permit a number of users to access locked data while locking. In workflow base, data sharing method is not included in its definitions, hence any lock mechanisms can be used.

When exclusive locks are used for data sharing, time constraints among activity objects are newly created. Consider an example that two activity objects a_1 and a_2 in the same WFT share a file f using exclusive lock. If a_1 uses f with exclusive lock, a_2 must wait until the lock is released. Therefore a time constraint $e_1 < s_2$ is newly created.

This phenomenon can be treated in a uniform way on workflow base. When a_1 begins to use f with exclusive lock, a special message object "release notification" is added into O of a_1 and I of a_2. When a_1 releases the lock, "release notification" is sent to the WFT from a_1, and a_2 catches it as an input. Note that a horizontal flow $a_1 \Longrightarrow a_2$ is newly created when "release notification" is added. According to the discussions in Section 4.2, a time constraint $e_1 < s_2$ is also created automatically.

4.5 Summary

In this Chapter, we discussed four extensions to workflow base. First we showed the way to realize loopback flows into workflow base, using ECA rules. Next we discussed about two kinds of constraints, time constraints and resource constraints, and about various properties among them: interferences of these constraints with flows in workflows and their solutions; applications such as scheduling. And finally, we showed that exclusive lock mechanism causes horizontal flows dynamically.

In order to apply workflow base to real cooperative works, some extensions discussed in this chapter must be needed. However, workflow base is so suitable for database technologies that extensions can be easily done using database technologies. In other words, these extensions prove high affinities of workflow base with databases.

Chapter 5

Database Operations on Workflow Base

As mentioned in Section 2.1, conventional workflow management systems have no standard data manipulation language. This causes several problems into WFMSs: the lack of interoperability among WFMSs, as pointed out in 2.1; poor functions for reorganizing workflows such as workflow reuse, view functions including the creation of to-do lists, workflow optimization in business process reengineering, etc.

Data manipulation language is what database technologies make the greatest contribution to workflow management, because data manipulation of general purpose is one of the most essential facilities in DBMSs, and therefore many technologies such as SQL are developed.

Workflow base provides a formal model of workflows, which is suitable for data manipulation of database technologies. In this chapter, we give several operations manipulating workflows on workflow base.

They are defined as an extension of ordinary relational algebra. That is, these operations get a set of WFTs or a set of activity objects from

WFTs in WFB. The operations are classified into three categories:

- Query operations.
  A *new* set of WFTs (or of activity objects) is obtained from the previously defined workflows No effects over the original WFTs (or activity objects).

- Operations handling WFTs.
  Though some effects may cause over other WFTs, they keep WFTs as workflows to be consistent.

- Operations handling activity objects directly.
  Any effects may cause over WFTs. Sometimes the effects make WFTs no more than workflows.

As some of them do not always keep the closure property of relational algebra, an operation for keeping closure property is also provided. The combination of this operation with others makes the closure property to be kept.

When using these operations, users can organize new workflows from the workflows in WFB. For example, the following functions are available using the operations:

- Views over workflows, such as to-do lists.
- Investigation of progress.
- Workflow reorganization.
- Reuse of workflows.

Note that any ordinary set operators or operations in relational algebra can be used for manipulating WFTs or activity objects. And also note that the components of an activity object are accessible using

dot notations. That is, $a.I$ means I of a. Hence the notations such as $a.I := a.I \cup \{kunishima\}$ can be available. We do not discuss these points further in this thesis.

5.1 Query Operations

5.1.1 Selection

Two selection operations on a WFB can be considered: selection for a set of activity objects, and selection for a set of WFTs.

Selection of activity objects, $\sigma_{expr}\, W$, gets a set of activity objects satisfying an AO-expression *expr* from a specified WFT W. AO-expressions are recursively defined as follows:

- $p \in P$ is an AO-expression, where $p \in \mathcal{P}$.
- $m \in I$ is an AO-expression, where $m \in \mathcal{M}$.
- $m \in O$ is an AO-expression, where $m \in \mathcal{M}$.
- ancestor(a) is an AO-expression, where $a \in W$.
- descendant(a) is an AO-expression, where $a \in W$.
- If both *exp*1 and *exp*2 are AO-expressions, *exp*1 op *exp*2 is also an AO-expression, where op is a conventional logical operator.

If W is omitted, a WFT of all activity objects in a WFB, W_{all}, is assumed as W. This assumption is all applicable on other operations. Evaluation value of an AO-expression *expr*, eval(*expr*) is defined as follows. For an activity object $a' = (I, O, P, S) \in W$,

$$\begin{aligned}
&\text{eval}(p \in P) = \texttt{true} \text{ iff } p \in P. \\
&\text{eval}(m \in I) = \texttt{true} \text{ iff } m \in I. \\
&\text{eval}(m \in O) = \texttt{true} \text{ iff } m \in O. \\
&\text{eval}(\text{ancestor}(a)) = \texttt{true} \text{ iff } a \longrightarrow a', a' \in W. \\
&\text{eval}(\text{descendant}(a)) = \texttt{true} \text{ iff } a' \longrightarrow a, a' \in W. \\
&\text{eval}(exp1 \,\text{op}\, exp2) = \text{eval}(exp1) \,\text{op}\, \text{eval}(exp2)
\end{aligned}$$

Selection of WFTs, $\sigma'_{expr} W$, on the other hand, gets a set of WFTs satisfying a WFT-expression *expr* from a specified WFT W. WFT-expressions are recursively defined as follows:

- wft(W') is a WFT-expression, where W' is a WFT.
- wfi(W') is a WFT-expression, where W' is a WFT.
- general(W') is a WFT-expression, where W' is a WFT.
- special(W') is a WFT-expression, where W' is a WFT.
- If both *exp*1 and *exp*2 are WFT-expressions, *exp*1 op *exp*2 is also a WFT-expression, where op is a conventional logical operator.

Evaluation value of a WFT-expression *expr*, eval(*expr*) is defined as follows. For a WFT W'',

$$\begin{aligned}
&\text{eval}(\text{wft}(W')) = \texttt{true} \text{ iff } W' \sqsubseteq W'', \neg\exists W_1.W' \sqsubseteq W_1 \sqsubseteq W''. \\
&\text{eval}(\text{wfi}(W')) = \texttt{true} \text{ iff } W'' \sqsubseteq W', \neg\exists W_1.W'' \sqsubseteq W_1 \sqsubseteq W'. \\
&\text{eval}(\text{general}(W')) = \texttt{true} \text{ iff } W' \sqsubseteq W''. \\
&\text{eval}(\text{special}(W')) = \texttt{true} \text{ iff } W'' \sqsubseteq W'. \\
&\text{eval}(exp1 \,\text{op}\, exp2) = \text{eval}(exp1) \,\text{op}\, \text{eval}(exp2)
\end{aligned}$$

Both selection operations are similar with selection operator in relational algebra. However, there is a difference between those two: the result of σ is not always a closed WFT, meanwhile the result of σ' is always closed. That is, when σ is applied on a workflow, the result is not always a workflow.

Now we provide an operation to get a workflow from a set of activity objects, *get workflow* operation, noted as η. Its syntax is $\eta_W\, a$ or $\eta_W\{a_1, \cdots, a_n\}$). It gets a maximal workflow including an activity object a or activity objects $\{a_1, \cdots, a_n\}$ respectively, from a specified WFT W.

By using η, a workflow can be obtained from the result of σ.

5.1.2 Operations on Has-a Hierarchies

In general, a WFT organizes hierarchically by vertical flows. Hence there exists a requirement to hide some sub activity objects from a WFT to look at an overview of work. In order to satisfy the requirement, we provide operations *hide* and *show* as the reverse operator of *hide*.

Hide operation, noted as $\psi(W_1, W_2)$, returns a WFT $W_1 - W_2$ where W_1 is a WFT, W_2 is a closed WFT, and $W_2 \subseteq W_1$. Intuitively, this operator hides all the descendants of the activity objects in W_2. In other words, ψ works as an abstraction operation on a WFT. The obtained WFT does not always satisfy closed property.

Show operation, denoted as $\Psi\, W_1$, is the reverse operation of ψ. For a WFT W_1, this operation returns a WFT

$$W_1 \cup \bigcup_{a \in W_1} \text{descendant}(a).$$

W_1 may not be closed. On the other hand, the obtained WFT is always closed. Intuitively this operation shows all the descendants of W_1. In other words, Ψ works as a detailing operation on a WFT.

5.1.3 Operations on Specialization Hierarchies

Specialization hierarchies of WFTs are useful for retrieving WFTs from their instantiation relationships. For example, by using specialization hierarchies, we can retrieve all the instances of the same parent. This manipulation is frequently used in investigating the progress status of work.

General operation, noted as $\vartheta(W,\theta)$ is defined as

$$\vartheta(W,\theta) \stackrel{def}{=} \{W' | W'\theta = W\}.$$

This operation obtains a set of WFTs each of which is a parent of W over the partial ordering $\sqsubseteq$.

Special operation, noted as $\Theta(W,\theta)$, is defined as

$$\Theta(W,\theta) \stackrel{def}{=} \{W' | W' = W\theta\}.$$

This operation obtains a set of WFTs each of which is a child of a WFT W over the partial ordering $\sqsubseteq$.

We also define the transitive closure of ϑ and Θ, noted as ϑ^+ and Θ^+, respectively. These operation obtain all the ancestors and all the descendants on the specialization hierarchies, respectively.

5.2 Operations Handling WFTs

5.2.1 Grouping, Ungrouping

First we define an operation for grouping multiple activity objects into one activity object, and its reverse operation.

For a closed WFT $W = \{a_1, \cdots, a_n\}$, *grouping* operation, noted as $\omega\, W$, returns an activity object

$$a' = (\bigcup_{i=1}^{n} I_i - \bigcup_{i=1}^{n} O_i, \bigcup_{i=1}^{n} O_i - \bigcup_{i=1}^{n} I_i, \bigcup_{i=1}^{n} P_i, W),$$

and lets $W' = (W' - W) \cup \{a\}$ for all WFTs $W'.W \subseteq W'$. Intuitively this is a "grouping" operation. It regards a WFT W as a sequence of one work, and creates an activity object including W as subworks.

Next the reverse operation of grouping, *flattening* operation, noted as $\Omega\, a$, is defined. Let $a_1 = (I_1, O_1, P_1, S_1)$. For all activity objects a_1 where $a = (I, O, P, S) \in S_1$, this operation lets $S_1 = S_1 \cup S - \{a\}$ if $S \neq \emptyset$. If $S = \emptyset$, it does nothing.

5.2.2 Split, Concatenation

Split operations, noted as $\gamma(a, W)$, is defined as follows. For an activity object $a = (I, O, P, S)$ and a closed and consistent WFT $W = \{a_1, \cdots, a_n\}$ where $W \subseteq S$, this operation returns a WFT $W' = \{a'_1, a'_2\}$ where:

$$\begin{aligned} a'_1 &= (\bigcup_{i=1}^{n} I_i - \bigcup_{i=1}^{n} O_i, \bigcup_{i=1}^{n} O_i - \bigcup_{i=1}^{n} I_i, P, W) \\ a'_2 &= ((I - I'_1) \cup O'_1, (O - O'_1) \cup I'_1, P, S - W) \end{aligned}$$

Intuitively this operation splits a WFT S from a specified WFT W. In general, the obtained WFT W' is not always acyclic in $\Longrightarrow$. That is, two horizontal flows, $a'_1 \Longrightarrow a'_2$ and $a'_2 \Longrightarrow a'_1$, hold simultaneously. To make W' to be acyclic in $\Longrightarrow$, one of these two conditions must be satisfied: $I'_1 - O'_1 \subseteq I$ or $O'_1 - I'_1 \subseteq O$.

Concatenation operation, noted as $\Gamma(a_1, a_2)$, is closely related with split operation. If neither $a_1 \Longrightarrow a_2$ nor $a_2 \Longrightarrow a_1$ are satisfied, this operation lets $I_2 = I_2 \cup O_1$. If one of the two conditions are satisfied, this operation does nothing.

Intuitively it connects two activity objects a_1, a_2 by a horizontal flow. From the definition of horizontal flow, the outputs of a_1 must be added to the inputs of a_2 when no horizontal flow is found between a_1 and a_2.

5.3 Operations Handling Activity Objects

Some kinds of changes on WFTs, such as changing flows or adding new activity objects, need operations on activity objects in the WFTs directly. Moreover, operations handling WFTs shown in the previous section are implemented using operations on activity objects. For these reasons, operations handling activity objects directly are necessary for WFB.

In addition to the conventional set operations or the operations in relational algebra, we provide these operations listed below for handling activity objects:

- new(a, I, O, P, S)
  It creates a new activity object.

- destroy(a)
  It destroys a specified activity object a.

- foreach a in s do *Operator*
  It executes an operation WFT *Operator* for each activity object a in a set s. *Operator* can be any operator on an activity object. This operator is useful when the maintainer wants to do the same operations on the number of activity objects. For example, if yokota's works are all transferred to kunishima, the maintainer will do the

following operation:

$$\text{foreach } a \text{ in } \underset{yokota \in P}{\sigma} W_{all} \text{ do } a.P := \{kunishima\};$$

However, these operations may cause integrity violations of WFTs. For example, if the operation **destroy**(dispatch-1) is applied on the workflow "review-process" in Figure 3.2, "review-process" will be no more than a workflow because it is not closed and not connected. Hence the operations shown in this section are mainly for the maintainers of workflow base.

5.4 Operations on Activity Objects With Plugins

Plug-in operation, $\bowtie^+$, is provided for adding plugins such as time plugin or resource plugin to activity objects. It is defined as:

$$a \bowtie^+ L \overset{def}{=} a \bowtie L$$

where a is an activity object and L is a plugin. As shown in this definition, plug-in operation on an activity object is in actual same as natural join operations in relational algebra[1].

Plug-in operation $\bowtie^+$ on a WFT is defined as:

$$W \bowtie^+ L = \{a_1 \bowtie^+ L, \cdots, a_n \bowtie^+ L\}$$

where $W = \{a_1, \cdots, a_n\}$.

Plug-out operation, π^+, is defined as a reverse operator of $\bowtie^+$ such as:

$$\underset{L}{\pi^+} a_1 \overset{def}{=} (I, O, P, S)$$

[1]The notations of the operators in relational algebra are based on [Ull88].

where $a_1 = (I, O, P, S, L)$. Plug-out operation is a special case of projection operation in relational algebra.

Plug-out operation on a WFT can be defined as same as the plug-in operator on a WFT:

$$\underset{L}{\pi^+} W = \{\underset{L}{\pi^+} a_1, \cdots, \underset{L}{\pi^+} a_n\}$$

where $W = \{a_1, \cdots, a_n\}$.

Plug-in operation and plug-out operation can be applied into activity objects already plugged in by another plugins, with no modification of the definitions above.

5.5 Examples on Applying Operations

We show some examples of applying the operations on workflow-base.

5.5.1 Retrievals

Consider a situation doing a review process based on the workflow in Figure 3.2. The chair wants to investigate the progress of the review process of a submitted paper "A100". He gets the workflow of the process by applying this operation:

$$\eta \underset{A100 \in I}{\sigma} W_{all};$$

Then he finds the review process is delayed at reviewer *Taro*. He wants to know all the works *Taro* is doing. *Taro*'s all activity objects can be obtained by applying the following:

$$W_{taro} := \underset{Taro \in P}{\sigma} W_{all};$$

$$\begin{array}{lcl}
\text{registration-process} & = & \{\text{registration, send-registration}\} \\
\text{registration} & = & (\text{entry-sheet, participant-name, secretary,} \\
 & & \{\text{register into the participants list.}\}) \\
\text{send-registration} & = & (\text{participant-name, acceptance-sheet,} \\
 & & \text{secretary, } \{\})
\end{array}$$

Figure 5.1: WFT of a Registration Process

The precise of each workflows can be obtained by:

$$W_{taroPrecise} := \Psi\, W_{taro};$$

Consider another query. A committee member *jiro* wants to investigate that other committee members have asked *taro* to review submitted papers. Because review$_{11}, \cdots,$ review$_{nj_k}$ in Figure 3.3 are children of a WFT "review" over the partial ordering $\sqsubseteq$, this query is described as the following:

$$\underset{taro \in P}{\sigma} \underset{\mathrm{wfi}(review)}{\sigma'} W_{all};$$

5.5.2 Updates of Workflows

Figure 5.1 shows a workflow of a registration process of the same conference.

"send-registration" in Figure 5.1 and "send" in Figure 3.2 are almost the same process in the sense that a secretary sends something to someone. For this reason, these two processes can be joined into one process "send-object" by applying the following operations:

$\mathsf{new}(\text{send-object}, \text{address}, \text{objects}, \text{secretary}, \{\})$;
$\gamma(W_1, \{\text{send-review-report}\})$;
$\gamma(W_2, \{\text{send-registration}\})$;
$\vartheta(\{\text{send}\},$
$\{\text{address/participants-name}, \text{objects/acceptance-sheet}\})$;
$\vartheta(\{\text{send-registration}\},$
$\{\text{address/all-review-reports}, \text{objects/review-letter}\})$;

5.6 Summary

Operations to a WFB are defined as an extension of ordinary relational algebra. These operations get a set of WFTs or a set of activity objects from WFTs in WFB. They do not always keep the closure property of relational algebra. Some operations are provided in order to let a set of activity objects into a WFT. The combination of these operations with others makes the closure property to be kept.

When using these operations, users can organize new workflows from the workflows in WFB. For example, the following functions are available using the operations: views over workflows, such as to-do lists; investigation of progress; workflow reorganization; reuse of workflows.

The operations are classified into three categories: query operations, operations handling WFTs, and operations handling activity objects.

Chapter 6

Agents Reconsidered

To make a collaborative work possible, related programs and persons should have a common protocol, which resolve each heterogeneity with some protocol. We call these entities related with the collaborative work such as database systems, knowledge-base systems, constraint solvers, application programs, persons, and so on, as a problem solver. Therefore *agents*, wrapping problem solvers by a common protocol, should be considered in a collaborative work.

Considering a big problem such as workflow, we can find many applications which require multiple *heterogeneous* problem solvers:

- Modeling Heterogeneity
  The complexity of a given problem requires a combination of multiple heterogeneous problem solvers,
- Spatial Heterogeneity:
  Spatially distributed problem solvers are required to process a given problem.
- Temporal Heterogeneity:

For a new problem, a new problem solver is not necessarily developed: that is, multiple existing problem solvers must be reused for a new problem.

There have been some approaches: an arithmetic calculator in Prolog and a constraint logic programming language with a single constraint solver. Such a restricted approach seems to be neither flexible nor promising for most applications.

Further, considering the spread of distributed environments, there might be similar sources, each of which does not have complete information. In such an environment, we can frequently get better results by accessing and merging multiple information sources or multiple problem solvers. In other words, cooperation among distributed sources are frequently required. Considering such applications and environments, heterogeneous, distributed, cooperative problem solvers will become more important and can play a role of a very large knowledge-base. The situation is almost the same, because there might be many agents who can possibly solve a given problem. A coordinator agent can select and judge an agent for the work.

From workflow and workflow base points of view, in a unit of workflow definition, all agents should have the same protocol, for which we use an environment concept as in Helios. In this chapter, we discuss how to define such environments for workflow management.

6.1 Logic-Oriented Definition

In the previous chapters, we consider a workflow as a set of activity objects by corresponding an activity object to a work. An agent which (or who) is responsible for a work is embedded in its activity object.

However, as an agent usually has multiple kinds of works, we consider another definition of an agent:

$$a/[w(I) = O] \Leftarrow S|C$$

where a is an agent identity, w is an work identity, I is a set of inputs, O is a set of outputs, S is a set of sub-agents, which are called by a, and C is a set of constraints which are delivered from a to S. As in deductive object-oriented languages such as F-logic[KL89], we can correspond the rule to conventional object-orientation concepts: a work identity corresponds to a method identifier, I is a set of input values, O is a set of return values, and $S|C$ is its implementation.

As an agent do many kinds of works, it is defined as follows:

$$\begin{gathered} a/[w_1(I_1) = O_1] \Leftarrow S_1|C_1 \\ a/[w_2(I_2) = O_2] \Leftarrow S_2|C_2 \\ \vdots \\ a/[w_n(I_n) = O_n] \Leftarrow S_n|C_n \end{gathered}$$

or

$$\begin{aligned} & a/[w_1(I_1) = O_1, w_2(I_2) = O_2, \cdots w_n(I_n) = O_n] \\ & \Leftarrow S_1 \cup S_2 \cup \cdots \cup S_n | C_1 \cup C_2 \cup \cdots \cup C_n \end{aligned}$$

In this definition, we introduce concurrency in a easily because each work does not share variables with other works, however it is very difficult to control it. Further, as each agent is not encapsulated, it is difficult to correspond an agent to a person who is responsible for its work.

6.2 Agent-Based Definition

Considering autonomous agents who execute various works, we had better introduce a concept of an *agent*, which can be defined independently from definitions of workflows. Each agent can estimate workload and schedule a set of works. In this subsection, we define an agent in the sense of *Helios* and discuss an agent as a person.

6.2.1 Agents and Environments

As mentioned in Section 2.4, an agent is defined as follows:

agent := (capsule, problem-solver) |
(capsule, environment, {agent$_1$, ..., agent$_n$})

where a *capsule* is a module which contains a translator, and each problem-solver is enclosed in a capsule. An encapsulated problem solver is called an *agent*, and a problem-solver is called a *substance*.

A *simple* agent is defined as a pair of a *capsule* and a problem-solver: conceptually, the problem-solver is encapsulated in the capsule.

A *complex* agent is defined as a triple of a capsule, an environment, and a set of agents (agent$_1$,...,agent$_n$), where an environment is a field where agent$_1$,...,agent$_n$ can exist and communicate mutually. Conceptually, a pair of an environment and a set of agents can be considered as a problem-solver; a new agent can be defined by encapsulating them. That is, an agent can be hierarchically organized. We show such structure in Figure 2.2. Since an encapsulated problem-solver or an environment can be considered as an agent and the outside of an agent is an environment, the user can be considered as an environment.

A common space for agents is called an *environment*. An environment takes care of message-passing between agents in it, and manages global information for those agents. Each agent has its own logical name that is unique in the environment.

A capsule and an environment is defined as follows:

capsule := (agent-name, methods, self-model,
translation-rules, negotiation-strategy)
environment := (agent-names, common-type-system,
negotiation-protocol, ontology)

In a capsule, an *agent-name* is an identifier of the agent. Each agent has its own agent-name that is unique in the environment. A *method* is a definition of *import methods* and *export methods*. An import method defines a method by which the agent is called, and an export method defines a method that the agent can call. An agent with only import methods is called *passive* and an agent with both methods is called *active*: only an agent which sends new messages through export methods can negotiate with other agents. A *self model* defines what the agent can do in terms of the name of functions provided by the agent. An environment extracts the necessary information from self models in agents to dispatch messages between agents. *Translation-rules* define translation between internal representation of the agent and common representation given by its environment in the initialization of the problem solving system.

In an environment, *agent names* state which agents are in the environment. A *common type system* defines a type system used to type all messages in the environment. A *negotiation protocol* defines the protocol used by all agents in the environment. Under a *negotiation protocol* in an environment, each agent defines a *negotiation strategy* to communicate with other agents. An *ontology* defines the transformation of the contents of messages between agents, while a capsule converts the syntax and type of messages between the common type system and the intrinsic type system of the corresponding problem-solver.

To define this information, we introduce a capsule description language *CAPL* (CAPsule description Language), and an environment description language *ENVL* (ENVironment description Language). Programs written in those languages are processed by their corresponding compilers.

Although various information is defined locally in each environment and each agent, a *message* between agents is in the form of a global *communication protocol* consisting of the following:

- Message identifier
  An identifier used for identifying a message. This field is unique within an environment.
- Message type
  As described above, a message is either a method invocation or an answer. The former message is called a *query message*, and the latter message is called a *reply message*. This field is used to distinguish a query message from a reply message.
- Sender agent identifier
  This field contains the agent name of the agent that sends this message.
- Designation of destination agents
  The methods of designating destination agents in a query message are described below. In a reply message, this field contains the agent name of the agent that is the sender of the corresponding query message.
- Transaction identifier
  If the update of the content of a destination problem-solver is attendant on the invocation of a message, then a transaction identifier is required to control it. This field contains a transaction identifier. For nested transactions, a transaction identifier with a nested structure is used.

- Status
  This field contains information on the status of invoked methods for error handling.
- Message content
  In a query message, this field contains a method invocation, and in a reply message, this field contains the answer to the invocation.

In the previous subsection, we consider some difficulties in logic-oriented approach. On the other hand, in this agent-oriented approach, as each agent is autonomous, concurrency control can be done by a capsule or a substance, and correspondence with a person is made possible by wrapping by a capsule.

6.2.2 Coordination among Agents

In our workflow model, (I, O, P, S), an agent P controls a set, $S = \{a_1, a_2, \cdots, a_m\}$, of activity objects. It can be written as a complex agent as follows:

$$P = (c, e, \{a_1, a_2, \cdots, a_n\})$$

In Helios, messages are dispatched as follows:

- Initialization
  First, during the initialization of agent processes in the environment, the environment constructs a map of a logical agent name and a physical process address (or IP address). Secondly, the environment gathers method information and function information in self models from each agent and constructs two kinds of maps: a method and an agent name; a function name and an agent name. Such maps work for dispatching messages among agents.

- Dispatching:

  As a method or a function does not necessarily corresponds to an agent uniquely, a message is possibly sent to multiple agents. This mechanism is useful for the followings:

  - It is unnecessary to specify an agent name in problem solvers explicitly.
  - It is possible to send simultaneously a message to possible agents.

  An environment decides to send a message sequentially or in parallel to candidates listed by the maps, and processes answers sequentially or by grouping as a set. In the case of set grouping, aggregation functions can be specified in an environment. Such a mode can be selected in a query message.

As already mentioned, differently from Helios, P is responsible for dispatching works (messages) to agents and coordinating their execution. In the sense, P provides the capsule c and includes the functions of an environment e.

A message is analyzed and its corresponding processing plan is constructed as a dependency graph by a parent agent as in a query in conventional distributed databases. Synchronization information between sub-messages is attached to each sub-message and controlled by the capsule of each agent: i.e.,if necessary, child agents can communicate with each other. The coordination protocol between a parent agent and child agents is rather simple:

- success/failure:

  An child agent sends a success or failure message to the parent agent. The parent agent redispatches the work to another child agent or cancels all works dispatched to all child agents related to the work. There might be various reasons of failures: unsatisfaction of time constraints, lack of resources, and so on.

- resource requirements:

  A child agent can request some resources necessary for executing its own work to the parent agent. The parent agent sends additional resources to the child agent if the parent has enough resources. If the parent does not have necessary resources, it requests them to other child agents. If a child agent has enough ones, it sends them to the parent.

We assume that a person involved in some workflow has an appropriate interface to the capsule of the corresponding agent. As a person can do many works, he is connected to many capsules. In this thesis, we do not discuss the topic.

6.2.3 Relations between Workflow Base and Agent Definitions

We have two kinds of definitions about a workflow: a workflow definition and a agent definition. As mention in Section 3.5, a workflow base consists of a workflow definition and its execution model. A workflow definition corresponds to a static aspect of workflow, while an execution model corresponds to its dynamic aspect, where activity objects are dynamically generated and activated in P-box and their relations are defined in C-box in the form of definite clauses. A definite clause, $p \leftarrow q_1, q_2, \cdots, q_n$, is a dispatching plan, which corresponds to an activity object p, which is also in P-box. That is, an isolated activity object in P-box just corresponds to an agent definition.

6.3 Summary

Considering a big problem such as workflow, we can find many applications which require multiple *heterogeneous* problem solvers. To make a collaborative work possible, related programs and persons should have a common protocol, which resolve each heterogeneity with some protocol.

There have been some approaches: an arithmetic calculator in Prolog and a constraint logic programming language with a single constraint solver. Such a restricted approach seems to be neither flexible nor promising for most applications.

From workflow and workflow base points of view, in a unit of workflow definition, all agents should have the same protocol, for which we use an environment concept as in Helios. In this chapter, we discussed how to define such environments for workflow management.

A problem solver, a generic term of database systems, knowledge-base systems, constraint solvers, application programs, persons, and so on, is called an *agent* by wrapped by a common protocol.

Considering such applications and environments, heterogeneous, distributed, cooperative problem solvers will become more important and can play a role of a very large knowledge-base. The situation is almost the same, because there might be many agents who can possibly solve a given problem. A coordinator agent can select and judge an agent for the work.

Chapter 7

System Architecture of Workflow Base

In this chapter, we discuss about how to implement workflow base, mainly from system architecture point of view. Though workflow base is closely related to database systems, it has various features not found in traditional database systems, such as an execution model based on production systems. First we show a system interface between workflow base and application programs. After that, overall of the architecture and each functional components are described.

7.1 The Interface Between Applications and Workflow Base

Workflow base has two roles for its application programs, called *workflow clients*: as a workflow database and as a workflow server. That is, workflow base is more than a workflow repository in workflow management systems; it can behave as a server of workflow management systems itself because it provides an execution model of each workflows.

Therefore workflow base has two modules in its interface: interface to the workflow database and interface to the workflow server.

The former provides database operations described in Chapter 5. Workflow clients uses this interface as querying workflow base, such as retrieving, reorganizing workflows, etc.

Meanwhile the latter provides the protocol between the capsule and the problem solver mentioned in Chapter 6. In the coordination system point of view, all agents related with a collaborative work of workflows are implemented in the workflow server and coordinate each other in the style of Chapter 6. A workflow client is denoted as a problem solver in this structure.

7.2 The System Architecture of Workflow Base

Workflow base organizes from four functional components: Database, WFT organizer, WFT interpreter, and Query processor. An overview of the system architecture is shown in Figure 7.1.

Database stores all information about WFTs such as their definitions, all activity objects, all P-box and C-box components, assignments, progress status, etc. Remark that this is no need for object-oriented database, though we use the term "objects" in workflow base. Activity object is not encapsulated because it has no concepts of methods. Hence it is more similar to tuple in relational database than object in object-oriented database.

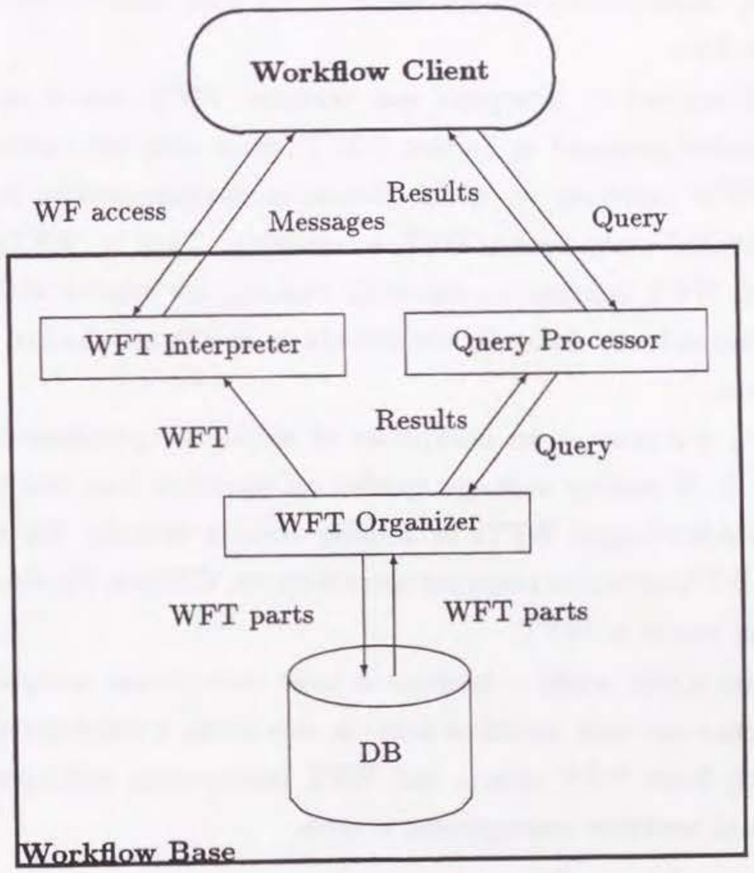

Figure 7.1: System Architecture of Workflow Base

WFT organizer constructs WFTs from the information in Database, and decomposes WFTs in data who belongs to the schema in Database. In other words, WFT organizer translates WFTs and schema of Database each other. It plays a role of hiding database implementation — relational database, object-oriented database, etc. — from other components of workflow base.

WFT interpreter interprets and executes WFTs based on its execution model proposed in Section 3.2. It must keep the current status of all WFTs currently on work. Hence, in implementation level, it is newly invoked every time a WFT is available. That is, WFTs now on work and WFT interpreters currently running are related with one-to-one correspondence. Integrity constraints on WFTs are checked by WFT interpreter.

Query processor is an interpreter of workflow operations shown in Chapter 5. It mainly manages queries on workflow base and the operations which changes WFTs or activity objects directly. On the other hand, WFT interpreter manages operations on WFTs in the frame of the execution model of WFT.

Agents in real world — humans in most cases — use *workflow clients* to communicate with workflow base. In this sense, a client-server system consisting from WFT clients and WFT interpreters corresponds with traditional workflow management system.

7.3 Summary

In this chapter system architecture of workflow base is discussed. Though the roles of each software components are briefly mentioned, more discussions about the architectures of them should be needed. In especial,

these two points must be discussed: interpretation procedure for the execution model WFT interpreter should have, and database schema for modeling WFTs.

Chapter 8

Comparisons with Related Researches

Many researches about workflow management systems have been done from the various aspects. Moreover, there are also similar concepts as workflows in various research areas, such as groupware, process modeling, and software process engineering. In this chapter, we pick up several related researches from various research areas, and compare them with our model especially from the workflow model point of view.

8.1 Groupware Approach

Automation of office work has been one of the hot topics in the research area of groupware [EN80], and many similar concepts as workflow management systems had been proposed before the term "workflow" are widely known.

8.1.1 OM-1

OM-1 [Ish86, IO91], categorized as an *office procedure model*, represents knowledge of well-structured cooperative work both in control flow and in office structures, such as data structures and organization structures, using object-oriented modeling. Three objects — data, activity, and agent — are provided as basic office objects. Office procedure is represented as a graph whose nodes are basic office objects. Links between office objects represent various relationships between office objects, such as control flow, responsibility, belonging, and operations on data. Four styles of connections for control flows are provided: AND join, AND fork, XOR join, and XOR fork.

OM-1 has the ability to represent control flow and office structures in a uniform style. However, this ability makes office procedures to be much complex. Our workflow model manages control flows and office structures in a separate way but with relationships. This makes workflows to be simple. Moreover, our model has a formal execution model which OM-1 does not provide.

8.1.2 Action Workflow

Action Workflow [MMWFF92] is a workflow model based on conversation act theory [WF86]. It represents all cooperative works as an action workflow loop, a loop of four processes: proposal, agreement, performance, and satisfaction. Each process in an action workflow loop can be an action workflow loop. Hence an action workflow is organized hierarchically in general. This paper also gives an architecture of workflow management systems based on action workflow model. The Workflow Management Server, a core system of the workflow management system, manages both the definitions of workflows and the progress of workflows using databases.

Action Workflow is a model suitable for ill-structured cooperative work. However, processes provided in Action Workflow are too primitive to represent well-structured cooperative work in a simple way. Moreover, the execution semantics of the workflows in Action Workflow is not clear. Data models of the workflows in the Workflow Management Server is not discussed.

8.1.3 Regatta

The Regatta project, formed in 1991 to develop software to support workgroups and to aid in reengineering work processes, proposes a model for collaborative work and a graphical language to support this model [Swe93]. In this model, work process is modeled as a network of tasks, each of which represents a task request, commitment, or question. Though this modeling concept is similar with Action Workflow, the model provides more functions than Action Workflow such as dynamic modification of flows, incremental automation, etc.

The workflow model of the Regatta project has the similar concepts as WFT/WFI. Before workflow instances are executed, they are created from workflow templates by assigning real-world entities into variables. The features such as hierarchical workflows, dynamic changes, view functions as private to-do lists, etc., are also similar as those of our models. However, formal semantics is not given in these models. Moreover, the view functions in these models are somewhat ad-hoc, not based on formal operations.

8.1.4 MEGUMI

Tarumi et at. proposed and developed a workflow management system based on MEGUMI, rule based e-mail system [TTY95, YTT95]. In this system, workflows are realized by the formed e-mail circulation with the rule-based control in MEGUMI. Hence this system does not assume a central workflow server. The rules realizing workflows are described as ECA rules by HyperScheme, a programming language extending Scheme. They also discussed about databases for workflow management [YTT95].

Problem of the system is the capability of HyperScheme from the workflow definition point of view. As HyperScheme is a programming language for general purpose, it can represent illegal workflows. Though GUI based workflow definition tool is provided in order to restrict the ranges of workflows to be defined, there still remains possibilities users define illegal workflows by customizing rules not using the workflow definition tool.

8.1.5 WorkWeb

WorkWeb [TIAT95, TIT^{+}96] is much different from other workflow management systems. It deals with some constraints about the common resources of the organizations — objects, persons, and money. Agents are provided for each person, each resources, each shared data, and each workflows, and they negotiates each other to resolve conflicts of these constraints. The negotiation is done over a number of work processes running in parallel at one organization.

WorkWeb can treat more wider constraints than our workflow model shown in Chapter 4. However, the way for managing each workflow process is not discussed in WorkWeb. This means the constraint management in WorkWeb can be compatible with our workflow model.

8.2 Process Modeling Approach

Process modeling is a research area closely related with workflows. It has been studied mainly in software process modeling, in modeling of reactive systems, and in office information systems. In recent years, the similarities between process modeling and workflows are pointed out [Rob96], and researches of applying process modeling into workflows have been studied.

8.2.1 Activity Management System

Activity Management System (AMS) [TLA91] is a knowledge-based system which supports the representation and execution of procedural knowledge, which is expressed like as workflows. In this system, the architecture of AMS can be stratified into three layers: a control structure to handle the interruption, resumption, and cancelation of tasks; a mechanism to schedule tasks that are in progress; a mechanism to handle missing information. This system also provides some office operators such as send, request, acknowledge, and answer. Concatenating these operators, it organizes procedural knowledge which is similar with speech act model [WF86]. The operators are implemented based on Petri nets.

Though AMS can express cooperative works flexibly, there is no discussion about agents as executors of the units of work.

8.2.2 Statechart

Statechart [Har87] is a hierarchical finite automaton with concurrent execution and broadcast mechanisms for communications between them. Its semantics is formerly defined like finite automaton using transitions between sets of exclusive states, called configurations. Based on statechart, a computerized environment for the development of reactive systems, called STATEMATE [HLN$^+$90], is proposed. STATEMATE has a design database of statechart with general query language, which have an expressive power almost same as that of the conjunctive query in relational databases. Though STATEMATE is proposed for modeling reactive systems, it is also suitable for process modeling. From this viewpoint, we can find some papers applying statecharts into software process modeling [KH89, HK89].

Many similarities can be found between our workflow model and statechart/STATEMATE. First, both are based on finite automaton with some extensions like hierarchy, concurrent execution, synchronization, etc. Secondly, both give formal semantics of the models. Thirdly, both systems use database for process data management. And finally, both systems provide general query languages based on relational algebra. The query language of STATEMATE can express queries on a set of states as well as on a set of statecharts, although few discussions about the data model of statecharts are found in [HLN$^+$90].

In spite of the similarities with our model, statecharts lacks some features which are important for workflow management. Statechart provides no function to change its own execution dynamically; there is no discussion about agents as executors of work, instantiation like WFT/WFI, integrity constraints of the model, etc.

8.2.3 KyotoDB

KyotoDB [MA90] is a software project database based on an object model. In this system, both process programs and data are encapsulated into objects, and stored into one database. Work process, called unit workload in KyotoDB, is modeled as a sequence of work slice, a directed graph of work process nodes, validation nodes, and communication nodes with other unit workloads.

KyotoDB treats the units of work as an object in a strict sense; each object permits to be accessed only via its own methods. When the user wants to access objects based on set operations, he has to use general query languages on OODBMS such as OQL.

8.3 Database Approach

As mentioned in Section 2.1, almost all researches about workflows from the database point of view treat transaction management among processes. We found few papers discussing workflows from the data model point of view.

8.3.1 C&Co

C&Co [FKB95] is a programming language with some coordination extension features in C. Concurrent execution of processes, dependencies between processes, transactions with retrying or compensating options, synchronization, etc. are newly introduced in C.

In order to implement workflow management systems with database technology, programming language should have these features newly introduced in C&Co. However, in the workflow language point of view, C&Co does not provide sufficient capabilities for representing workflows; that is, the functions C&Co provide for workflow management are too primitive to represent workflows.

8.3.2 Event-Condition-Message Rules

M. Rusinkiewicz et al. propose a rule-based workflow model closely related with database technologies [JCR96]. In this model, a workflow is represented as a set of Event-Condition-Message (ECM) rules, whose semantics is "When event E occurs, evaluate condition C. If it evaluates to true, send message M". Tasks are implicitly related to each other through ECM rules; that is, when a task is finished, an event of "finishing" is sent to a workflow and the corresponding ECM rules are fired and evaluated. If a condition evaluates to true, a message is sent to a set of tasks to be invoked. A workflow in this model is depicted as a hierarchical state transition diagram. They also discusses database schema to store workflow definitions and workflow instances, query languages, and system architecture of prototype.

Their standpoints are quite similar as ours: they use database management systems to store the static specification of workflows, as well as their run-time information; they provide a method for querying the database to get several information about workflows. However, workflows are stored as a directed graph in their model. This makes editing a workflow specification by database query languages to be difficult, as they pointed out in [JCR96].

Chapter 9

Conclusions

In this thesis we proposed a flexible framework of workflow management suitable for database technologies, *workflow base*.

In this model, a workflow is defined as a set of objects (*activity objects*), each of which corresponds with the units of work in workflows. Two kinds of flows in workflow base, horizontal flows and vertical flows, are treated as constraints among activity objects, hence they are created dynamically from the definitions of activity objects. We provide important features of workflow management systems as applications of database technologies: workflow instantiation based on generalization/specialization hierarchy of workflows; execution model based on production systems; loopback flows are defined using ECA rules; view functions and query functions as a set of database operations; various constraints on workflows as integrity constraints; agent as a problem solver under heterogeneous distributed environment.

Workflow base has an important contribution into workflow researches other than these original features.

When a number of agents cooperate each other, they share some data in almost all cases. In this sense, data sharing is essential activity in cooperative work, as noted in [GS87], and database will surely be an infrastructure of groupware technology. However, it is not in today's groupware. Why?

I think one of the reasons is a gap between groupware from database point of view and database from groupware point of view. From database point of view, groupware is software on a distributed environment, sometimes executed automatically; from groupware point of view, on the other hand, database is software to make capabilities of human beings to be higher. Both viewpoints are true; database technologies supporting cooperative work should satisfy the requirements from both viewpoints.

Workflow base supports well-structured cooperative work from both viewpoints: to provide a formal but simple workflow model with concrete execution semantics from database point of view; to provide general and powerful query language on workflows from groupware point of view.

In this sense, research of query languages on workflow base is the most important future work. Though an operation set on workflow base proposed in Chapter 5 is based on relational algebra, it is quite exhaustive. From the algebraic point of view, relational algebra has many good properties: equality with relational calculus, relationships with Datalog, etc. Refinement of the operation set to establish "workflow algebra" would give powerful and easy-to-use workflow management systems, as relational database did in the database area.

References

[AAA+96] G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, R. Günthör, and C. Mohan. Advanced transaction moldels in workflow contexts. In *Proceedings of the 12nd International Conference of Data Engineering*, February 1996.

[AS94] Kenneth R. Abbott and Sunil K. Sarin. Experience with workflow management: Issues for the next generation. In *Proceedings of the Conference on Computer-Supported Cooperative Work*, pages 113–120. ACM Press, October 1994.

[AS96] Gustavo Alonso and Hans-Jörg Schek. Research issues in large workflow management systems. In *Proceedings of NSF Workshop on Workflow and Process Automation in Information Science*, pages 126–132, May 1996.

[AYT95] Akira Aiba, Kazumasa Yokota, and Hiroshi Tsuda. Heterogeneous distributed cooperative problem solving system Helios and its cooperation mechanisms. *International Journal of Cooperative Information Systems*, 4(4):369–385, December 1995.

[BN95] Richard Blumenthal and Gary J. Nutt. Supporting unstructured workflow activities in the Bramble ICN system. In *Proceedings of ACM Conference on Organizational Computing Systems*, pages 130–137, August 1995.

[BTKdlT93] Douglas P. Bogia, William J. Tolone, Simon M. Kaplan, and Eric de la Tribouille. Supporting dynamic interdependencies among collaborative activities. In *Proceedings of ACM Conference on Organizational Computing Systems*, pages 108–118, November 1993.

[CB88] Jeff Conklin and Michael L. Begeman. gIBIS: A hypertext tool for exploratory policy discussion. *ACM Transactions on Office Information Systems*, 6(4):303–331, October 1988.

[CL84] W. Bruce Croft and Lawrence S. Lefkowitz. Task support in an office system. *ACM Transactions on Office Information Systems*, 2(3):197–212, 1984.

[CMB+90] Terrence Crowley, Paul Milazzo, Ellie Baker, Harry Forsdick, and Raymond Tomlinson. MMConf: An infrastructure for building shared multimedia applications. In *Proceedings of the Conference on Computer-Supported Cooperative Work*, October 1990.

[EGR91] Clarence Ellis, Simon Gibbs, and Gail Rein. Groupware: Some issues and experiences. *Communications of the ACM*, 34(1):38–58, January 1991.

[EKR95] Clarence Ellis, Karim Keddara, and Grzegorz Rozenberg. Dynamic change within workflow systems. In *Proceedings of ACM Conference on Organizational Computing Systems*, pages 10–21, August 1995.

[EN80] Clarence A. Ellis and Gary J. Nutt. Office information systems and computer science. *ACM Computing Surveys*, 12(1):27–60, March 1980.

[FGHW88] Fernando Flores, Michael Graves, Brad Hartfield, and Terry Winograd. Computer systems and the design of organizational interaction. *ACM Transactions on Office Information Systems*, 6(2):153–172, April 1988.

[FKB95] Alexander Forst, Eva Kühn, and Omran Bukhres. General purpose work flow languages. *Journal of Distributed and Parallel Databases*, 3(2):187–218, April 1995.

[FS86] Gregg Foster and Mark Stefik. Cognoter, theory and practice of a collaborative tool. In *Proceedings of the Conference on Computer-Supported Cooperative Work*, pages 7–15, 1986.

[GH94] D. Georgakopoulos and M. Hornick. A framework for enforceable specification of extended transaction models and transactional workflows. *International Journal of Intelligent and Cooperative Information Systems*, 3(3), September 1994.

[GHS95] Dimitrios Georgakopoulos, Mark Hornick, and Amit Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Journal of Distributed and Parallel Dataabases*, 3(2):119–153, April 1995.

[GS87] Irene Greif and Sunil Sarin. Data sharing in group work. *ACM Transactions on Office Information Systems*, 5(2):187–211, April 1987.

[Har87] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[Hew86] Carl Hewitt. Offices are open systems. *ACM Transactions on Office Information Systems*, 4(3):271–287, July 1986.

[HK89] Watts S. Humphrey and Marc I. Kellner. Software process modeling: Principles of entity process models. In *Proceedings of the 11th International Conference on Software Engineering*, pages 331–342, 1989.

[HLN$^+$90] David Harel, Hagi Lachover, Amnon Naamad, Amir Pnuell, Michael Politi, Rivi Sherman, Aharon Shtull-Trauring, and Mark Trakhtenbrot. STATEMATE: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16(4):403–414, April 1990.

[IHH96] Kyoko Iiduka, Hiroaki Higaki, and Yutaka Hirakawa. TORES: Task order evolution system. Technical Report GW18-6, IPSJ Special Interest Group on Groupware, June 1996. in Japanese.

[INMS96] Motoki Isaka, Hiroshi Nunokawa, Masatoshi Miyazaki, and Norio Shiratori. Metalevel description in workflow management system. Technical Report GW15-21, IPSJ Special Interest Group on Groupware, January 1996. in Japanese.

[IO91] Hiroshi Ishii and Masaaki Ohkubo. Message-driven groupware design based on an office procedure model, OM-1. *Journal of Information Processing*, 14(2):184–191, 1991.

[Ish86] Hiroshi Ishii. Office work specification by office model OM-1. Technical Report OS86-24, IPSJ Special Interest Group on Office Systems, September 1986.

[JCR96] Dominique Jean, Andrzej Cichocki, and Marek Rusinkiewicz. A database environment for workflow specification and execution. In *Proceedings of International Symposium on Cooperative Database Systems for Advanced Applications*, volume 2, pages 491–500, Kyoto, December 1996.

[JLJL82] Peter Johnson-Lentz and Trudy Johnson-Lentz. Groupware: The process and impacts of design choices. In E. B. Kerr and S. R. Hiltz, editors, *Computer-Mediated Commmunication Systems: Status and Evaluation.* Academic Press, 1982.

[JMR92] Matthias Jarke, Carlos Maltzahn, and Thomas Rose. Sharing processes: Team coordination in design repositories. *International Journal of Intelligent and Cooperative Information Systems*, 1(1):143–167, March 1992.

[KCM91] Simon M. Kaplan, Alan M. Carroll, and Kenneth J. MacGregor. Supporting collaborative processes with ConversationBuilder. In *Proceedings of ACM Conference on Organizational Computing Systems*, pages 69–79, November 1991.

[KH89] Marc I. Kellner and Gregory A. Hansen. Software process modeling: A case study. In *Proceedings of the 22nd Annual Hawaii International Conference on System Science*, volume II, pages 175–188. IEEE, 1989.

[KK95] Takeo Kunishima and Yahiko Kambayashi. Workflow model on a workflow management system WorkFlowBase. Technical Report DBS104-41, IPSJ Special Interest Group on Database Systems, July 1995. in Japanese.

[KL89] M. Kifer and G. Lausen. F-Logic — a higher order language for reasoning about objects, inheritance, and schema. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 134–146, Portland, June 1989.

[KS95] Narayanan Krishnakumar and Amit Sheth. Managing heterogeneous multi-system task to support enterprise-wide operations. *Journal of Distributed and Parallel Databases*, 3(2):155–186, April 1995.

[KY96] Takeo Kunishima and Kazumasa Yokota. Flexible workflow frameworks for supporting collaborative work. In *Proceedings of International Symposium on Cooperative Da-

tabase Systems for Advance Applications, volume 2, pages 501–508, Kyoto, December 1996.

[MA90] Yoshihiro Matsumoto and Tsuneo Ajisaka. A data model in the software project database KyotoDB. *Advances in Software Science and Technology*, 2:103–121, 1990.

[Mah93] Dirk E. Mahling. Enactment theory as a paradigm for enabling flexible workflows. In *Proceedings of ACM Conference on Organizational Computing Systems*, pages 202–209, November 1993.

[MCC95] Dirk E. Mahling, Noël Craven, and W. Bruce Croft. From office automation to intelligent workflow systems. *IEEE Expert*, 10(3):41–47, June 1995.

[MD89] Dennis R. McCarthy and Umeshwar Dayal. The architecture of an active data base management system. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 215–224, June 1989.

[MGL$^+$87] Thomas W. Malone, Kenneth R. Grant, Kum-Yew Lai, Ramana Rao, and David A. Rosenblitt. Semi-structured messages are surprisingly useful for computer supported cooperative work. *ACM Transactions on Office Information Systems*, 5(2):115–131, April 1987.

[ML84] Murray S. Mazer and Frederick H. Lochovsky. Logical routing specification in office information systems. *ACM Transactions on Office Information Systems*, 2(4):303–330, October 1984.

[MMWFF92] Raúl Medina-Mora, Terry Winograd, Rodrigo Flores, and Fernando Flores. The Action Workflow approach to workflow management technology. In *Proceedings of the Conference on Computer-Supported Cooperative Work*, pages 281–288, November 1992.

[Ple95] Dimitris Plexousakis. Simulation and analysis of business processes using GOLOG. In *Proceedings of ACM Conference on Organizational Computing Systems*, pages 311–322, August 1995.

[Rob96] William N. Robinson. Goal-oriented workflow analysis and infrastructure. In *Proceedings of NSF Workshop on Workflow and Process Automation in Information Science*, pages 31–37, May 1996.

[RS94] Marek Rusinkiewicz and Amit Sheth. Transactional workflow management in distributed systems. In *Proceedings of International Workshop on Advances in Databases and Informations Systems*, pages 18–33, May 1994.

[SAM91] Sunil K. Sarin, Kenneth R. Abbott, and Dennis R. McCarthy. A process model and system for supporting collaborative work. In *Proceedings of ACM Conference on Organizational Computing Systems*, pages 213–224, November 1991.

[SBF$^+$87] M. Stefik, D. G. Bobrow, G. Foster, S. Lanning, and D. Tartar. WYSIWIS revised: Early experiences with multiuser interfaces. *ACM Transactions on Office Information Systems*, 5(2):147–186, April 1987.

[Sch96] Thomas Schäl. *Workflow Management Systems for Process Organisations*. Number 1096 in Lecture Notes in Computer Science. Springer-Verlag, 1996.

[SFB$^+$87] M. Stefik, G. Foster, D. G. Bobrow, K. Kahn, S. Lanning, and L. Suchman. Beyond the chalkboard: Computer support for collaboration and problem solving in meetings. *Communications of the ACM*, 30(1):32–47, 1987.

[SMK90] Allan Shepherd, Niels Mayer, and Allan Kuchinsky. Strudel — an extensible electronic conversation toolkit. In *Proceedings of the Conference on Computer-Supported Cooperative Work*, pages 93–104, October 1990.

[Suc83] Lucy A. Suchman. Office procedure as practical action: Models of work and system design. *ACM Transactions on Office Information Systems*, 1(4):320–328, 1983.

[Swe93] Keith D. Swenson. Visual support for reengineering work processes. In *Proceedings of ACM Conference on Organizational Computing Systems*, pages 130–141, November 1993.

[TIAT95] Hiroyuki Tarumi, Yoshihide Ishiguro, Takayoshi Asakura, and Atsushi Tabuchi. Beyond workflow: A proposal of work web system. Technical Report GW12-10, IPSJ Special Interest Group on Groupware, June 1995. in Japanese.

[TIT$^+$96] Hiroyuki Tarumi, Yoshihide Ishiguro, Atsushi Tabuchi, Kenji Yoshifu, Koji Kida, and Takayoshi Asakura. An implementation of the WorkWeb system. Technical Report GW15-22, IPSJ Special Interest Group on Groupware, January 1996. in Japanese.

[TLA91] Michel Tueni, Jianzhong Li, and James Ang. Knowledge-based office automation and CSCW. *Studies in Computer Supported Cooperative Work*, pages 183–194, 1991.

[TTY95] Hiroyuki Tarumi, Atsushi Tabuchi, and Kenji Yoshifu. Workflow implementation with rule-based e-mail. *Transactions of Information Processing Society of Japan*, 36(6):1322–1331, June 1995. in Japanese.

[Ull88] Jeffrey D. Ullman. *Priciples of Database and Knowledge-Base Systems*, volume 1. Computer Science Press, 1988.

[WF86] Terry Winograd and Fernando Flores. *Understanding Computers and Cognition.* Addison-Wesley Publishing Company Inc., 1986.

[wfC] Workflow Management Coalition homepage. http://www.aiai.ed.ac.uk/WfMC/.

[Wor96] Workflow Management Coalition. Workflow management coalition terminology & glossary, issue 2.0. available from http://www.aiai.ed.ac.uk/WfMC/, June 1996. Document Number WFMC-TC-1011.

[YA94] Kazumasa Yokota and Akira Aiba. A new framework of very large knowledge bases. In K. Fuchi and T. Yokoi, editors, *Knowledge Building and Knowledge Sharing*, pages 192–199. Ohmsha and IOS Press, 1994.

[YKN96] Kazumasa Yokota, Takeo Kunishima, and Hideyuki Nakanishi. Hypothetical query processing for workflow managemen. In *Proceedings of the Seminar on Logical Databases and the Meaning of Change*, pages 44–49, Dagstuhl, September 1996. UPMAIL Technical Report No. 129, Uppsala University.

[YKN97] Kazumasa Yokota, Takeo Kunishima, and Hideyuki Nakanishi. A framework of query processing for workflow management. *Lecture Notes in Artificial Intelligence*, 1997. to appear.

[YTT95] Kenji Yoshifu, Atsushi Tabuchi, and Hiroyuki Tarumi. Co-action of a workflow system and a database. Technical Report GW9-23, IPSJ Special Interest Group on Groupware, January 1995. in Japanese.

List of Publications by the Author

Books

1. Y. Kambayashi, M. Arikawa, T. Kunishima, S. Konomi, H. Takada, and Y. Miyauchi. Distributed and Cooperative Media Series 4 — Hypermedia and Objectbase. Published from Kyoritsu Pub. Co. Ltd. Nov. 1995 (in Japanese).

Major Publications

1. T. Naruse, T. Saito, and T. Kunishima. On an Algorithm Obtaining an Approximated Minimum Ball Containing n Balls. Trans. IEICE, D-II, Vol. J73-D-II, No. 10, pp. 1792-1795, Oct. 1990 (in Japanese).

2. Y. Kambayashi, Q. Chen, and T. Kunishima. Coordination Manager: A Mechanism to Support Cooperative Work on Database Systems. Proc. 2nd Far-East Workshop on Future Database Systems, pp. 176-183, Apr. 1992.

3. T. Kunishima and K. Yokota. Flexible Workflow Framework for Supporting Collaborative Works. Proc. International Symposium on Cooperative Database Systems for Advanced Applications. pp. 501–508, Dec. 1996.

4. K. Yokota, T. Kunishima, and H. Nakanishi. A Framework of Query Processing for Workflow Management. Lecture Notes in Artificial Intelligence, 1997 (to appear).

5. T. Kunishima and K. Yokota. Workflow Base: A Flexible Workflow Model with Database Technologies. Under submission in Trans. IPSJ (in Japanese).

Technical Reports

1. T. Kunishima, S. Matsumoto, H. Ogino, H. Hiraishi, and S. Yajima. Computer-Aided Design using Multi Screen Graphic MCMS System. Proc. IPSJ Graphics and CAD Symposium, pp. 227-236, Nov. 1989 (in Japanese).

2. T. Naruse, T. Kunishima, and T. Saito. Consideration on Setting up Bounding Volumes. IEICE Special Interest Group on Pattern Recognition and Understandings, Technical Report No. PRU89-65, 1989 (in Japanese).

3. E. Oomoto, K. Tanaka, H. Nunokawa, T. Kunishima, M. Yoshikawa, and Y. Masunaga. An Experiment of Collaborative Electronic Publishing — Design and Implementation of Database Workbook —. IPSJ Special Interest Group on Database Systems, Technical Report No. DBS99-17, July 1994 (in Japanese).

4. T. Kunishima and Y. Kambayashi. Workflow Model on a Workflow Management System WorkFlowBase. IPSJ Special Interest Group on Database Systems, Technical Report No. DBS104-41, July 1995 (in Japanese).

5. S. Uemura, J. Adachi, M. Arikawa, Y. Kiyoki, S. Shimojo, T. Kato, I. Kojima, K. Saisho, and T. Kunishima. Researches on Multimedia Information-base Technology. IPSJ Special Interest Group on Database Systems, Technical Report No. DBS106-21, Jan. 1996 (in Japanese).

6. K. Yokota, T. Kunishima, and H. Nakanishi. Hypothetical Query Processing for Workflow Management. Proceedings of the Seminar on Logical Databases and the Meaning of Change (UPMAIL Technical Report No. 129, Uppsala University), pp. 44–49, Dagstuhl, Germany, Sep. 1996.

Convention Records

1. T. Kunishima and Y. Kambayashi. Applying a Logical Language HILOG-R to CAD Databases. 43th National Convention Record of IPSJ, Vol. 4, pp. 158-159, Oct. 1991 (in Japanese).

2. T. Kunishima, T. Furukawa, and Y. Kambayashi. Interactive Support for Designing Object-Oriented Databases. 44th National Convention Record of IPSJ, Vol. 4, pp. 169-170, Mar. 1992 (in Japanese).

3. T. Kunishima and Y. Kambayashi. Schedule Management in a Computer-Supported Cooperative Work Environment based on Da-

tabase Systems. 45th National Convention Record of IPSJ, Vol. 6, pp. 281-282, Oct. 1992 (in Japanese).

4. T. Kunishima, T. Shimada, and Y. Kambayashi. Implementation and Applications of Generalized Trigger Mechanisms for VirtualOffice. 46th National Convention Record of IPSJ, Vol. 6, pp. 247-248, Mar. 1993 (in Japanese).

5. T. Kunishima and Y. Kambayashi. A Description Method for Cooperative Activities using Dynamic Finite Automata. 46th National Convention Record of IPSJ, Vol. 6, pp. 249-250, Mar. 1993 (in Japanese).

6. T. Kunishima and Y. Kambayashi. An Inheritance Mechanism on Aggregation Hierarchies of Cooperative Works. 47th National Convention Record of IPSJ, Vol. 6, pp. 283-284, Oct. 1993 (in Japanese).

7. T. Kunishima and Y. Kambayashi. An Execution Method for Workflows Based on Activity Objects. 48th National Convention Record of IPSJ, Vol. 6, pp. 241-242, Mar. 1994 (in Japanese).

8. T. Kunishima and Y. Kambayashi. A View Function on Workflow Model. 49th National Convention Record of IPSJ, Vol. 6, pp. 247-248, Sept. 1994 (in Japanese).

9. T. Kunishima and Y. Kambayashi. Implementation of A Prototype for A WorkFlow Management System "WorkFlowBase". 50th National Convention Record of IPSJ, Vol. 6, pp. 183-184, Mar. 1995 (in Japanese).

10. T. Kunishima and Y. Kambayashi. A Study for Openness of Workflow Management Systems. 51th National Convention Record of IPSJ, Vol. 6, pp. 179-180, Sept. 1995 (in Japanese).

11. N. Segawa, M. Hiji, T. Kunishima, and K. Watanabe. Description in the Frame of Dynamism of Human Communications. 51th National Convention Record of IPSJ, Vol. 6, pp. 187-188, Sept. 1995 (in Japanese).